Soccer Robot Navigation in Grid Environments Based on Reinforcement Learning Algorithms

Eduard P. Enoiu∗
Mälardalen University
Västerås, Sweden
eeu09001@student.mdh.se

Raluca Marinescu†
Mälardalen University
Västerås, Sweden
rmu09001@student.mdh.se

ABSTRACT
Autonomous mobile robots have been extensively studied not only as an element of industrial and home automation, but also as a test bed in Robocup competitions to academically establish the achievement of artificial intelligence. One of the essential and critical research areas in autonomous robotics is the learning ability which supports robots to autonomously navigate and adapt to a given environment without human guidance.

In this paper, we introduce a control architecture for learning mobile robots to navigate in a grid-based model of the environment, and describe our experiments using it to learn control policies for a simple obstacle avoidance task.

1. INTRODUCTION
Past years have engrossed a great deal of interest in autonomous robots. The development is partially due to the increasing demands for utilizations of robots in human society, for example in industrial automation, and space exploration. Developing such complex autonomous robots requires research in areas of control, reasoning, and perception. Existing and upcoming research efforts in this area will endure to attempt for improved dependability and better autonomy. One of the main challenges in developing navigation mechanisms, is the problem of combining reactive with deliberate control[13], [5].

Reactive control, often called local navigation, is an important capability for mobile robots to learn or plan a path between two specific locations using the current sensory inputs lacking prior knowledge of the environment. Reactive control strategies have been suggested and applied in different approaches [9] for navigation tasks including obstacle avoidance, and wall following. Using sonar sensors to model the environment occupancy maps have been built, introducing various path-planning methods like the one presented in [4].

Although reactive control can adjust to many types of environments it is susceptible to errors because of no prior information about the environment. One of the solutions is to supply an updated model of the global environment by using the sensory inputs for exploration purposes. Once the environment is modelled in a map world, the robot has a complete view of the environment. There are many studies on environment models, such as grid-based maps [6] and road maps (Voronoi diagrams and visibility graphs) [8], [16]. Among them, grid-based map method is the most popular way to represent a given environment. In this method, the workspace is divided in two kinds of grid cells containing values of the information that is recognized as free space or obstacle, based on the actual position of the obstacles. By using this representation method, the path can be demarcated as a sequence of cells which begins at the start cell and ends at the endpoint cell. Grid-based method is used to gain simplicity on building, representation, and maintaining indoor environments. Using this kind of representation method, in the 80s and 90s, behaviour based control policies were proposed [11]. These systems make usage of different forms of internal representations and perform computations on them in order to decide what path to choose. Similarly, various implementations are based on fuzzy logic and neural network [20] or on Reinforcement Learning (RL)[2]. RL is a machine learning technique that defines a certain scenario, where a robot attempts to improve its behaviour by taking actions and receiving rewards or punishments. Because RL enables a robot to learn autonomously from its own environment and experience, it is well suited framework for learning control policies for mobile robots. Nevertheless, it is often difficult for a programmer to translate knowledge from environment and to plan how to complete a task into terms that are useful for the robot. Grid-based maps are simple to be created and updated, and can be easily implemented with RL algorithms.

In this paper, we focus on the application of learning methods within the Robocup research field. RoboCup was founded in 1997 in order to establish a research testbed for autonomous intelligent systems [7]. There are several leagues, where each one is focusing on different aspects of robotics. In every league teams of robots are contending in a soccer-like setting. In 2010, Mälardalen University es-

*Master in Intelligent Embedded System
†Master in Intelligent Embedded System
established a new generation of soccer-playing robot named Saga,¹ that strives to participate in Robocup’s Middle Size League. The aim of the project is to build highly competitive robots, and in particular a control architecture using reinforcement learning. The purpose of this paper is to give evidence to the thesis that reinforcement learning algorithms are a valuable asset to the growing toolbox of robot navigation and to show how the basic framework can be adapted to a control architecture made for soccer-playing robots.

To address all the aspects described above, in Section 2, we propose a Navigation Architecture, presenting an integrated navigation unit for learning control policies in autonomous robots. First, the grid map paradigm is presented for modelling the environment. Second, we briefly describe RL, and introduce a framework for effectively using it on mobile robots. Third, a motion control system is proposed based on an omnidirectional robot platform, in order to translate robot’s movement along any desired path. In Section 3, we verify our presented method using an obstacle avoidance task in simulation, before concluding the paper in Section 4.

2. NAVIGATION ARCHITECTURE

The Navigation Architecture is intended as a meaningful basis for navigation purposes in mobile robots. We consider a three level representation of the architecture as shown in Figure 1. These three levels are: the Grid Map level, the RL framework and the Motion System. The coordination between the first level and the third one is guaranteed by the second one. Every level has a required functionality that is explained in the next sections.

Basically robot navigation calls upon three processes:

- **Map-learning**, which is the process of memorizing the data acquired by the sensors in a suitable grid map representation.
- **Localization**, which is the process of deriving the current position of the robot within the grid map.
- **Control Policies or Path-learning**, which is the process of choosing a sequences of actions to reach a goal, given the current position.

Localization and Map-learning are symbiotic methods and are using grid maps for localizations purposes. The Map-learning method, often referred to as SLAM, i.e. simultaneous localization and mapping, is a very dynamic area of research in the robotics community. Because the environment recognition method based on sensor inputs is outside the paper’s goal, it will be described only as an abstract attribute. We will assume that the sensors input data are acquired in a system named Sensor Fusion, using sensor observations, that provides all the sensor information required for a prototype of the environment. Therefore, we accept throughout this paper, that sensor mapping is available and is used for building a global map of the environment named Grid Map.

![Figure 1: Overall Navigation Architecture.](image)

Once the map is provided, an estimate position of the robot inside the map is available, and a goal position is known, the robot should be capable to navigate from its current position to the goal position. This control policy relies in our paper on the RL Framework that allows the robot to learn from the map how to reach a goal position. Basically using the current map states, the RL Framework learns a control policy, and the Motion System is translating the commands into actual motion trajectories.

2.1 Grid-based Maps

Grid-based maps are a fundamental paradigm for modelling indoor environments that employ probabilistic sensor interpretation models. Therefore, it supports robust mapping and navigation strategies and allows a variety of robotic tasks to be addressed through operations performed directly on the grid map.

The grid-based approach [4] represent the world environment with evenly spaced grids, where each grid cell may specify whether the equivalent region of the environment is free or occupied space. For our simulation testing purposes it is an acceptable trade-off.

The grid-based maps considered here are discrete, two dimensional occupancy grids. Each grid cell $(x, y)$ in the map has a related value $P_o(x, y) \in [0, 1]$ that measures the individual certainty that the cell is occupied. The values $P_o(x, y)$ are updated based on the sensor information such as data from a stereo-vision system [10]. Suppose the threshold value is $\delta \in (0, 1)$, then there is the condition of $P_o(x, y)$

¹See the web page http://robotik.se/Saga/ for more information about Saga Robotic Platform
for the decision if the cell \((x,y)\) is occupied:
\[
\begin{cases}
    P_o(x, y) < \delta & \text{cell } (x,y) \text{ is Empty} \\
    P_o(x, y) > \delta & \text{cell } (x,y) \text{ is Occupied} \\
    P_o(x, y) = \delta & \text{cell } (x,y) \text{ is Unknown.}
\end{cases}
\]

Figure 2 (a) shows how the probability of grid-cell occupancy is determined using perception. A mobile robot with a range of sensors such as sonar sensors, infrared light sensors, or cameras [10] can perceive the context around it while moving in an unknown environment, and learn by updating the grid-based maps. Usually the three major components of building grid-based maps are:

- **Interpretation.** Sensors observations are represented in occupancy values.
- **Integration.** Several sensor observations are combined over time to return a single occupancy value.
- **Position estimation.** The position of the robot is constantly traced.

We assume for simplicity that the robot operates in an environment with only stationary obstacles. A typical navigation task in a stationary grid-based world is described in Figure 2 (b), which depicts how the mobile robots work with a pre-built grid-based map.

Taking all these into consideration, the next section will introduce the capability of a robot to learn knowing that the environment can be perceived as a discrete set of grid cells.

### 2.2 RL Framework

RL is a machine learning paradigm that offers algorithms for optimizing an agent’s behaviour by using the capability of the agent to sense rewards from the environment. An agent in our approach can be considered a robot that interacts with the environment by its sensors and actuators (motors).

There are several algorithms suggested in the literature [15]; we have used probably the most popular Q-learning algorithm because of its programming simplicity [18].

![Figure 2: Grid-based approach: (a) grid-cell occupancy detection using sensors, and (b) a typical task in a grid-based world.](image)

![Figure 3: Standard Model of RL.](image)
Algorithm is to define the Q-values. \( Q^*(s, a) \) is the expected reward if the agent takes action \( a \) in state \( s \) and follows policy \( \pi \):

\[
Q^*(s, a) = r(s, a) + \gamma V^*(s'),
\]

where \( s' \) is the state, in which the agent is by taking action \( a \) in state \( s \).

<table>
<thead>
<tr>
<th>Input: ( S, A ) and ( Q(s, a) )</th>
<th>Output: ( Q(s, a) )</th>
</tr>
</thead>
<tbody>
<tr>
<td>Observe current state ( s );</td>
<td></td>
</tr>
<tr>
<td>repeat</td>
<td></td>
</tr>
<tr>
<td>Choose and execute action ( a );</td>
<td></td>
</tr>
<tr>
<td>Observe reward ( r );</td>
<td></td>
</tr>
<tr>
<td>Observe new state ( s' );</td>
<td></td>
</tr>
<tr>
<td>( Q(s, a) \leftarrow r + \gamma \max_{a'} Q(s', a') );</td>
<td>( s \leftarrow s );</td>
</tr>
<tr>
<td>until terminal state is reached;</td>
<td></td>
</tr>
</tbody>
</table>

**Algorithm 1:** Q-learning algorithm

Q-learning is shown in Algorithm 1. It guarantees convergence to optimal values of \( Q^*(s, a) \), if \( Q \)-values are represented without any function approximations, rewards are bounded and every state-action pair is visited infinitely often. To satisfy the last condition, every action has to be chosen with non-zero probability. Probability \( P(a|s) \) of choosing action \( a \) in state \( s \) is defined as [12]:

\[
P(a|s) = \frac{k Q(s, a)}{\sum_j k Q(s, a_j)},
\]

where \( k > 0 \) is a constant that determines exploitation-exploration rate. Big values of \( k \) will make the robot to choose actions with above average values. On the other hand, small values will make the robot to select actions arbitrarily.

Optimum values for \( V^*(s) \) can be acquired from \( Q^*(s, a) \) when the next equality is true:

\[
V^*(s) = \max_{a'} Q(s, a').
\]

We can now examine the complexity of reinforcement learning algorithms for the path planning task. We first consider the complexity of reaching a goal state for the first time. The worst-case complexity of reaching a goal state with RL (and terminate the learning there) offers a lower bound of finding the shortest path, as this cannot be done without knowing where the goal states are. By "worst case" we mean an upper bound on the total number of phases for a primarily informed learning procedure.

We assume that a Q-learning algorithm is zero-initialized (all Q-values are zero initially) and operates on the goal-reward task. The first Q-value that changes its own value is the Q-value of the action that indicates the robot to a goal state. For all other actions, no information about the state space is known and all Q-values persist to be zero. Since the action selection phase has no information on which to base its decision, it's performing in random exploration, and the robot has to choose actions according to a uniform distribution. Then, the robot reaches a goal state ultimately, but the average number of steps required can be exponential in \( n \) number of states [19].

One of the difficulties of using Q-learning for mobile robots is the problem of integrating previous knowledge into the learning process. Q-learning can acquire information about its environment by taking actions and observe their effects. If we assume no prior knowledge about the environment, the agent is forced to choose, as mentioned earlier, more-or-less arbitrary actions. With more information, in the form of rewards, Q-learning can improve its approximation of the value function. We will address this problem in the next section.

2.2.2 Learning Process

Without certain prior knowledge of the environment, the learning process is almost surely condemned to be unsuccessful. If we assume that the grid map is not complete enough to adequately control the robot, we can address this problem by supplying an exploration control policy based for example on a human controlling the robot with a joystick. In soccer matches from Robocup Middle Size League, where no human control are accepted, it will be best to explore the environment during the game with an autonomous control policy.

By splitting the learning into two phases, we make sure that the robot will be capable of finding reward-giving states, and the learning will not stall. This exploration stage is depicted in Figure 4 as **Phase 1 Learning**. The RL system will passively watch the generating states. It is important to note that we are not trying to learn the trajectories generated by the supplied policy; but simply generating experiences to fill the grid map. Alternatively for our simulation experiments
we will neglect this phase working only with pre-build maps, and predefined obstacles.

Q-learning stage can start after the map is complete enough to control the robot. This stage is shown in Figure 4 as Phase 2 Learning. During this step Q-learning algorithm is computing a sequence of actions based on the generated grid map. Learning in episodic tasks requires a very large number of learning repetitions, and therefore can result in wear of parts if this stage is done on the real robot. Therefore, we will choose to do the learning phase in the simulator. Learning in simulation has further benefits as well. Information can be simply stored from simulation runs and referred to later [14]. Nevertheless, there are some shortcomings by doing this in simulation. No matter how good a model of the environment is, it can never be flawless. A learning policy in simulation may not operate correctly in a real environment because of any number of limitations in the model.

Another problem is that rewards match up to real events in the environment. For example, for an obstacle avoidance task, the robot might acquire a reward of 100 for reaching the goal, and -100 for hitting the obstacle. In principle, this is all that is needed for the robot to learn the optimal policy. Usually a reward function can be designed as a distributed function with 0 values on most grid cells, expect for a few places (the obstacles and goal in the above example). However, another way to design the reward function is to take in account, for example, the distance to the goal or the distance to the obstacles. This type of reward functions give more data after each action but are more problematic to build. In this work, we will mostly be interested in distributed reward functions, since they are generally much simpler to design.

Once the learning has reached an optimal policy, it can be brought to the real robot for navigation by actuating the motors using the motion system. The motion system is described in the next section.

2.3 Motion System

Once the learned policy is generated by the RL framework, the motion system’s job it is the translation of the learned motion along any desired path. The solution is trying to reduce the complexity of the motor control algorithm and to simplify the robot’s motion by using the advantages of an omnidirectional drive robot. In contrast to differential drive mobile robots, omnidirectional mobile robots are capable of driving in any direction not depending on their actual heading. For a real implementation of an omnidirectional robots please refer to [1]. Next we present the general principles involved in translating the low level actions into motor contributions for driving the robot along a path.

Three motors are mounted with 120° between them, aligned in an equilateral triangle scheme as it is shown in Figure 5. The wheels used are a variation of the so-called Swedish wheels, which use rollers with a rotation direction which is neither parallel nor perpendicular to the motor axis.

It was assumed that the front of the robot (opposite to the back motor position as shown in Figure 5) represents 0° direction. The wheel driving direction is perpendicular to the motor axis. We consider that the three wheels driving directions of the robot are 150° for \( M_{\text{back}} \), 30° for \( M_{\text{right}} \) and 270° for \( M_{\text{left}} \). Based on this configuration, some computations need to be done in order for the robot to drive along a path.

To drive the robot in a desired direction it is necessary to calculate every motor’s contribution to the movement. This contribution is given by two variables: motor rotation direction and velocity.

Algorithm 2: Motor contribution for Motion System

Algorithm 2 shows a simple computation for linear motion that uses the cosine of desired motion direction’s angle projected on each wheel drive direction multiplied by the velocity. The velocity and angle are given as an input directly, or by using the desired coordinates of the point in the grid map and the desired time to get in that point.

It is possible to control an omnidirectional robot perfectly with the previously introduced method if the friction between the wheels and the floor is infinite. However, in the real world the friction of the robots is limited. Therefore, we refer the reader to [3] for a thorough description of the motion system and other complex motions.

3. PRELIMINARY RESULTS

To demonstrate the feasibility and effectiveness of our navigation architecture, some preliminary experiments were carried out by computer simulation. As shown in Figure 6, the simulator implements a simple model of the soccer field used in Robocup competitions [7]. We assume that in this envi-
environment we have to learn a simple obstacle avoidance task using a pre-build grid map. This grid map is used as an input for the RL Framework. After finding the optimal policy, the motion system is used for translating the commands into actual trajectories.

3.1 Simulation Environment

As mentioned earlier, we use grid map to present the environment. Assuming that the space in which the robot moves is a limited area with obstacles in two-dimensional plane, and with the upper left point as the origin, the grid map is extended by horizontal X-axis and vertical Y-axis. A rectangular environment is divided as follows: the x-axis is divided equally into m parts and so is the y-axis. As a result we have a grid m x n. In Figure 7 a two-dimensional finite grid is shown. This grid is overlaid on the top of a simulated Robocup standard field. Every position on the grid is a possible state for our robot, and moving to a connected grid is the action the robot can take.

Due to the omnidirectional drive of the robot, the complete action space contains paths in all 360° direction. However, RL approaches only allow finite and –usually– very small sets of actions. Therefore, we use a discretization of the complete action space described in Section 2.3, i.e. the robot is permitted to select one of eight directions organized in 45° angles, as showed in Figure 7. After selecting one of the eight actions, the motion system is updated appropriately to achieve the desired action.

For a successful simulation it is necessary to state some basic assumptions:

- We use grid maps with a fix start point, two players (obstacles) and a goal state (the ball in our case);
- The purpose of the robot is to arrive at the final goal avoiding obstacles (go to the ball avoiding other players);
- The robot can move up, down, left, right, and diagonal directions and select one motion from 8 actions autonomously.
- Goal arriving reward is 100 and obstacle hit penalty is -100.

3.2 Experiments

As mentioned in Section 3.1, the task for testing the control architecture is an obstacle avoidance problem. The goal is to have the robot drive to the goal state, while avoiding the obstacles in the environment. In all of the experiments reported in this section, two obstacles were used. We assume that obstacles occupy one grid cell.

The robot explores every state until it reaches the goal. We call each exploration an episode. In one episode the agent will move from initial state until the goal state. Once the agent arrives at the goal state, the program goes to the next episode.

The parameter’s settings for the RL framework are: exploration policy $\epsilon$-greedy, $\epsilon = 0.2$, discounted factor $\gamma = 0.8$, all the Q values are initialized as 0. Motion system settings: translation speed of the robot is fixed and only linear movement is used. Figure 8 shows the experimental results of navigation in several grid environments in the form of a learned trajectory of the robot for the obstacle avoidance task. The framework is capable of learning good control policies as depicted by the robot’s trajectory shown as a red line in Figure 8.

The size of the Q-function grows exponentially in order to produce accurate control actions, which also brings a long convergence time from 22 minutes for the smallest grid environment test (Figure 8(c)) to 58 minutes for the largest environment test (Figure 8(a)). The algorithm needs approximately 15000 learning runs until it finds an optimal
policy. In our simulations we have not paid much attention to the efficiency of the RL framework, but validated only the effectiveness of RL in the control of mobile robots in grid environments.

4. CONCLUSIONS AND FUTURE WORK
In this paper, we have introduced a control approach for an omnidirectional robot using RL. We provided a simple representation of the environment using grid-based maps, and combined this with an integrated control scheme for an omnidirectional robot. In order to learn a correct control policy, the RL framework can have the control of the robot only after the grid-map is incorporated with knowledge from the real environment.

The control architecture was tested in a simulator using different grid environments in order to establish some accurate control patterns for navigation purposes. Due to the fact that Q-learning requires a large amount of training episodes, our RL framework has been evaluated, so far, only in simulation. The simulation experiments based on our approach demonstrate the effectiveness of the proposed control architecture. All the software implementations were done in Ada programming language and GTKAda toolkit, and the source code is available online\(^2\). The results show that the integration of RL can greatly help an autonomous robot to acquire a successful obstacle avoidance task and we believe that our framework shows the premises of using RL techniques on a real omnidirectional soccer playing robot.

In future work, we will investigate a number of extensions to the methods proposed here, and begin to implement them on physical robotic systems as opposed to computer simulations. We believe it will be straightforward to derive the functional analogues of the Navigation Architecture. Also to reach the goal of learning on real hardware a lot of work has to be performed, especially on speeding up the learning process.

5. ACKNOWLEDGMENTS
The work was done for Saga Robotic Project from Mälardalen University, Sweden. The authors wish to thank Prof. Lars Asplund for the helpful discussions and comments.

6. REFERENCES


