Reinforcement Learning-Based Path Planning for Autonomous Robots *

Dennis Barrios Aranibar[1] **, Pablo Javier Alsina**[1]

[1]Laboratório de Sistemas Inteligentes
Departamento de Engenharia de Computação e Automação
Universidade Federal do Rio Grande do Norte
Campus Universitário – Lagoa Nova 59.072-970 - Natal - RN - Brasil

{dennis,pablo}@dca.ufrn.br

Abstract. *In this paper a path planning method, based on reinforcement learning, is proposed. It was implemented for a small two-wheeled robot. This approach is inspired on potential field functions and reinforcement learning. The basic idea is to model the problem as a state-action problem and find a function that give us the value of each state, which means how good is to stay on it, in the workspace. Starting in a state and following in the direction of the gradient of the function, a robot is going to arrive to the goal state. The solution given by this method is easier to be executed by a robot with non-holonomic constraints than the solutions of the most commons methods, which generate almost always a polygonal line path.*

1. Introduction

One of the goals in Robotics is to create autonomous robots, and one of it most important problems is motion planning that can be stated as follows: How can a robot decide what motions to perform in order to achieve goals.[Latombe, 1991]
Motion planning consists of guide the robot from an initial configuration (position-orientation) to a goal configuration, in a finite time interval, without obstacle collisions. The classic way to solve this problem is splitting it in less complex subproblems. The problem is divided in path planning, trajectory generation and trajectory execution.[Pedrosa et al., 2003]
There exists a large number of methods for solving the path planning problem. Some methods require the workspace to be two-dimensional and the objects to be polygonal. The most common methods are based on road-map, cell decomposition and potential fields.[Latombe, 1991]
The principal problem of these approaches is that most of them produce collision-free polygonal line paths, and this kind of geometric paths are good to avoid collisions but not for non-holonomic robot execution. We need adaptation techniques to make this paths executable by robots with non-holonomic constraints.[Alsina et al., 2002]
In the other hand, Reinforcement Learning is learning what to do so as to maximize a numerical reward signal. The learner is not told which actions to take, but instead must discover which actions yield the highest reward by trying them. Beyond the agent and the environment, one can identify four main sub-elements of a reinforcement learning system: a policy, a reward function, a value function, and, optionally, a model of the environment. A policy defines the learning agent's way of behaving at a given time, a reward

*This work is supported by Conselho Nacional de Desenvolvimento Científico e Tecnológico CNPq/Brasil.

function defines the goal in a reinforcement learning problem and indicates what is good in an immediate sense, a value function specifies what is good in the long run, and, the model of the environment is something that mimics the behavior of the environment. For example, given a state and action, the model might predict the resultant next state and reward.[Sutton and Barto, 1998]
In this paper a path planning method is proposed, this approach is based on reinforcement learning and is inspired on potential fields methods. As potential field, this method works with a function that is used to navigate in the workspace without obstacle collisions.
First, it is important to find this function, which shows the value of being in all configurations in the configuration space, the next step is to go in the direction of the gradient of this function, but following some rules that are going to make the path easier to be executable by a robot with non-holonomic constraints.
A model of the environment is proposed in order to find paths that are executable by non-holonomic robots. A reward function is given in order to avoid obstacle collisions and get a goal position. In order to navigate in the environment without collisions an optimal policy and its value function are found by a reinforcement learning algorithm. Finally a free-collision path with non-holonomic constraints is obtained from the optimal policy to guide the robot from a initial configuration to a goal configuration.
This approach can avoid the problem of path generation without non-holonomic constraints, and give a model of the environment where a path from any configuration to the goal configuration can be found.
In section 2 the reinforcement learning-based path planning approach is presented, in section 3 the experimental results obtained from applying this method to a small two-wheeled robot are shown; a comparison between this method and the depth-first potential fields method is presented. Finally in section 4 the conclusions obtained from this work are given.

2. Reinforcement Learning-Based path Planning

The proposed method was tested in a small two-wheeled robot, in this section all the method will be explained based on characteristics of this robot. As shown in section 1, in this approach the model of the environment and a reward function must be defined, and a reinforcement learning algorithm must be selected to use.

2.1. Environment´s model

First, the state space needs to be defined, including, if possible, all the states of the robot. The state of the robot is completely defined by it configuration at a given moment. The workspace is polluted with obstacles. A useful way to work with obstacles and the robot, is working in the configuration space, that is represented in R^3. Then, the state space is the same as the configuration space (Figure 1).
Next, the possibles actions of the robot need to be defined for all states in the state space, considering their correspondent effect (the next state resulting of each action). Because the robot have non-holonomic constraints and always moves forward, five actions are defined (Figure 2): Go forward (1), go right-forward (2), go left-forward (3), turn right (4) and turn left (5).
Actions were defined relative to the present robot orientation, the next state depends on the robot orientation angle, for example if the robot is in the position (3,2) with orientation 0 and it takes the action 1, then the next state will be position (4,2) with orientation 0. But, if the robot is in the same position but with orientation $\pi/2$, taking the same action the next state will be position (3,3) and orientation $\pi/2$. In order to reduce the complexity of the program without losing efficiency, possibles next states for an action are reduced

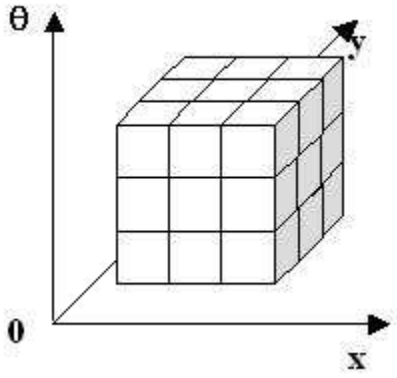


Figure 1: State space of the robot

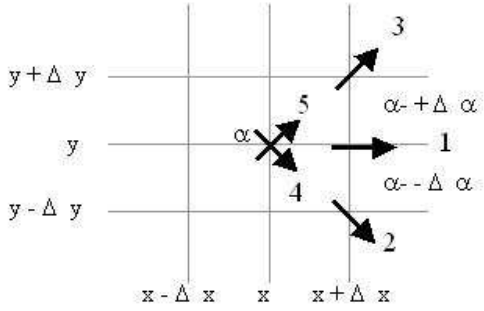


Figure 2: Actions of the robot

to eight possibilities (Figure 3) with centers in 0, $\pi/4$, $\pi/2$, $3\pi/4$, π, $5\pi/4$, $3\pi/2$ and $7\pi/4$. For example, if the state space have 36 diferents quantization levels for the angle and the robot is in a position (x,y) with orientation $\pi/5$, it is in the state (x′,y′,4), where x′, y′ and 4 represent a position in the quantized state space corresponding to the real configuration; if it takes the action 2 (go right-forward); the next state will be (x′+1,y′,3).

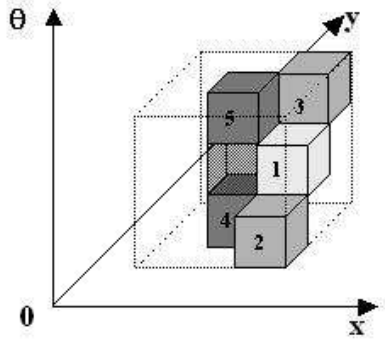
2.2. Reward function

In order to define the reward function, the next definitions are given:

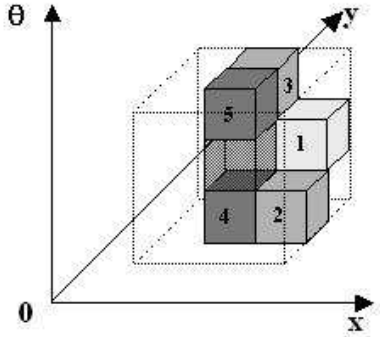
1. All the states not including a valid position (position in the work-space) are called invalid states.
2. All the states including at least one valid position (position in the work-space) are called valid states.
3. All the valid states including at least one point of an obstacle are called obstacle states.
4. The valid state that includes the final configuration is called final state.
5. All the valid states that are not obstacle states neither a final state are called free states.
6. Final state, invalid states and obstacle states are called terminal states.
7. Free states are called non-terminal states.

The immediate reward, for all states in the state space is defined as:

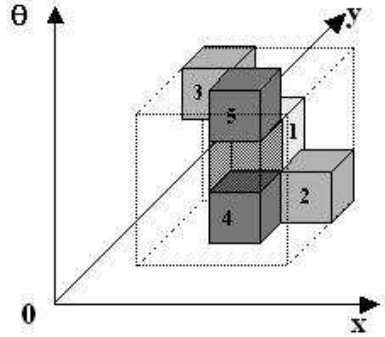
1. The reward for non-terminal states (free states) is $r = 0$.
2. The reward for terminal states is:
   - The reward for the final state is $r = 1$.
   - The reward for invalid states is $r = 0$.



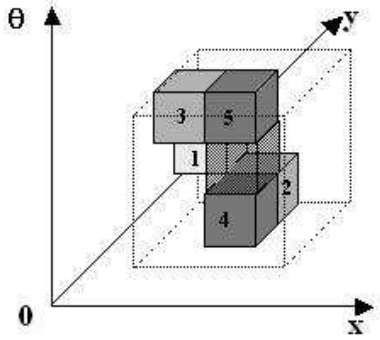
(a) $-\pi/8 < \theta \leq \pi/8$



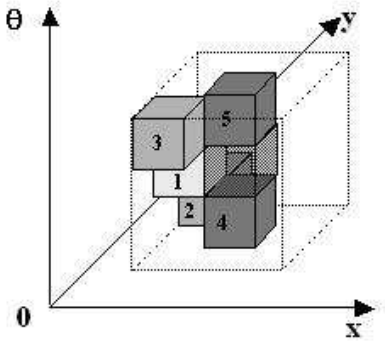
(b) $\pi/8 < \theta \leq 3\pi/8$



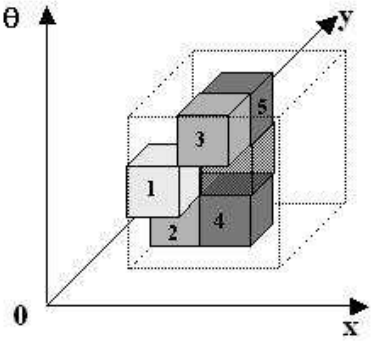
(c) $3\pi/8 < \theta \leq 5\pi/8$



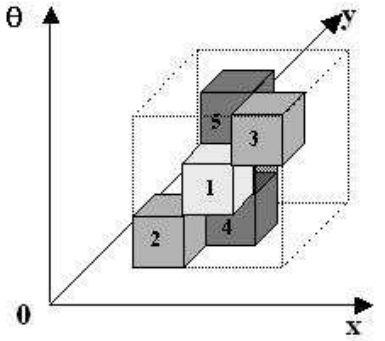
(d) $5\pi/8 < \theta \leq 7\pi/8$



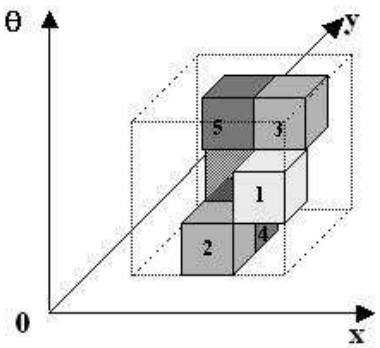
(e) $7\pi/8 < \theta \leq 9\pi/8$



(f) $9\pi/8 < \theta \leq 11\pi/8$



(g) $11\pi/8 < \theta \leq 13\pi/8$



(h) $13\pi/8 < \theta \leq 15\pi/8$

Figure 3: States after robot´s actions for non-terminal states

- The reward for obstacle states is $r = -1$.

Rewards defined above will make the path generated by this approach a free-collision path.
Because the robot has to go forward in the direction of the gradient of the value function only in non-terminal states, the state-action map given in section 2.1 is only valid for non-terminal states. In the case of terminal states all the actions will have as next state the original state.

2.3. Reinforcement Learning Algorithm Selection

It is important to use Reinforcement Learning algorithms that give good performance in terms of calculation time and quality of results. For small state spaces (number of states less than 8000) it is proposed to use dynamic programming. For medium-sized state spaces (number of states between 8000 and 60000) it is proposed to use Q-Learning. And for large state spaces (number of states greater than 60000) it is proposed to use Q-learning combined with a tool that could interpolate the value function, a useful tool for such interpolation could be a neural network. The algorithms for Dynamic Programming and Q-learning are showed in algorithm 1 and algorithm 2.
For a large configuration space and small state space it is necessary to use a method for interpolating the points not included in the solution given by the algorithm. A good method in this case is Hermite's Interpolation because the path planning method is going to return the points of the path and the orientation in each point. It means that the method is going to return the function and its first derivate, which are the only things necessaries to obtain a (2n+1)-degree polynomial, where n is the number of position-orientations to interpolate. It is recommended to interpolate two points in order to obtain third-degree polynomials.
For a large configuration state and a large state space it is not necessary to use a interpolation method because the path will be almost ready to be executed.
For a small configuration state and a small state space work as in large configuration state and large state space.

3. Experimental Results

As explained in section 2 the proposed method was implemented for a small two-wheeled robot. The robot dimensions are 19x9.5 cm, and the workspace dimensions are 108x128 cm. The path execution was implemented using the controller proposed by [Vieira et al., 2003]. Results for the proposed method, with Q-learning and Dynamic Programming algorithms, and the potential fields depth-first method[Latombe, 1991] were compared.
The robot was exposed to five situations:

1. Workspace without obstacles and the goal right in front of the robot.
2. Workspace without obstacles and the goal at the left side of the robot and with opposite orientation.
3. Workspace with an obstacle in its center and the goal behind it.
4. Workspace with two obstacles and the goal behind them.
5. Workspace with tree obstacles and the goal behind them.

The total area occupied by the obstacles in the last three situations was the same. In experiments the state space was obtained from six diferents quantization levels of the configuration space, where its three dimensions (x,y,θ) were quantified in 7,10,15,20,30 and 45 quantization levels. In the program with the Q-learning algorithm differents number of iterations were used, which are: 9000, 15000, 20000, 45000 and 75000. In the dynamic

Algorithm 1 Path Planning using Dynamic Programming

1. Initialization

   $qX \in \mathbb{N}$, $qY \in \mathbb{N}$, $qA \in \mathbb{N}$ {Quantization levels for (x, y, α) configurations)}
   $aErr \in \mathbb{R}$ {Acceptable error for calculated value function}
   $\mathcal{A}$ = All possible actions of the robot
   $V(s) \in \mathcal{R}$ and $\pi(s) \in \mathcal{A}$ {Initial Value Function and Policy}
   $\mathcal{S}$ = Configuration space using qX, qY and qA
   $\mathcal{R}_{ss'}$ = Immediate reward values for all positions in the configuration space

2. Policy Evaluation

   repeat
     $err \leftarrow 0$
     for all $s \in \mathcal{S}$ **do**
       $v \leftarrow V(s)$; Observe s' taking action $\pi(s)$
       $V(s) \leftarrow \mathcal{R}_{ss'}^{\pi(s)} + \gamma V(s')$; $err \leftarrow max(err, |v - V(s)|)$
     end for
   until $err < aErr$

3. Policy Improvement

   $PolicyStable \leftarrow true$
   for all $s \in \mathcal{S}$ **do**
     $b \leftarrow \pi(s)$; $\pi(s) \leftarrow argmax_a(\mathcal{R}_{ss'}^a + \gamma V(s'))$
     if $b \neq \pi(s)$ **then**
       $PolicyStable \leftarrow false$
     end if
   end for
   if not $PolicyStable$ **then**
     go to 2
   end if

Algorithm 2 Path Planning using Q-learning

1. Initialization

   $qX \in \mathbb{N}$, $qY \in \mathbb{N}$, $qA \in \mathbb{N}$ {Quantization levels for (x, y, α) configurations)}
   $N \in \mathbb{N}$ {Number of episodes for Q-learning algorithm}
   $Q(s, a) \in \mathcal{R}$ {Initial Action-Value Function for all states and actions}
   $\mathcal{S}$ = Configuration space using qX, qY and qA
   $\mathcal{R}_{ss'}$ = Immediate reward values for all positions in the configuration space
   $episode \leftarrow 0$ {Actual episode}

2. Finding a Policy

   repeat
     $episode \leftarrow episode + 1$; initialize s'
     repeat
       Choose a from s using policy derived from $Q(s, a)$(e.g., ϵ-greedy)
       Take action a, observe s',$\mathcal{R}_{ss'}$
       $Q(s, a) \leftarrow Q(s, a) + \alpha[\mathcal{R}_{ss'} + \gamma max_{a'} Q(s', a') - Q(s, a)]$
       $s \leftarrow s'$
     until s is terminal
   until $episode = N$
   $\pi(s) \leftarrow argmax_a Q(s, a)$

Angle Variation	Reinforcement Learning	Depth-first
$0 \leq \Delta\theta < \pi/12$	88.39 %	96.93 %
$\pi/12 \leq \Delta\theta < \pi/6$	9.14 %	2.94 %
$\pi/6 \leq \Delta\theta < \pi/4$	1.67 %	0.00 %
$\pi/4 \leq \Delta\theta < \pi/3$	0.28 %	0.00 %
$\pi/3 \leq \Delta\theta < 5\pi/12$	0.40 %	0.00 %
$5\pi/12 \leq \Delta\theta < \pi/2$	0.03 %	0.00 %
$\pi/2 \leq \Delta\theta < 7\pi/12$	0.04 %	0.13 %
$7\pi/12 \leq \Delta\theta < 2\pi/3$	0.06 %	0.00 %

Table 1: Angle variation comparison between depth-first method and reinforcement learning-based method (Based on all situations in the experiments)

Workspace type	Calculation time
Workspace without obstacles	394 ms.
Workspace with total area of obstacles $\approx 1\%$	301 ms.
Workspace with total area of obstacles $\approx 2\%$	263 ms.
Workspace with total area of obstacles $\approx 3\%$	248 ms.
Workspace with total area of obstacles $\approx 5\%$	228 ms.
Workspace with total area of obstacles $\approx 7\%$	218 ms.
Workspace with total area of obstacles $\approx 8\%$	202 ms.

Table 2: Calculation time for diferents workspace types; with Q-Learning algorithm, 15000 iterations and quantization of 15x15x15.

programming algorithm an error tolerance of 0.0000001 was used. The parameter γ used in both programs were 0.9 and parameters α and ϵ for the ϵ-greedy policy were 0.1 and 1. The first important result is about the orientation angle and its variation, which is important because is used as a measure of how easy the path can be executed by a robot with non-holonomic constraints.

As showed in table 1, the depth-first method has more percentage of variations between 0 and $\pi/12$ than the reinforcement learning method; but, instead, it has more percentage of variations between $\pi/2$ and $7\pi/12$. Analyzing table 1 and paths generated by these methods, it is observed that depth-first method has more straight line sub-paths but has abrupt curves too, which are not good for execution by a robot with non-holonomic constraints; in the other hand, paths generated by reinforcement learning-based method has more smooth curves that are easily executed by robots with this kind of constraints.

Next, it is important to talk about the calculation time of the proposed method. As showed in table 2, the calculation time of the method decrease as the total area occupied by the obstacles increase. It occurs because the Q-learning algorithm stops an iteration when it achieves a terminal state, and states with obstacles points are considered as terminal states, then the time for each iteration is reduced. Another reason is because the value function, which is the goal of a reinforcement learning algorithm needs less time to converge, it occurs because the value of terminal states is always the value of the immediate reward and it converges quickly, then the probability of having an state with a not small error in the value function decrease. Table 2 shows the calculation times for Q-learning algorithm with 15000 iterations and a quantization of fifteen levels for each x,y and θ. It can be noted that the more total area occupied by obstacles the workspace has, the less calculation time this approach needs.

In section 2.3 was proposed to use dynamic programming for small state spaces and Q-Learning for medium-sized state spaces. This proposal is verified in figure 4 that shows

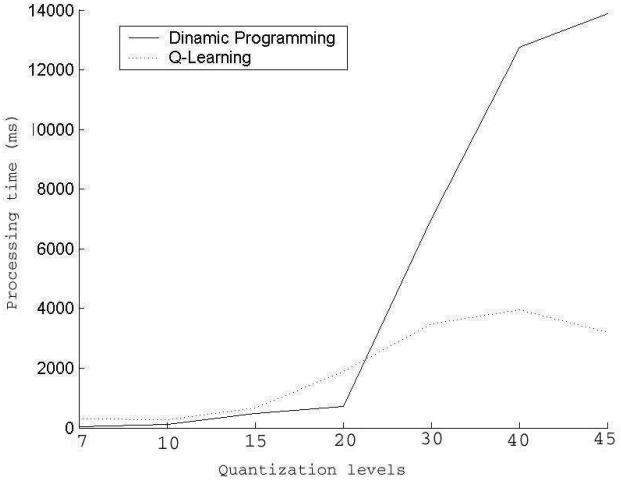


Figure 4: Calculation time comparison between Q-Learning and Dynamic Programming algorithms

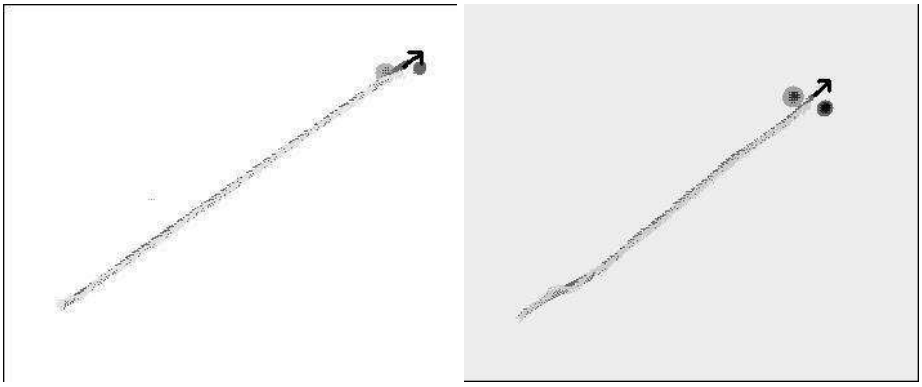
Final Orientation Error	Reinforcement Learning	Depth-first
$0 \leq \Delta\theta < \pi/12$	80.43 %	20.00 %
$\pi/12 \leq \Delta\theta < \pi/6$	13.04 %	20.00 %
$\pi/6 \leq \Delta\theta < \pi/4$	2.17 %	20.00 %
$\pi/4 \leq \Delta\theta < \pi/3$	0.00 %	0.00 %
$\pi/3 \leq \Delta\theta < 5\pi/12$	0.00 %	0.00 %
$5\pi/12 \leq \Delta\theta < \pi/2$	4.34 %	20.00 %
$\pi/2 \leq \Delta\theta < 7\pi/12$	0.00 %	0.00 %
$7\pi/12 \leq \Delta\theta < 2\pi/3$	0.00 %	20.00 %

Table 3: Comparison of Orientation Error in goal configuration using a path generated by Depth-first and Reinforcement Learning-based Methods

the calculation time in milliseconds for diferents quantization levels that are 7, 10, 15, 20, 30, 40 and 45 levels for each x, y and θ. It was verified that for quantization levels that result in less than 8000 states is better to use dynamic programming and for quantization that result in more than 8000 states is better to use q-learning. In quantization levels superior to 45 levels for each x, y and θ this two algorithms have high calculation times. Then, in this cases, a interpolation tool for the value function is needed.

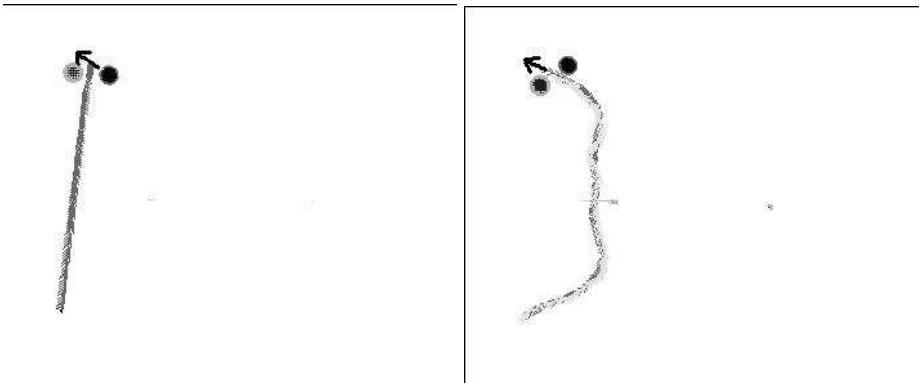
Another important result was found analyzing the final orientation. Table 3 shows the orientation error at the goal position, which is the difference between the goal orientation and the final orientation attained by the robot after following a path generated by the depth-first method and reinforcement learning-based method. As showed in this table, the path generated by the reinforcement learning method assures, almost in all cases, that the robot achieves the goal position with low orientation error. In the other hand, the depth first method doesn't assure anything about it.

Experimental results of execution of planned paths are shown in figures 5, 6, 7 and 8, where the goal configuration and the final configuration achieved by the robot are analyzed. The goal position orientation is represented by the black arrow. The robot is represented by two circles. The robot position (x,y) is located at the middle point of the straight line segment between the centers of both circles. The robot orientation θ is the angle between the perpendicular to this segment and the horizontal line. The generated path is represented in dark gray and the executed path is represented in light gray.



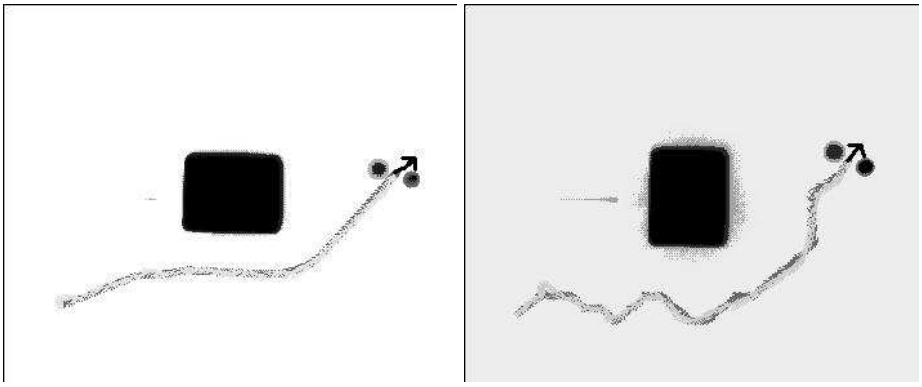
(a) Depth-First (b) Reinforcement Learning

Figure 5: Experimental results for a workspace without obstacles and the goal right in front of the robot



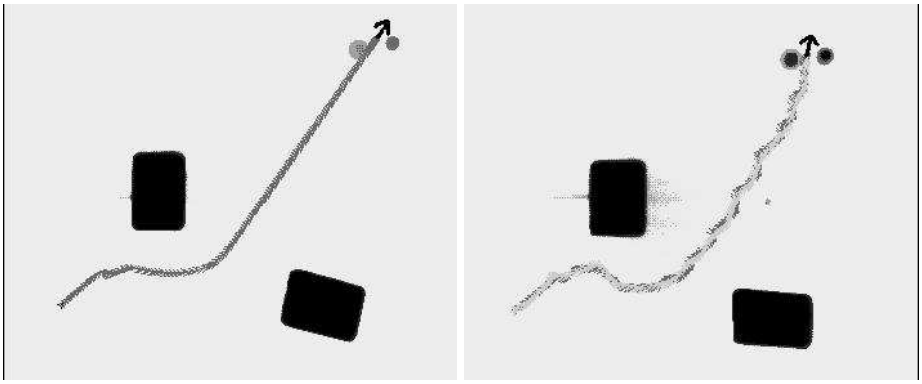
(a) Depth-First (b) Reinforcement Learning

Figure 6: Experimental results for a workspace without obstacles and the goal at the left side of the robot and with opposite orientation



(a) Depth-First (b) Reinforcement Learning

Figure 7: Experimental results for a workspace with an obstacle in its center and the goal behind it



(a) Depth-First (b) Reinforcement Learning

Figure 8: Experimental results for a workspace with two obstacles and the goal behind them

4. Conclusions

This paper introduces a new approach for path generation in the motion planning problem. Paths generated by Reinforcement learning-based method are more easily executed by robots with non-holonomic constraints than paths generated by the most commons methods, which generate almost always a polygonal line path.
The more total area occupied by obstacles the workspace has, the less calculation time this approach needs.
For small state spaces is better to use dynamic programming than Q-Learning and for medium-sized state spaces is better to use Q-Learning than dynamic programming.
The path generated by the reinforcement learning method assures, almost in all cases, that the robot achieves the goal position with low orientation error.

References

Alsina, P. J., Gonçalves, L. M., Vieira, F. C., Pedrosa, D. P., and Medeiros, A. A. (2002). Minicurso: Navegação e controle de robôs móveis. Congreso Brasileiro de Automática - CBA 2002.

Latombe, J.-C. (1991). *Robot Motion Planning*. Kluwer Academic Publishers.

Pedrosa, D. P., Medeiros, A. A., and Alsina, P. J. (2003). Um método de geração de trajetória para robôs não-holonômicos com acionamento diferencial. In *Simpósio Brasileiro de Automação Inteligente*, pages 840–845, Bauru, SP, Brazil.

Sutton, R. S. and Barto, A. G. (1998). *Reinforcement Learning: An introduction*. The MIT Press.

Vieira, F. C., Medeiros, A. A., and Alsina, P. J. (2003). Dynamic stabilization of a two-wheeled differentially driven nonholonomic mobile robot. In *Simpósio Brasileiro de Automação Inteligente*, pages 620–624, Bauru, SP, Brazil.