Automatic Robot Navigation Using Reinforcement Learning

Master thesis

Department of Computer Science
Faculty of Mathematics and Natural Sciences
University of Groningen

Submitted by: Amirhosein Shantia
Student number: s1951289

First supervisor: Prof. Dr. Michael Biehl
Second supervisor: Dr. Marco Wiering

11/29/2011

Table of Contents

List of figures

Abstract

It is extremely difficult to teach robots the skills that humans take for granted. Understanding the robot's surrounding, localizing and safely navigating through an environment are examples of tasks that are very hard for robots.

The current research on navigation is mainly focused on mapping a fixed and empty environment using depth sensory data and localizing the robot location based on robot odometry, sensory input and the map. The most common navigation method that is widely used is to map the environment using a 2D laser range finder and localize the robot by using iterative closest point algorithms. There are also studies on localization and mapping the environment using 3D laser data and the scale invariant feature transform to correct the robot odometry. However, these methods heavily rely on the precision of the depth sensors, have poor performance in outdoor environments, and require a fixed environment during training.

In the presented method, the robot brain organizes a set of visual keywords that describe the robot’s perception of the environment similar to that of human topological navigation. The results of its experiences are processed by a model that finds cause and effect relationships between executed actions and changes in the environment. This allows the robot to learn from the consequences of its actions in the real world. The robot is resistant to non-major changes in the environment during training and testing phases. More specific, the robot takes several pictures from the environment with an RGB camera during the training phase. The raw images will be processed using the histogram of oriented gradients method (HoG) to extract salient edges in major directions. By using clustering on HoG results, similar scenes will be clustered based on visual appearances. Furthermore, a world model is made from the observations and actions taken during training. Finally, during testing, the robot selects actions that maximize the probability to reach its goal using model-based reinforcement learning algorithms. We have tested the method on the pioneer 2 robot in the AI department's robotic lab to navigate to a user selected goal from its initial position.

Chapter 1

1. Introduction

It is extremely difficult to teach robots the skills that humans take for granted, for instance, the ability to orient the robot with respect to the objects in the room, and to memorize and reconstruct a three dimensional scene. In addition, navigating and localizing, responding to sounds, interpreting speech, and grasping objects of varying sizes, textures and fragility count as difficult robotic tasks. Even something as simple as telling the difference between an open door and a window is a complex task for a robot. Another obstacle for the development of robots is the high cost of hardware such as sensors that enable a robot to determine the distance to an object as well as motors that allow the robot to explore the world and manipulate an object with both strength and delicacy. But prices are dropping rapidly. In South Korea the Ministry of Information and Communication hopes to put a robot in every home there by 2013. The Japanese Robot Association predicts that by 2025, the personal robot industry will be worth more than $50 billion a year worldwide, compared with about $5 billion today (Gates, 2007).

A focus to develop service and assistive robot technology with high relevance for future personal applications is necessary. The focus lies in domestic and urban service robotics that require Self organizing brains, Human-Robot-Interaction and Cooperation, Navigation and Simultaneous Localization and Mapping (SLAM) in dynamic environments (Thrun, 1998) (Weng, et al., 2001) (Leonard & Durrant-Whyte, 1991), Computer Vision and Object Recognition under natural light conditions, and Object

Manipulation. The first expectation from a complete autonomous robot is the ability to navigate autonomously in a changing environment while maintaining safety. Therefore, in this thesis, we focus on robot navigation which is one of the most important parts of a robotic framework.

1.1. Background

In this section we present a brief overview of navigation in robotics, histogram of oriented gradients, and reinforcement learning methods.

1.1.1. Navigation

For any mobile device, the ability to navigate in the environment is the most important required capability. Staying in healthy operational mode comes first, but if any tasks are to be performed that relate to specific places in the environment, navigation is a must and is one of the most important tasks in daily domestic activities. In the following, we will present an overview of navigation systems and try to identify the basic blocks of a robot navigation system, types of navigation systems, and have a closer look at its related components.

Navigation is the ability to understand the current position and to be able to plan a path towards some goal location. In order to navigate in an environment such as a house, the robot or any another mobile device requires somehow a map of the environment and the ability to interpret that representation.

Navigation can be defined as the combination of the three fundamental competences:

1- Self-Localization
2- Path Planning
3- Map-Building and Map-Interpretation

Map in this context denotes any mapping of the world onto an internal representation.

Robot localization denotes the robot's ability to understand its own position and orientation within the frame of reference. Please note that this localization does not necessary mean the exact metric position on the environment map. Information that connects the location or builds a partial map is also sufficient. This is the case in humans, we do not map our environment precisely, but we connect our received visualization of the environment and extract a partial map out of it.

Path planning is effectively an extension of localization. The robot should be able to know how to reach a goal state from its current position. Map building can be in the

shape of a metric map or any notation describing locations in the robot frame of reference.

The most popular type of localization method, largely used in domestic service robots, is probabilistic models of the robot's motion control where the robot has probabilistic motion models and uncertain perception models. Integrating these two probability distributions using, for example, Kalman or particle filters, gives us the real location of the robot (Smith & Cheeseman, 1986). By using 2D and 3D planar mapping as an extension, the performance of such systems increase significantly. In (Thrun, 2002), the author reviews methods to solve 2D SLAM, such as maximum likelihood estimation (Frese & Hirzinger, 2001), (Folkesson & Christensen, 2003), expectation maximization (Thrun, Fox, & W., 1997), and extended Kalman filter (Dissanayake, Newman, Clark, Durrant Whute, & Csorba, 2001). One main problem with these approaches is that the observations and maps are built manually from earlier information concerning the environment's geometry, appearance and topology. For example, in some studies, (Simmons & Koenig., 1995) and (Tomatis, Nourbakhsh, & Siegwart, 2003), the environment geometry is standardized.

Another localization method which is very popular in the middle-size soccer RoboCup league is based on the global appearance from omni directional-camera images (Zivkovic, Bakker, & Krose, 2005) (Booij, Terwijn, & Zivkovic, 2007) (Goedeme, Nutting, Tuytelaars, & van Gool, 2007) (Valgren, Duckett, & Lilienthal, 2007). Images are distinguished by change in regions or points of interest, and the localization is done by calculation of similarity in the distances between points of interest. These approaches are based on image-appearances to segment the environment, taking advantage of recognizing spots from distant locations with full view images. However, similar to probabilistic models, a standard environment, and manual training is required.

All these methods only try to solve the localization problem in navigation schemes. Even after localization, navigating to different goal locations is a complex task. Methods are required to deal with localization uncertainties and external forces such as new obstacles and changes in the environment. In our proposed method we tackle localization and navigation at the same time by connecting the topological information with reinforcement learning.

1.1.2. Histogram of Oriented Gradients

A popular method in machine vision is the use of histograms of oriented gradients which is based on histograms of image gradient orientations in a dense grid. The idea is that if

we divide a picture into a dense grid and calculate the normalized histograms of oriented gradients, we will have a special code. Since the code is based on edge magnitudes and orientations of these sub images, it is rarely possible that two different pictures give the same edge information and code, even without precise knowledge of the corresponding gradient or edge positions. This is implemented by dividing the image window into small regions (cells). For each cell, we calculate a local 1-D histogram of gradient directions or edge orientations over the pixels of the cell for the eight major directions. The combined histogram entries form the representation of each image. For better robustness against illumination, shadowing, etc., it is also useful to contrast-normalize the local responses before using them. We will refer to the normalized descriptor blocks as Histogram of Oriented Gradient (HOG) descriptors (Dalal & Triggs, 2005).

1.1.3. Reinforcement Learning

Machine learning is programming to optimize a performance criterion using example data or previous observations. Learning a model with partially defined parameters is the execution of a computer program to optimize the parameters of the model using the training data or previous observations. Machine learning uses the theory of statistics in building mathematical models, because the main task is making inference from a sample. In applications such as navigation, grabbing, and exploration, the output of the system is a sequence of *actions.* In such a case, a single action is not important; what is important is the *policy* that defines the sequence of correct actions to reach the goal given the current state of the environment. Such learning methods are called *reinforcement learning* algorithms (Alpaydin, 2004) (Kaelbling, Littman, & Moore, 1996). In reinforcement learning, the learner is a decision-making agent that takes actions in an environment and receives reward (or penalty) for its actions in trying to solve a problem. After a set of trial-and error runs, it should learn the best policy, which is the sequence of actions that maximizes the total reward (Sutton & Barto, 1998). One of the most famous methods of completing tasks in robotics is the use of behaviour based models (Arkin, 1998). Each behaviour requires a sequential set of actions to be completed and reinforcement learning is the best candidate for such systems.

1.1.4. Automatic Navigation Using Reinforcement Learning

The robot brain organizes a vocabulary of keywords that describe the robot's perception of the environment. The results of its experiences are processed by a model that finds cause and effect relationships between executed actions and changes in the environment. This allows the robot to learn from the consequences of its actions in the real world. More specific, the robot starts with a training procedure. During training, the

robot takes pictures with an RGB camera. The raw images will be used by the histogram of oriented gradients (HoG) method to extract salient edges in major directions. Each picture will be divided into several rectangular cells. Each cell's gradient picture will be calculated and the histogram of the major oriented gradients will be calculated. Therefore, each picture will consist of several histograms which will be used later to approximately localize the robot. Next, a clustering algorithm such as K-means, or neural gas, will be used to cluster pictures that are similar to each other. Then during navigation, a goal picture is selected, using reinforcement learning the best set of actions will be selected to take the robot to its goal. However, there is uncertainty in the system. Therefore, each action can bring the robot to several states. After each action is done, the new picture will be tested against the clustered pictured and the new state will be selected based on the clustering results. The new decision will be made by the reinforcement learning algorithm. After obtaining the optimized action sequences for each behaviour, the internal model can be updated based on the outcome of the behaviour. Finally, to test whether the world model of the robot is correct, a set of navigation benchmarks will be designed.

1.2. Thesis Goals and Contribution

The objective of this research is to implement a navigation system that can automatically gather topological information about the environment, process the data, and navigate using reinforcement learning methods to a goal location. The research questions that we aim to answer are:

1. Can we develop a navigation system based on topological information extracted by histograms of oriented gradients?
2. Can we develop this navigation system without user interference in any of the phases?
3. Can we develop a continuous learning method that automatically adapts to changes in the environment?

1.3. Thesis Structure

The thesis structure can be seen in Figure 1. In chapter 2, we discuss the literature study that we have done on reinforcement learning (RL) methods. Dynamic Programming, model-free RL, model-based RL, and partially observable Markov decision processes are the main sections of this chapter. We continue the thesis by presenting the state of the art image processing methods in chapter 03. We start the chapter by introducing histogram equalization, noise reduction and image smoothing methods. Next, we discuss our edge detection method and extraction of histogram of oriented

gradients. Chapter 3 is concluded by a presentation of clustering methods used in this research. Chapter 4 mainly is about our robotic software and hardware framework. We also discuss the approach we used for implementation of the behaviors required to complete the navigation task. In chapter 5 we discuss the results we got from the experiments. Finally, in chapter 6, we conclude the thesis by summarizing the results and suggesting improvements for future work.

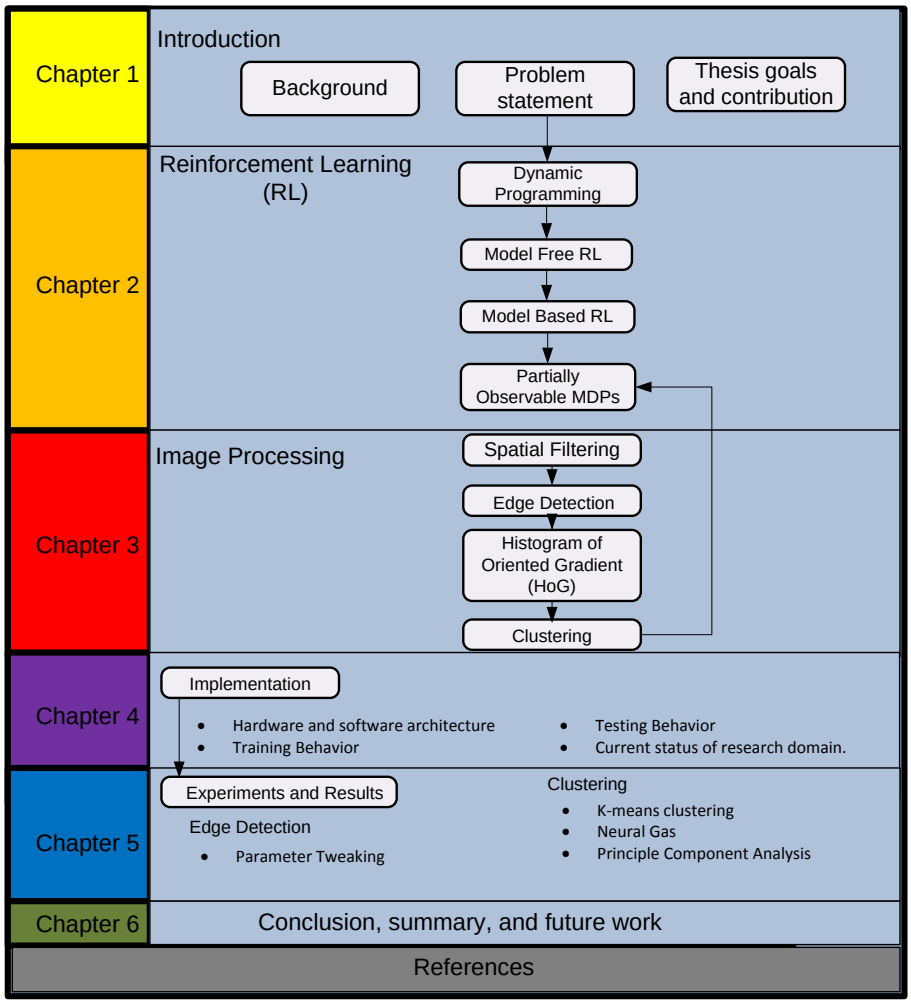

Figure 1 Thesis structure

Chapter 2

2. Reinforcement Learning

The human navigation system is very complex. From the moment that an infant starts crawling, a combination of sensory data is fed to the brain, an action is generated by the brain, and the child will receive feedback. Most of the time, perhaps, the child just randomly moves around to explore the environment. Other times the child moves toward a certain goal, like a toy, his/her[1] parents etc. Over time he learns the characteristics of the environment and can easily navigate through the environment. When he grows up, this task is much faster and he immediately remembers visual scenes and connects them together in order to correctly navigate to the destination. This complex navigation not only uses a visual memory, but also semantics, understanding of physical laws, and common sense (Maguire, Burgess, & O'Keefe, 1999) (Smith & Cheeseman, 1986). Therefore, implementing a similar approach for robots is challenging. Having a robot with pressure sensors everywhere, like our skin, ability to learn, and a complex brain is almost impossible. Therefore, we decided to imitate the human navigation using only the part which is about visual memory. We humans usually memorize the important part of the scene, special patterns, textures, objects, edges etc. and then connect these scenes together and will make a visual route to the goal. During

[1] From now on, to avoid repetition of his/her, by using his or her we mean both male and female subjects.

this process, a rough visual map is also built which helps us understand the environment.

The best method to imitate this learning behavior in humans is reinforcement learning. In reinforcement learning, the learner is a decision-making agent that takes actions in an environment and receives reward (or penalty) for its actions in trying to solve a problem. After a set of trial-and error runs, it should learn the best policy, which generates the sequence of actions that maximizes the total reward.

2.1. Dynamic Programming

Dynamic programming (DP) is a very powerful algorithmic paradigm in which a problem is solved by identifying a collection of sub problems and tackling them one by one. It starts by solving the smallest problems, next, it uses the answers to small problems to help figure out larger ones, until the whole problem is solved. The method can be applied both in discrete time and continuous time settings. The value of dynamic programming is that it is a "practical" (i.e. constructive) method for finding solutions to extremely complicated problems. However, continuous time problems involve technical difficulties. If a continuous time problem does not admit a closed-form solution, the most commonly used numerical approach is to solve an approximate discrete time version of the problem. Since under very general conditions one can find a sequence of discrete time DP problems whose solutions converge to the continuous time solution, the time interval between successive decisions tends to zero (Kushner, 1990). Dynamic programming can also be used to compute optimal policies for Markov decision processes. Three well known methods are used to compute the policy and value function, namely, Policy iteration, Value iteration, and linear programming. Policy iteration evaluates a policy by computing the value of each state by solving a set of linear equations. After that, the policy is changed so the actions with highest Q-values are chosen. In value iteration, for all the states, all the actions are evaluated, and actions with the highest Q-values will assign the value of each state. This procedure is continued until the values stop changing. Linear programming maximizes the value function subject to a set of constraints. We will show the policy and value iteration algorithms, but will first discuss the Markov Decision Process framework.

2.1.1. Markov Decision Processes

A Markov decision process (MDP) is a controllable dynamic system whose state transitions depend on the previous state and the action selected by a policy. The policy

is based on a reward function that assigns a scalar reward to each state-action pair. The objective is to find a policy that maps states to actions in a way that maximizes the expected long-term cumulative reward, given an arbitrary initial state.

A Markov decision process consists of:

- A discrete time counter $t = 0, 1, 2, 3, \dots$
- A finite set of states $S = \{S_1, S_2, S_3, \dots, S_N\}$. A state at time t is denoted as s_t
- A finite set of actions $A = \{A_1, A_2, A_3, \dots, A_M\}$.
- A transition probability function T. We use $T(s, a, s') = P(s_{t+1} = s' \,|s_t = s, a_t = a)$ to define the transition probability to the next state s_{t+1} given s_t and a_t.
- A reward function R that assigns a scalar number to a state/action pair $R(s, a) \in \mathbb{R}$. We assume that the reward function is deterministic.
- A discount factor $\gamma \in [0,1]$ is used to discount rewards received later.

2.1.2. Policy Iteration

Policy iteration calculates an optimal policy and always terminates in finite time (Littman, 1996). This is because we have a limited number of actions and states; therefore the maximum number of policies is $|A|^{|S|}$. Policy iteration makes an update at each iteration of the algorithm. The algorithm is divided in two parts: policy evaluation and policy improvement. The algorithm starts with an arbitrary policy and value function. The symbol π is the policy and $\pi(s)$ is the action selected by the policy in state s. The policy is evaluated by iterating through all the states and solving the following set of linear equations:

$$V_\pi(s) = R\big(s, \pi(s)\big) + \gamma \sum_{s' \epsilon\, S} T(s, \pi(s), s') V_\pi(s')$$

The value of the policy in each state is equal to the reward received by the action done using the policy plus the transition probabilities to the next states multiplied by the discounted value of the policy in the next states. After evaluation, a policy improvement step is done. The new policy in each state will be the action which had the highest value in the respective state.

$$\pi(s) = argmax_a \left(R(s, a) + \gamma \sum_{s' \epsilon\, S} T(s, a, s')\, V_\pi(s') \right)$$

The policy evaluation and improvement steps should be repeated for a specific number of times until the policy is not changed anymore. The algorithm stops with the optimal value function V^* and the optimal policy π^*.

The complexity of the algorithm is only for the evaluation part, a simple comparison is done for the improvement step. Each iteration of this algorithm takes $O(|A||S|^2 + |S|^3)$ time that is more than that of value iteration, but policy iteration needs fewer iterations than value iteration.

2.1.3. Value Iteration

The value iteration algorithm in contrast to policy iteration, does not fully evaluate a policy before the update steps. The method starts with an arbitrary policy and value function. For each state, all the Q-values of the possible actions are calculated,

$$Q(s,a) = R(s,a) + \gamma \sum_{s' \epsilon S} T(s', a, s\,) V(s')$$

Then the new value function will be calculated by,

$$V(s) = max_a\big(Q(s,a)\big)$$

This is continued until V(s) converges. We say that the values converged if the maximum value difference between two iterations is less than a certain threshold, δ

$$max_{s \in S} \left| V^{(l+1)}(s) - V^{(l)}(s) \right| < \delta$$

where "*l*" is the iteration counter. Because we only care for the actions with maximum value, it is possible that the policy converges before the values converge to their optimal values. The complexity of the method is, $O(|S|^2|A|)$, for each iteration. However, there is often a small number $K < |S|$ of next possible states, so complexity decreases to $O(K|S||A|)$.

Value iteration repeatedly performs a one-step look ahead, and this is the big difference between value iteration and policy iteration. In contrast to policy iteration, however, value iteration is not guaranteed to find the optimal policy in a finite number of iterations (Littman, 1996).

2.2. Model-Free Reinforcement Learning

Reinforcement learning can be counted as an automatic learning method. There exists an environment which requires to be explored, and knowledge is gained by the outcomes of the agent's actions (Sutton & Barto, 1998). In reinforcement learning problems, the agent receives input data from the environment. Based on this data, the agent selects an action and receives an internal reward based on the quality of the actions. The goal of the agent is to select the actions in each state which lead to the largest future cumulative rewards which are discounted by a certain factor. In order to solve this problem, different action sequences are executed and the system learns how much long term reward the agent receives on average by selecting a particular action in a particular state. These estimated values are stored in a Q-function which is used by the policy of the reinforcement learning method to select an action. There are two types of reinforcement learning, direct or model-free and indirect or model-based. In model-free reinforcement learning, exploring the unknown environment and learning to choose the correct action sequence is done simultaneously. On the other hand, in model-based RL, first an estimation of the surrounding environment, world, is required and then a dynamic programming approach is used to compute the Q-function. We will first describe the most important RL methods: Temporal difference learning (Sutton, 1988) and Q-learning (Watkins, 1989).

2.2.1. Temporal Difference Learning

As described before, a model is defined by the reward received and probability distributions of the next state and the respective actions. When these are known, we can use dynamic programming to find the optimal policy. However, we rarely have an a-priori model with perfect knowledge of the surrounding environment. Therefore, exploration of the environment is necessary. Consequently, in the case of navigation, significant changes to the environment such as full redecoration will not happen. However, changing a location of a single chair or table is allowed. As we will see shortly, when we explore and get to see the value of the next state and reward, the reinforcement learning algorithm uses this information to update the value of the current state. These algorithms are called temporal difference algorithms, because they take into account the difference between the current estimate of the value of a state (or a state-action pair) and the discounted value of the next state and the reward received.

2.2.2. Q-Learning

One of the simplest reinforcement learning algorithms is Q-learning. (Watkins, 1989)

Initialize all $Q(s, a)$ arbitrarily
For all episodes
  Initialize s
  Repeat
    Choose a using policy derived from Q, e.g., ϵ-greedy
    Take action a, observe r and s'
    Update $Q(s, a)$:
      $$Q(s, a) = Q(s, a) + \eta\left(r + \gamma max_{a'} Q(s', a') - Q(s, a)\right)$$
    $$s = s'$$
  Until s is terminal state

Figure 2 Q-learning, which is an off-policy temporal difference algorithm

(Watkins & Dayan, 1992). In Q-learning, the agent learns the optimal policy by repeatedly executing the actions with the highest estimated future reward intake, or performing an explorative action. An example explorative policy to choose actions is an ϵ-greedy method in which with a fixed probability p the action with highest Q-value is selected and a random action is selected otherwise. The algorithm is shown in Figure 2. The reward r is the value given for action a taken in state s. The step size η defines the learning rate. At each time step the algorithm uses one step look ahead to update the currently selected state/action pair. Q-learning updates all the state/action pairs in the solution path a single time, spreading the final goal reward one step back in the chain. For this reason, it takes a long time till the Q-value changes drop and the system reaches a stable state. Although slow, it is proved that Q-learning will converge to the optimal policy if all the state/action pairs are traversed infinitely often while using an annealing scheme for the learning rate (Watkins & Dayan, 1992). This method is called off-policy because the value of the best next action is used without using the policy that can choose an explorative action.

2.3. Model-based Reinforcement Learning

It is possible to learn a model of the environment by experience. Combining models with reinforcement learning has a wide range of possible advantages. If the agent learns a model and then computes the respective Q-functions, then the learning speed can be significantly improved. Models help improving the exploration behavior. If an agent, in our case a robot navigating in a room, uses a model, it can simulate possible scenarios resulting from a specific action. For example, the robot can plan how to roughly reach the kitchen before executing the movement.

In this section we describe how models can be learned by monitoring the agent in the environment and how they can be used to compute a policy.

2.3.1. Extracting a Model

Given a set of experiences, we have to make a model and compute the parameters for it. The Maximum Likelihood function is a proper method to find which model and parameters reproduce the experimental data best. The likelihood function gives the probability $P(\theta; \rho|\mathrm{X})$ in which θ is the model, ρ the parameters of the model, and X the experimental data. Following the Bayes' rule we have:

$$P(\theta; \rho|\mathrm{X}) = \frac{P(\mathrm{X}|\theta; \rho)\ P(\rho|\theta)P(\theta)}{P(\mathrm{X})}$$

$P(\mathrm{X})$ acts as a normalizing constant and shows the probability of generating the data. Assuming the model is correct, we can understand how good the guessed parameters are. In our problem, we do not know which model is correct. One way of extracting the necessary parameters from a set of experiences is to count the frequency of the occurrence of experimental data, which are quadruples of the form (s_t, a_t, r_t, s_{t+1}) received during exploration of the environment. For this, the agent uses the variables below:

$C_{ij}^a =$ Number of transitions from state i to state j after executing action a

$C_i^a =$ Number of times the agent has executed action a in state i

$R_i^a =$ Sum of the rewards received by the agent by executing action a in state i

The maximum likelihood model (MLM) contains maximum likelihood estimates which maximize the likelihood function. We use matrices to store transition probabilities, and rewards. The estimation of these matrices is done by computing the average probabilities over possible transitions and the average reward.

$$\hat{T}(i, a, j) \leftarrow \frac{C_{ij}^a}{C_i^a}$$

$$\hat{R}(i, a) \leftarrow \frac{R_i^a}{C_i^a}$$

In order to reduce the time, we let the robot to randomly move around, or we manually drive it to experience different states. After sufficient information is gathered, the

system will traverse through all the stored data and updates the respective matrices. If observations are without noise, which is almost impossible in our case, we have a deterministic reward and transition function and the estimated reward for a particular transition from state s, by action a, to state $s`$ is known and fixed after a single experience. However, in our case, to estimate the transition probabilities we need to have multiple examples of the transition in the experimental data, since there are multiple results because of the different stochastic outcomes of each state/action pair. Otherwise, the decisions made later will be based on insufficient data and this can lead to reduced performance or failure.

Bias. Since the extracted information is directly sampled from the underlying probability distribution and we use the maximum likelihood model with statistical transition probabilities and reward matrices, the estimator is unbiased.

Variance. The variance of transition probabilities $\hat{T}(s,a,s')$ after n occurrences of the state/action pair (s,a,s') is:

$$Var\left(\hat{T}(s,a,s')|\mathrm{n}\right) = \sum_{k=0}^{n}\left(\frac{k}{n} - T(s,a,s')\right)^2 \binom{n}{k}\left(1-T(s,a,s')\right)^{n-k}\left(T(s,a,s')\right)^k = \frac{T(s,a,s')\left(1-T(s,a,s')\right)}{n}$$

As can be seen, the variance goes to 0 as the number of experiences of each specific state/action pair goes to infinity. However, for usual problems, there is no need to accurately extract the probabilities by running a lot of experiments. It is possible to use the policy and exploration to focus on some parts of the state space. Since the policy is derived from the model directly; we need to learn from a large number of new experiences in order to avoid performance reduction because of the variance. Therefore, model-based learning is in fact a stochastic approximation algorithm.

Initialize $\boldsymbol{V(i)}$ to arbitrary values
Repeat
  For all $\boldsymbol{i} \in \boldsymbol{S}$
    For all $\boldsymbol{a} \in \mathbf{A}$
      $\boldsymbol{Q(i,a) \leftarrow \hat{R}(i,a) + \gamma \sum_{j \in S} \hat{T}(i,a,j)\, V(j)}$
    $\boldsymbol{V(i) \leftarrow max_a Q(i,a)}$
Until $\boldsymbol{V(.)}$ converges

Figure 3 Value iteration based on a Model

2.3.2. Value Iteration based on a Model

The value iteration method requires an expected reward and transition probabilities per state/action pair. Therefore, it is intrinsically based on a model itself. From the experiments we deduce the transition probability matrix $\hat{T}(s, a, s')$. The expected reward for each state/action pair is initially zero. Only the actions that connect a state to the final goal state will have a reward larger than 0. After a certain number of iterations, the values of each state will be stabilized. The algorithm seen in Figure 3 is based on the value iteration algorithm described in section 2.1.3.

2.3.3. Prioritized Sweeping

In the value iteration model-based approach we use the probabilistic graph to propagate state-value updates to other state-values. However, since the state space is fairly large in the case of navigation, the convergence of values may take a lot of time and slows down the learning process. When there are high probability transitions to distant states, a small change in their values will cause a chain of changes in other states. This change destabilizes the whole system and a lot of iterations will be required for convergence. Therefore, in order to efficiently distribute the state-values, some management of update steps should be performed so that only the most useful updates are propagated through all the states.

Prioritized sweeping was found by (Moore & Atkeson, 1993) which is an efficient management method that decides which updates have to be performed. This method uses a heuristic estimate of the size of the Q-values' update and assigns priorities for state updates based on that. The algorithm stores a backtracking model, which connects states to previous state/action pairs. After a number of state value updates, the predecessors of the state are inserted in a priority queue. Then the Q-values of the states with the highest priority in the priority queue are updated. For the experiments, we will use a priority queue for which an insert/delete/update operation takes $\mathrm{O}(\log n)$ with n the number of states in the priority queue.

Moore and Atkeson's prioritized sweeping uses a set of predecessor lists, $Preds(j)$, which contains all predecessor state/action pairs (j, a) of a state i. The priority of state i is stored in another list called $\rho(i)$. When the value of state i is updated, the transition

Promote the most recent state to the top of the priority queue
While $\boldsymbol{n < U_{max}}$ **AND the priority queue is not empty**
  Remove the top state $\boldsymbol{i}$ **from the priority queue**
    For all $\boldsymbol{a} \in \mathbf{A}$
      $\boldsymbol{Q(i,a) \leftarrow \sum_{j \in S} \widehat{T}(i,a,j)\left(\widehat{R}(i,a) + \gamma V(j)\right)}$
    $\boldsymbol{V'(i) \leftarrow max_a Q(i,a)}$
    $\boldsymbol{D \leftarrow |V(i) - V'(i)|}$
    $\boldsymbol{\rho(i) \leftarrow 0}$
    $\boldsymbol{V(i) \leftarrow V'(i)}$
    For all $\boldsymbol{(j,a) \in preds(i)}$
      $\boldsymbol{P \leftarrow \widehat{T}(i,a,j)D}$
      If $\boldsymbol{P > \rho(j)}$
        $\boldsymbol{\rho(j) \leftarrow P}$
        If $\boldsymbol{P > \lambda}$
          Promote $\boldsymbol{j}$ **to new priority** $\boldsymbol{\rho(j)}$
    $\boldsymbol{n \leftarrow n + 1}$

Figure 4 Moore and Atkeson's prioritized sweeping algorithm

from (j, a) to i contributes to the update of $Q(j, a)$. The priority of a predecessor state j is the maximum value of these kinds of contributions. The algorithm can be seen in Figure 4.

The parameter U_{max} denotes the maximal number of updates which is allowed to be performed per update sweep to keep the speed high. The parameter λ controls the update accuracy. On each loop, the current state/action pair will be put on the top of the queue, and then it will remove the top state from the queue and update its Q-value. Next, we store the amount of update in a temporary value D and assign zero to the priority of the current state. Finally, we traverse all the predecessors j of the state i, and if the transition probability of that state/action pair to the current state multiplied by D is bigger than the priority of state j and λ threshold, then we assign it as the new priority of that state and promote it in the priority list (Wiering, 1999).

2.4. Partially Observable States

In certain applications, such as navigation, the agent does not know the state exactly, but it has access to information via sensors. The observation helps the agent to estimate the state. In this thesis, the example is navigation in an unknown environment. The robot has a RGB camera. The image processing part of the software calculates important edge information, and feeds it to the agent. This information does not tell the robot its exact state, but gives some indication as to its likely state. Using the information about the edges in different parts of the image, the robot may only know that it is located somewhere in the living room near the door. The setting is like a Markov decision process, except that after taking a specific action a_t, for example moving forward for one meter, the new state S_{t+1} is not known because of the robot movement and perception uncertainties. For example, it is possible that a robot sees an obstacle and moves to a different direction, or because of the robot's imperfect odometry, it does not move exactly one meter. However, we have an observation O_{t+1} which is a stochastic function of s_t and a_t. This is a partially observable Markov decision process or POMDP. If, $O_t = s_t$ for all t, then the POMDP is reduced to an MDP. From the observation, we could deduce the real state (or rather a probability distribution for the states) and then take actions based on this. If the agent believes that it is in state S_1 with probability 0.4 and in state S_2 with probability 0.6, then the value of any action is 0.4 times the value of the action in S_1 plus 0.6 times the value of the action in S_2. One difference between POMDPs and MDPs is that the Markov property does not hold for the observations in a POMDP, which means the next state observation does not only depend on the current action and observation. When there is limited observation, or the observations are faulty because of the information received, two states may appear equal but are actually different from each other. If these two states require different actions, this can lead to a loss of performance, as measured by the cumulative reward. Therefore, it is essential that the agent has a failure recovery in case of such situations. The agent should somehow keep track of the past trajectory and compress it into a current unique state estimate. The past observations can also be taken into account by taking a part of the past using a window of observations as input to the policy or using a recurrent neural network to maintain the state without forgetting past observations. In this thesis we take into account the history of observations. The agent may also take an action to gather information and reduce uncertainty, for example, the robot can go to a search mode and moves randomly until it sees a familiar scene or landmark, or stop to ask for directions. The agent chooses between actions based on the amount of information they provide, the amount of reward they produce, and how they change the state of the environment.

One formal method to approach POMDPs is that the agent keeps an internal belief state b_t that is the guess of the agent about its current state based on the information received via sensors. The agent has a state estimator that updates the belief state b_{t+1} based on the last action a_t, current observation O_{t+1}, and its previous belief state b_t. There is a policy π that generates the next action a_{t+1} based on this belief state, in contrast to the real state in a completely observable environment. The belief state is a probability distribution over states of the environment given the initial belief state (before we did any actions) and the past observation-action history of the agent (without leaving out any information that could improve agent's performance) and the selected action. This approach relies on a model of the environment after which POMDP solutions can be used. Estimating such a model can be done with hidden Markov models, but these do not scale up well and need a lot of training examples. Therefore, we propose using past observations in a history window to disambiguate the current observation when necessary.

Chapter 3

3. Image Processing and Clustering

One of the most essential parts of robotics is vision and image processing. The same applies to humans, we are unable to easily follow our daily activities without our eyes and vision system. Most of the activities either require direct vision data for processing or vision data for feedback. Grabbing objects, navigation and path planning, any kind of recognition requires visual information. It is possible to survive without vision system, as shown by visually impaired people, but it reduces the ability of the person significantly, and there is no good replacement of such system in robotics. In chapter 2, we presented our method to solve a partially observable Markov decision process. In this chapter, we start by describing image processing preliminaries and continue by presenting our novel method to distinguish states from one another by using a set of image processing methods. Since our model-based reinforcement learning method requires a set of discrete states, we will end with clustering method to discretize the perceptual space.

3.1. Histogram Equalization

Histograms can be used for numerous spatial domain processing techniques. However, the histogram of one specific image can change, if we change the contrast of the image. For instance, the components of the histogram of a particular dark image are concentrated on the low side of the intensity scale, and if we lighten the same image, the components of the histogram will be biased toward the light side of the scale. In the

case of navigation, it is possible that data gathering is done in different times of the day. This means that the contrast of each image can be affected by the position of sun, or shades made by different lamps and objects. Therefore, before we use an image for our computational purposes, we need to use a method to lessen the sensitivity of histograms to changes of image contrast. This can be achieved by histogram equalization.

The histogram of a digital image with intensity levels of range $[0, L-1]$ is defined as a discrete function $h(r_k) = n_k$, where r_k is the kth intensity value, and n_k is the number of pixels having intensity value r_k. Using the following formula we calculate the new intensity values for the histogram equalized image.

$$s_k = \frac{(L-1)}{MN} \sum_{j=0}^{k} n_j$$

$$\text{for } k = 0, 1, 2, \dots, L-1$$

where MN is the total number of pixels, n_j is the number of pixels with intensity value r_j , and L is the total number of possible intensity levels in the image. At this point, s_k may contain fractions because they were generated by summing probability values. Therefore, we round s_k to the nearest integer. Finally, the intensity value of pixels of which their original intensity level is not included in s_k anymore, will be changed to the closest higher intensity value available in s_k.

Figure 5 shows the histograms of one image before and after equalization. The original image mostly shows low intensity values, but the equalized image includes a larger contrast range[2].

3.2. Spatial Filtering

After histogram equalization, a set of operations is required to be done on the image, such as smoothing, edge detection, etc. These operations require certain filters to be applied on the image using spatial filtering methods.

Two important concepts in linear spatial filtering are correlation and convolution. Simply, correlation is the process of moving a desired filter mask over an image and computing the sum of the products at each location. The mechanics of convolution are similar to those of correlation, except for the fact that in convolution the mask should

[2] picture source: http://www.cs.utah.edu/~jfishbau/improc/project2/

be rotated by 180 degrees in the beginning. In the following sections we explain two dimensional correlation and convolution, as we used in our work.

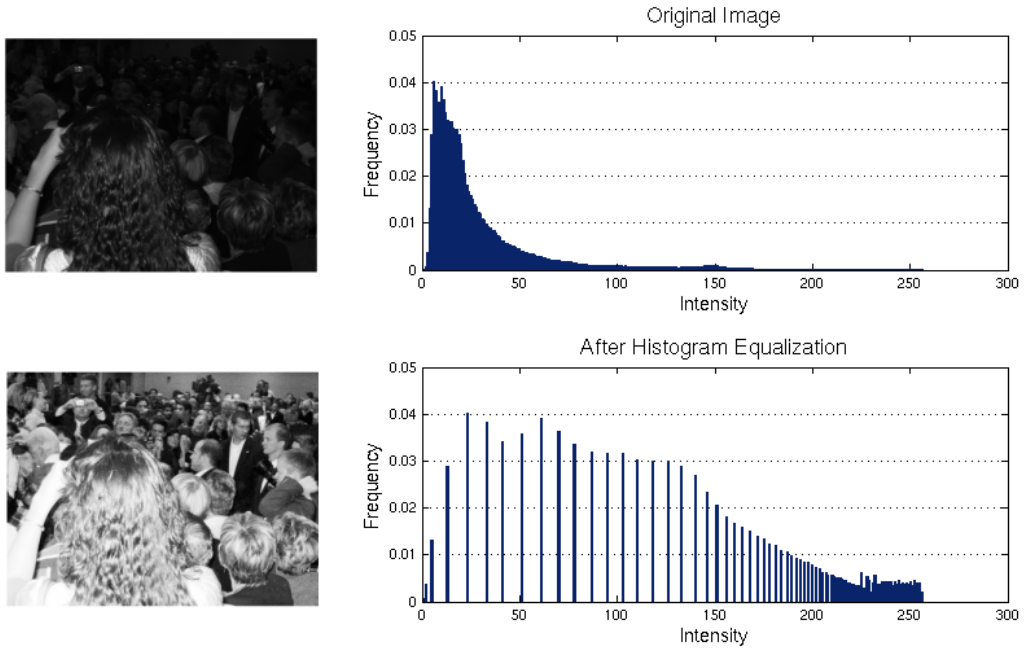

Figure 5 Histograms of a crowd, before and after equalization.

3.2.1. Correlation and Convolution

Having an image, and a filter of size $m \times n$, the first thing we need to do is to pad the image with a minimum of $m-1$ rows of zeros at the top and bottom, and $n-1$ columns of zeros on the left and right. The reason for this is that the center of the mask should traverse all of the picture pixels. When the center of the mask is on the border, some part of the mask will be outside of the image; therefore we need padding to avoid ambiguities. Then, we begin to slide the mask over the image to calculate either correlation or convolution by computing the sum of the products of filter weights and pixel values at each pixel of the image.

To compute the correlation of image $f(x,y)$ with filter $w(x,y)$ of size $m \times n$, which is denoted by $w(x,y) * f(x,y)$, we use the following equation:

$$w(x,y) * f(x,y) = \sum_{s=-a}^{a} \sum_{t=-b}^{b} w(s,t) f(x+s, y+t)$$

where

$$a = \frac{(m-1)}{2} \text{ and } b = \frac{(n-1)}{2}$$

If f has been padded appropriately, then we can apply this formula on all the pixels of f.

In a similar manner, to compute the convolution of image $f(x,y)$ with filter $w(x,y)$ of size $m \times n$, which is denoted by $w(x,y) * f(x,y)$, we use the following equation:

$$w(x,y) * f(x,y) = \sum_{s=-a}^{a} \sum_{t=-b}^{b} w(s,t) f(x-s, y-t)$$

As we already mentioned, we need to rotate the filter by 180°, before we start to slide it over the image. In convolution expression, this is applied by inserting minus signs on the f. Shifting f instead of w is done for notational simplicity, and the result is the same as if we have rotated the filter.

Based on the fact that using correlation or convolution to perform spatial filtering is a matter of preference, and each of them could be used to perform the intended operation, we have decided to use convolution in our work.

3.3. Noise Reduction

Image noise is a random (not present in the real object imaged) fluctuation of illumination or color information in images, and is usually an aspect of electronic noise. Noise in our case is usually produced by the sensor and circuitry of the digital camera. The digital camera noises can be divided as follows:

- **Amplifier Noise**: In colour cameras, more amplification is used in the blue colour channel than in the green or red channel. Therefore the blue channel data can be noisier than the other ones.
- **Shot Noise**: The dominant noise in the lighter parts of an image from an image sensor is typically caused by statistical quantum fluctuations. This noise is identified as photon shot noise. Shot noise has a root-mean-square value related to the square root of the image intensity, and the noises at different pixels are independent of one another. Shot noise follows a Poisson distribution, which is usually not very different from Gaussian.
- **Moving Noise**: This noise is caused when the speed of sensing the image is less than the speed of the camera. This happens when the picture is taken during camera movement. This can be counted as an external distortion more than camera noise.

These types of noises will reduce image processing performance significantly. We provide an example (Gonzalez & Woods, 2008) to see how these types of noise can be destructive in edge detection.

Figure 6 shows a close-up of four different ramp edges transiting from a black region to white region. The first image segment, located at the top of the figure, is free of noise. But the rest of the ramp edges are corrupted with additive Gaussian noise with zero mean and standard deviations of 0.1, 1.0, and 10.0 intensity levels. The graph below each image is a horizontal intensity profile passing through the center of the image, and the second and third columns indicate first and second-derivatives, respectively. As we go from the top to the bottom in the first column of Figure 6 the standard deviation is increased, and therefore, the Gaussian noise is increased. It is clear that, when the Gaussian noise is increased the first-derivatives become increasingly different from the noise free case. The second-derivatives are even more sensitive to the noise, and as the noise increases it gets more difficult to associate the second-derivatives to their ramp edges.

This example is a good illustration of sensitivity of derivatives to noise. Therefore, we need to use a method to first smooth the image and reduce noise, and then perform edge detection. Since, most of the images are affected by shot noise, we use a Gaussian smoothing filter to decrease the effect of the noise.

3.3.1. Gaussian Smoothing Filter

We use a Gaussian smoothing (also known as Gaussian blur) filter for blurring images and reducing noise and details. Mathematically, applying a Gaussian smoothing filter on an image is the same as convolving the image with a Gaussian function. The equation of a Gaussian function in one dimension is:

$$G(x) = \frac{1}{\sqrt{2\pi\sigma^2}} e^{-\frac{x^2}{2\sigma^2}}$$

In our work we use this filter in two dimensions, and it is the product of two Gaussians, one in each dimension:

$$G(x, y) = \frac{1}{2\pi\sigma^2} e^{-\frac{x^2+y^2}{2\sigma^2}}$$

where x is the distance from the origin in the horizontal axis, y is the distance from the origin in the vertical axis, and σ is the standard deviation of the Gaussian distribution.

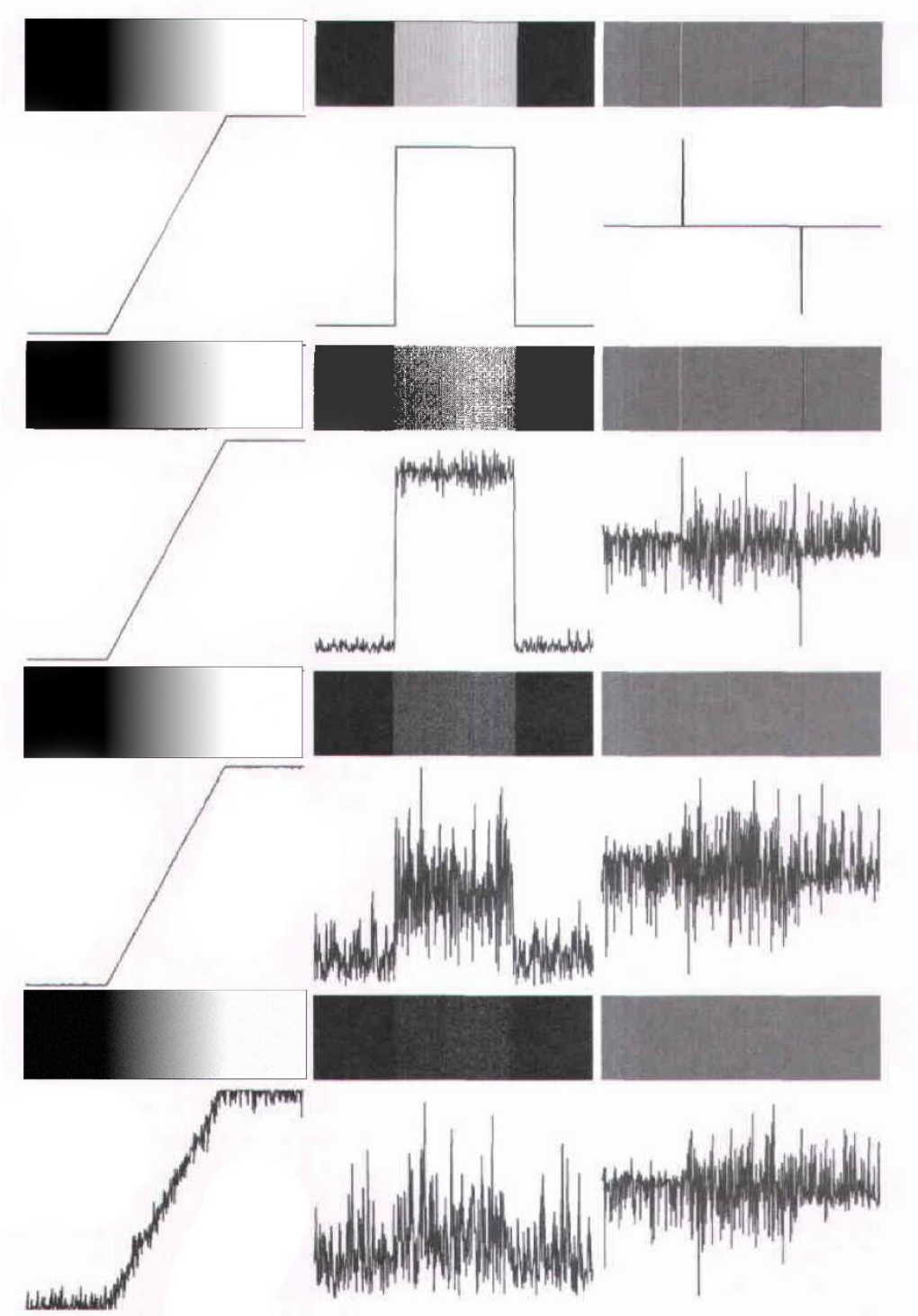

Figure 6 Four different ramp edges transiting from a black region to white region. The 2nd to 4th ramp edges are corrupted with additive Gaussian noise with zero mean and standard deviations of 0.1, 1.0, and 10.0 intensity levels. The second column is the first derivative. The third column is the second derivative. From the Image courtesy of Rafael C. Gonzalez.

A Gaussian smoothing filter is a low-pass filter, which attenuates high frequency signals. We use $G(x, y)$ to compute a 3×3 filter, $Z(x, y)$, and in future computations we will use this filter to speed up the computations.

3.4. Edge Detection

Now that the image is smoothed and its histogram is equalized, we can apply the main image processing methods. As mentioned in chapter 2, our goal is to implement a navigation system that is close in spirit to the human navigation method. It is found that humans mostly use topological information for their navigation with addition of semantics and texture detection (Maguire, Burgess, & O'Keefe, 1999). Our system, however, will only use topological information. To achieve this goal, we plan to extract topological information by extracting edge intensities and orientations. The idea is to split the image in several sub images, and find the salient edges and their orientation. One of the most famous edge detectors is the Canny edge detector which we will describe in the following subsection.

3.4.1. The Canny Edge Detector

Although the Canny edge detector (Canny, 1986) is one of the most complex methods of edge detection, it is a very robust approach and its performance is superior compared to other edge detector methods (e.g., the Marr-Hildreth edge detector). This approach is based on three main objectives:

- **Low error rate.** This means that all the edges of an object should be found, and the detected edges should be as close as possible to the real edges of the object.
- **Good localization of edge points.** The located edges should be as close as possible to the real edges of the object. This means that the distance between a point specified as an edge and the center of the real edge should be minimum.
- **Single edge point response.** Only one point should be returned by the detector for each real edge point. This means that the number of local maxima around the real edge should be minimum.

The Canny edge detector is a multi-step detection procedure. The steps are as follows:

1. Smoothing the input image by using a Gaussian filter in order to reduce the noise and undesirable details and textures:

$$f_s(x, y) = Z(x, y) * f(x, y)$$

Where $Z(x, y)$ is a 3×3 filter introduced in section 3.3.1.

2. Compute gradients in both x and y directions using any of the gradient operators (i.e., Roberts, Sobel, Prewitt, etc.) to get the magnitude and angle image. For our work we decided to use the Sobel gradient operator (Gonzalez & Woods, 2008):

$$M(x, y) = \sqrt{g_x^2 + g_y^2}$$

and

$$\alpha(x, y) = \tan^{-1}\left[\frac{g_y}{g_x}\right]$$

Where the Sobel masks for x and y gradients are:

$$Sobel_x = \begin{bmatrix} 1 & 0 & 1 \\ 2 & 0 & 2 \\ 1 & 0 & 1 \end{bmatrix} \quad \text{and} \quad Sobel_y = \begin{bmatrix} 1 & 2 & 1 \\ 0 & 0 & 0 \\ 1 & 2 & 1 \end{bmatrix}$$

The gradient images are calculated by convolving the Sobel masks on $f_s(x, y)$.

$$g_x(x, y) = Sobel_x * f_s(x, y)$$
$$g_y(x, y) = Sobel_y * f_s(x, y)$$

3. Thinning ridges of magnitude image by using non-maxima suppression. We check to see whether each non-zero $M(x, y)$ is greater than its two neighbors along g_x and g_y. If so, keep the magnitude unchanged, otherwise, set it to 0.
4. Finally, $g_N(x, y)$, which is the nonmaxima-suppressed image, should be thresholded. Canny's algorithm uses hysteresis thresholding to avoid including false edges and/or eliminating valid edges while setting the threshold. Hysteresis thresholding is performed by selecting two thresholds: a low threshold, T_L and a high threshold, T_H. Canny suggests in his method (Canny, 1986) that the ratio of the threshold T_H to threshold T_L be two or three to one. The thresholding operation can be visualized by creating two extra images:

$$g_{NH}(x, y) = g_N(x, y) \geq T_H$$

and

$$g_{NL}(x, y) = g_N(x, y) \geq T_L$$

where both $g_{NH}(x,y)$ and $g_{NL}(x,y)$ are set to zero at the beginning. After performing the thresholding operation, $g_{NH}(x,y)$ will have fewer nonzero pixels than $g_{NL}(x,y)$, and since $g_{NL}(x,y)$ is created with a lower threshold, all the nonzero pixels in $g_{NH}(x,y)$ will be included in $g_{NL}(x,y)$. Therefore, we remove all the nonzero pixels of $g_{NH}(x,y)$ from $g_{NL}(x,y)$:

$$g_{NL}(x,y) = g_{NL}(x,y) - g_{NH}(x,y)$$

After we perform thresholding, all the strong pixels (i.e., nonzero pixels in $g_{NH}(x,y)$) in $g_{NH}(x,y)$ will be specified as valid edge pixels and are marked. Based on the value of T_H the edges in $g_{NH}(x,y)$ might have gaps. However, longer edges can be formed by using the following four steps procedure:

1. Identify the next unvisited edge pixel, p, in $g_{NH}(x,y)$.
2. Mark all the weak pixels (i.e., nonzero pixels in $g_{NL}(x,y)$) that are connected to p as valid edge pixels.
3. If all nonzero pixels in $g_{NH}(x,y)$ have been visited go to step 4, otherwise, return to step 1.
4. Set all of the unmarked (as valid edge pixels) pixels in $g_{NL}(x,y)$ to zero.

At last, we can get the output image of the Canny edge detector algorithm by linking all the nonzero pixels from $g_{NL}(x,y)$ to $g_{NH}(x,y)$. In Figure 7 the main processes of obtaining the Canny edge image are demonstrated.

3.5. Histogram of Oriented Gradients

Based on the description in chapter 2, we need a method that can transfer the robot observations, pictures taken, into states. Therefore, pictures that are taken from close geographical locations should be also close to each other in our data space. In order to achieve this, we use the idea of histogram of oriented gradients (Dalal & Triggs, 2005). In this method, we first divide the picture in $m \times n$ rectangular cells. Next, we use histogram equalization on each image. After equalization, Gaussian smoothing is done on each picture to decrease the effect of noise. Next, using the Canny edge detector, we calculate the important edges. Then the magnitude and orientation images are recalculated on the Canny edge detector result. Finally, we make a histogram with eight bins that correspond to eight major directions for each cell. For each pixel in the cells, the orientation will be decided from the filtered orientation image, and the weight of the edge is calculated by normalizing the pixel's edge magnitude from the magnitude image. The final result will be added to the corresponding histogram bin.

Figure 7 (a) The original Image (b) The gradient image in direction y (c) The gradient image in direction x (d) The final Canny edge image

Because we use an edge histogram consisting of 8 edge directions, each sub image will result in eight real numbers. If we divide a picture in 5 by 5 cells, the result will be a vector of length 200. Thus, all the images are transformed to the same number of real values.

3.6. Clustering Methods

Now that we have vectors representing our observations, we can use the Euclidean distance function to find out how close they are together. For our model-based reinforcement learning approach we cluster observations to make them discrete. We are going to present two famous unsupervised clustering methods, K-means clustering, and Neural Gas.

3.6.1. K-Means Clustering

K-means (MacQueen, 1967) is one of the simplest unsupervised learning methods that solves the well-known vector quantization problem. The main idea is to define *k* centroids, one for each cluster. Usually, a good practice is to initially select the centroids as random members of the dataset. Then, we traverse the data set. For each point, the distance to all the centroids is calculated, and the label of the data point will be the label of the closest centroid. The distance measure can be anything, but the famous ones are Euclidean, Manhattan, and Mahalanobis distance. After a complete iteration through the data set, the centroids will be recalculated by averaging all the data points with the same label inside that cluster. The procedure is continued until the changes in the location of centroids are less than a certain threshold.

Initialize $\boldsymbol{m_i, i = 1, \ldots, k}$, **for example, to** $\boldsymbol{k}$ **random** $\boldsymbol{x_t}$

Repeat

For all $\boldsymbol{x_t \in X}$

$$\boldsymbol{b_i^t \leftarrow \begin{cases} 1 \; if \; \|x_t - m_i\| = min_j \|x_t - m_j\| \\ 0 \; Otherwise \end{cases}}$$

For all $\boldsymbol{m_i, i = 1, \ldots, k}$,

$$\boldsymbol{m_i = \sum_t b_i^t x_t / \sum_t b_i^t}$$

Until $\boldsymbol{m_i}$ **converge**

Figure 8 K-Means clustering algorithm

The algorithm which is shown in Figure 8 aims at minimizing an objective function, in this case a squared error function as shown below. The prototypes are $\boldsymbol{m_i}$, and $\boldsymbol{x_t}$ are the data points.

$$E(\{m_i\}_{i=1}^k | X) = \sum_t \sum_t b_i^t \|x_t - m_i\|^2$$

Where

$$b_i^t \leftarrow \begin{cases} 1 \; if \; \|x_t - \; m_i\| = min_j \|x_t - \; m_j\| \\ 0 \; Otherwise \end{cases}$$

K-means, however, has a number of problems which can severely reduce the reliability of its results:

- **Dead Units**
  It is possible that we randomly select an outlier as a centroid in K-means. The result is that the centroid will not be updated since its distance to the rest of the data is extremely high which makes the results biased and unreliable.
- **Multimodal Data**
  If the underlying data represent a multimodal shape, then K-means clustering error increases.
- **Dependance on Initialization**
  The results and reconstruction errors are significantly dependent on the initial locations of cluster centers.
- **Local Minima**
  K-means clustering does not guarantee global minimization. Because of the previous mentioned problems, this clustering method often falls into local minima.

3.6.2. Neural Gas

Neural gas (Martinetz & Schulten, 1991) is an artificial neural network, inspired by Kohonen's self-organizing map (Kohonen & Somervuo, 1998). The neural gas is a simple algorithm for finding optimal data representations based on feature vectors. The same as the k-means clustering method, the cluster centers are initialized to random data members. The method initializes a neighbourhood value β to later use in the update of the prototypes. Next, a random data point will be selected, and all the cluster centers will be ranked based on their distance to the data point. The rank is lower if the cluster centers are closer and vice versa. Therefore each cluster center m_i will have a rank value of μ_i. Finally, each cluster center will be updated using the following formula.

$$m_i = m_i + \eta \cdot e^{-\mu_i/\beta}(x_t - m_i)$$

After each epoch, the neighbourhood value β decreases. The pseudo code of the method can be seen in Figure 9. For our experiment, $\boldsymbol{\eta}$ is equal to one divided by the number of data points. For small values of β, effectively only the winning cluster

Initialize $m_i, i = 1, \ldots, k$, **for example, to** k **random** x_t
Repeat
  For all $x_t \in X$
    For all i**,**
      $D_i = \|x_t - m_i\|$
    Sort(D_i**) in ascending order**
    $\mu_i =$ **rank of cluster** i
  For all $m_i, i = 1, \ldots, k$,
    $m_i = m_i + \eta \cdot e^{-\mu_i/\beta}(x_t - m_i)$
  $\beta = \beta/z$
Until m_i **converge**

Figure 9 Neural Gas clustering algorithm

updates since all other cluster updates will be exponentially lower. In our experiment, the initial neighbourhood value β is selected as the number of clusters divided by two. These values were selected based on experiments and observations on our image datasets. Neural gas solves some of the K-means clustering problems, such as dead units, because of the simultaneous update of all clusters in each epoch.

Chapter 4

4. Implementation

In this chapter, first, we present the hardware characteristics of the robotic system. Next, we continue by describing the framework, programming languages, and architecture of the robot used for the proposed navigation system. Finally, we conclude the chapter by mentioning the libraries and open source programs used to develop our system.

4.1. Robotic Hardware

The robot which is used for all the phases in this thesis can be seen in Figure 10. As can be seen, the robot is consisting of a mobile platform, structural frame, different sensors, and a minimum of two processing units.

4.1.1. Processing Units

One laptop is required as the Brain of the robot. It is possible to add other processing units to the system to increase the computing speed and performance. All of the processing units are connected to a networking switch which enables them to connect to another platform (i.e., the moving platform), and exchange data over the network.

4.1.2. Sensors

The sensors available in this robotic system are as follow:

- RGB Camera (Connected to processing unit)
- Depth Camera (Connected to processing unit)

- Odometer (On Pioneer 2)
- Sonar (On Pioneer 2)
- Stall (On Pioneer 2)

The RGB camera is mounted on the bottom and top of the robot as can be seen in Figure 10. A Microsoft Kinect RGBD camera is used on top of the robot. The moving platform, Pioneer 2, has an odometer, Sonar, and stall sensors for which we only use the Odometer and stall for our navigation purpose.

4.1.3. Pioneer

The base mobile platform used in our system is the Pioneer-2 robot, manufactured by Aria Robotics. This robot has a size of 40x20x15cm and is capable of carrying 23Kg of weight in addition to its own weight without batteries. Pioneer 2 movements are done by two moving wheels connected to two different engines. It receives movement commands from the main processing unit, Brain (Figure 11).

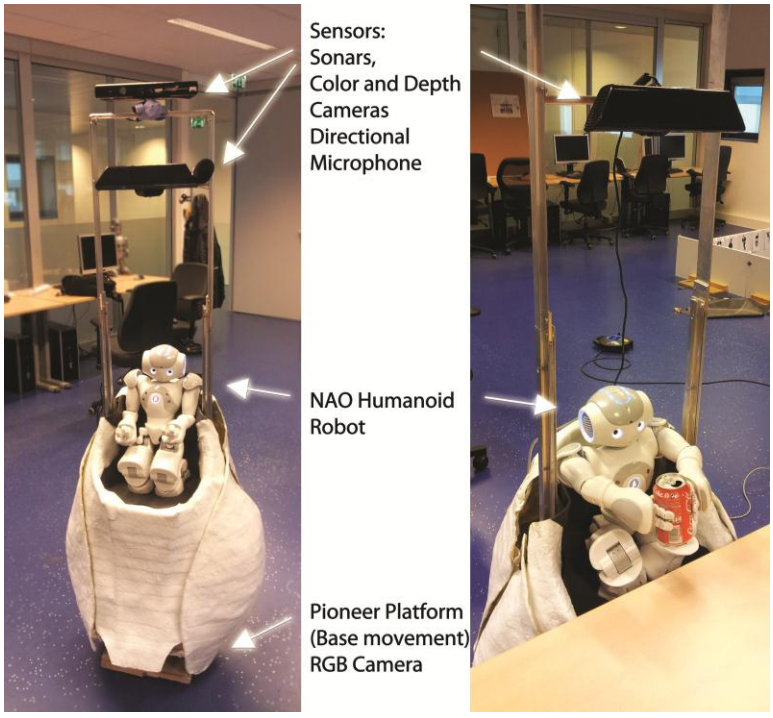

Figure 10 Robotic Platform

Figure 11 (left) Pioneer 2 Platform (right) Pioneer 2 AT Platform

4.2. Robotic Software

The robotic software framework of our robot is based on distributed behavior based robotics. As can be seen in Figure 12, The Brain of the robot is the main control center. It is connected to the behavior controller, sensor integrator, and body controller. The behavior controller is the location where all the high level decisions are made. Several behaviors run in parallel in order to make sure that the robot executes user commands. However, in order to make decisions, each behavior requires high level information input. The sensor integrator is the part that manages all the information received from sensor modules. Several sensory modules can be run on distributed computers. All these sensor modules occasionally can send information updates to the sensor integrator. Then, the sensor integrator will sort and place the data in the robot Memory. Memory is the location where all the high level processed information is stored and is accessible by all the main modules in the Brain. The main task of memory is to store processed sensory data, but it can also be used to store behavior status. The objects in the memory have a name and a timestamp, which means all the data stored during robot operation will be saved. Finally, to execute selected actions from the behaviors module, the commands are given to the body controller which has the authority to give commands to the robotic platform to either move, turn, manipulate or speak.

4.2.1. Programming Language

The main language used for the whole architecture is Python. This language was selected because of its simplicity in implementation and availability of image processing tools for this language. MATLAB was used for the main clustering part of the system because of its easy approach for matrix calculation. The processing time required for these experiments can become very large when constantly recalculating descriptors or

during clustering, especially considering some experiments will have to be repeated for different parameter settings. The required processing time was reduced by making use of threading while extracting features. The program design follows an Object Oriented approach. Methods and Features are described generically and are easily accessible for customization of existing features, as well as, the insertion of new objects. The Object Oriented approach has also aided greatly in creating the vision tool, which was discussed in section 3.

4.2.2. Libraries

A lot of standard libraries were used for development of this project. The most important library was the OpenCV Library. OpenCV was used for image acquisition, noise reduction, filtering, edge detection and histogram equalization. This library was also used to enable the obstacle avoidance method.

4.3. Methodology

The implementation of this project was separated into the following phases:

1. Image and action acquisition
2. Image smoothing
3. Image division
4. Edge detection
5. Histogram of oriented gradients extraction
6. Clustering
7. Goal based reinforcement learning – Value Iteration
8. Action control

Since our robotic framework is behavior based, we separated the task into training and testing navigation behaviors.

4.3.1. Training Behavior

The idea of training is to be able to leave the robot in any environment, and it should safely move around and record observations and actions taken without hitting an obstacle. Next, it should process the entire image database and convert them to real numbers, do clustering on the real numbers and be ready to start the navigation as soon as a goal location is selected.

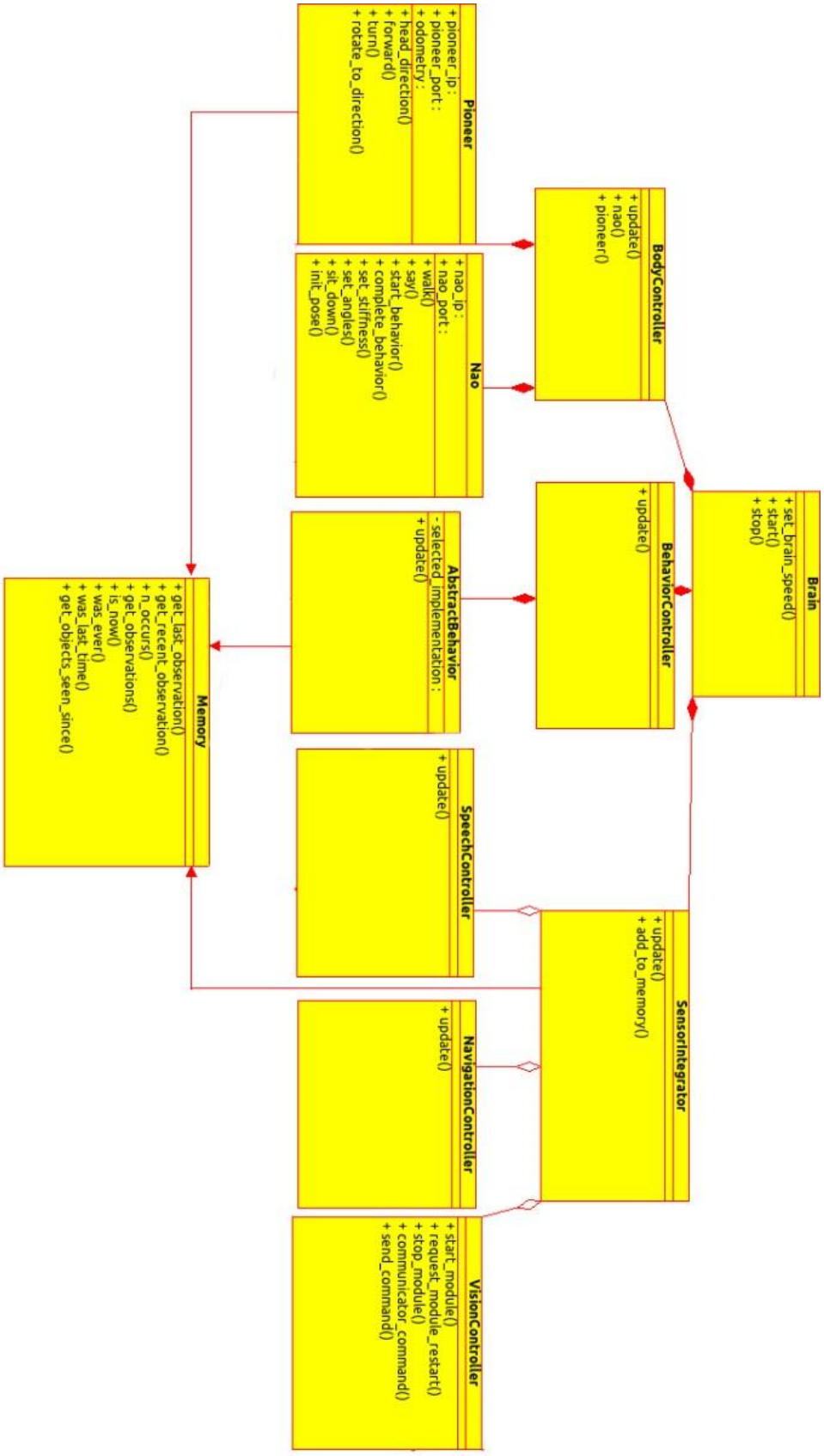

Figure 12 Robot Architecture UML Diagram.

Data Gathering

We implemented two types of training behavior, automatic and manual. In manual training behavior, the user controls movements of the robot by keyboard. He can select three actions: Move, turn left, and turn right. If there is an obstacle in front of the robot, the obstacle avoidance will take over and guides the robot safely to move away from it.

The same approach applies to the automatic method, the difference is that the actions are selected based on a random number generator. In order to increase the performance of data gathering, the robot cannot turn left after having a turn right or vice versa.

The detailed observation method is as follow:

Before taking each action, the robot will take a picture of the environment. An action will be selected and the robot starts to execute that action. The robot has X seconds to complete the action. If it is unable to execute the action, the action will be marked as finished after X seconds are passed, and another action will be selected. This is to overcome the possible hardware and software problems that can rise during the behavior execution. However, if the robot is aware of its internal problem, the behavior stops before the external modules are restarted and working.

When an action is finished, all variables are reset and another observation will be taken. There is one exception in this process. If the robot sees an obstacle in its movement trajectory, and a collision is imminent, the robot will stop the movement command, mark it as finished, and continues a new enforced action called “obstacle”. After an obstacle is recognized, the robot should move away from the obstacle for Y millimeters. The X second rule also applies for the obstacle action.

In order to make the states discrete, robot movement steps are fixed to Z millimeters. In our experiment we used 500mm, and 1000mm steps. The result of this part of the behavior is a directory with all the observations, and a single file with the observations and the actions taken after it. One advantage of this behavior is that automatic and manual data gathering from different days can be merged easily.

Image Processing

After the robot has gathered sufficient observations from the environment, it starts processing the observations and actions done during exploration. Each image undergoes the following operations:

- Image Division:
  Two types of division are used in our method. For the first method, each image is divided into $n \times m$ number of cells and added to a list for further processes. In the second method, a hierarchical division is used. First the whole image is added to the list. Next, the row and column division is multiplied by two, thus the image is divided into 4 sub images. This process continues until the resolution of cells reaches $n \times m$. This method is used to keep hierarchical relations in the image.
- Edge Detection:
  In order to have a robust method, it is necessary to extract the salient features of the image. Therefore, we used the Canny edge detector which is one of the most popular edge detection methods currently used in image processing applications. The Canny edge mask will be calculated for each of the sub images.
- Histogram of Oriented Gradients computation:
  After the final Canny mask is calculated, we recalculate the gradient images in x and y directions. From these two images we compute the magnitude and orientation images. Next, we make a histogram with eight bins corresponding to eight major directions. Finally, we pass through the orientation image, and wherever the Canny edge image has a non-zero element, we increase the related histogram bin based on the current pixel's orientation. The magnitude of the gradient is used to scale the effect of the pixel in the histogram.
- Vector Extraction
  All the histograms of the sub images are put together to make a vector of real numbers.

Clustering

After the image processing phase, we will have a directory full of vectors which represents each image. It is essential to cluster these vectors to keep similar scenes in the same cluster for later reinforcement learning processes. It is well known that unsupervised clustering gives good results when the number of clusters in the underlying data is known. Therefore, based on the robot step size in the training mode, we make an estimation of the number of clusters. We know the step size. The possible number of different scenes in each location is four because of four major turns. Therefore, if we assume that the length and width of the environment are l and w, then we divide $l \times w$ by the step size and multiply it by four to get our estimated number of clusters. Another number of clusters can also be chosen for testing purposes. We use two major clustering methods, K-means and Neural Gas.

- K-means Clustering:
  The vectors will be loaded by our MATLAB program and K-means clustering is used to cluster the input data. First, the clusters are initialized to randomly selected data points. Furthermore, a batch K-means will be done on the data until the clusters are stable, then on the same results an online K-means will be done to further reduce the reconstruction error until clusters are stable or a maximum number of iterations has passed. The results are the filenames with their cluster labels and the cluster centers. All of these results are written to files for further use by the main program.

- Neural Gas:
  We also use a batch Neural Gas method to try to improve the clustering process. The main difference between Neural Gas and K-means is, when each cluster center moves, other clusters also move based on the neighborhood values. This method takes significantly more time compared to K-means but has often less reconstruction error compared to K-means.

Transition Probability Matrix

After the clustering phase, in each cluster, there should be images that were taken from the same location in the environment. Now, we want to know which actions were taken in each location and how many times a step was made to another cluster. This results in a transition probability matrix which is required for implementation and execution of the value iteration method. In a nutshell, our approach automatically divided the environment in different discrete states. Each state is a scene that indicates a certain location in the environment.

The transition probability matrix is an $n \times m \times n$ matrix, where n is the number of clusters/states, and m is the number of actions, which is four in our navigation scheme. We read the action file mentioned in the *data gathering* sub-section and, all the picture names will be replaced by their corresponding cluster numbers. Finally, the transition probability matrix is made from the updated action file and is written to file to be used later by the reinforcement learning method.

4.3.2. Testing Behavior

The idea of testing is to turn on the robot on a random location in the environment, give it a goal location by selecting a picture from the data base or taking a new picture from the environment, and ask the robot to go to the location. The robot first will localize itself by extracting the current location state as mentioned in section 4.3.1. Then it will localize the goal location using the same method. After the goal is selected, value iteration updates the Q-values and State values according to the goal. Next, using the Q-values an action will be selected. The robot has X seconds to execute the action, otherwise the action will be marked as failed, and a new action will be selected. If the robot sees an obstacle during movement, the obstacle avoidance method takes over and moves the robot up to the default step size for the behavior or up to X seconds have passed. The behavior stops as soon as the goal cluster is reached.

Value Iteration

As soon as the goal cluster is identified, the state value of that cluster will be set to one hundred. Then, for each state and action, we traverse all other states and check whether the transition probability matrix is bigger than zero. If it is, it means an action was taken during the training phase. The new Q-value will be calculated based on the value iteration algorithm presented in Figure 3. In our case, we stop the method as soon as the distance between the state value vector of one update round with the other is less than a certain threshold.

Action Selection

The action selection is based on the ϵ-greedy exploration policy. First, the action with the highest value is found. Next, a random number between zero and one is selected. If the number lies between 0 and 0.25, a random action will be selected. Otherwise, the action with the highest Q-value will be executed. This is used to overcome the problem of local minima and possible deadlocks in the system because of clustering errors.

Chapter 5

5. Experiments and Results

In this chapter we discuss the experiments done to test the performance of each part of our navigation system, namely, data gathering, image processing, and navigation.

5.1. Environment

The environment used for testing all parts of our method can be seen in Figure 13. This small arena is part of a bigger laboratory. The walls on the bottom are small artificial walls with height of 50cm. The robot can see most of the lab from inside the designated environment. We deliberately allowed the robot to see the outer part in order to test the method against changes in the environment. During all parts of the experiment, the outer and inner layers were changed (Chairs location, People walking outside the arena, People walking inside the area, artificial Obstacles inside the arena).

5.2. Image Processing Results

We are using histograms of oriented gradients as the base method to estimate states and localize the robot. Therefore, the results are heavily dependent on the edges extracted from the Canny edge detector. As mentioned in section 3.4.1, the Canny edge detector requires image smoothing and two threshold numbers. To find the best parameters, we selected several images from the database to extract the best parameter set for our navigation. The experimented thresholds can be seen in Table 1.

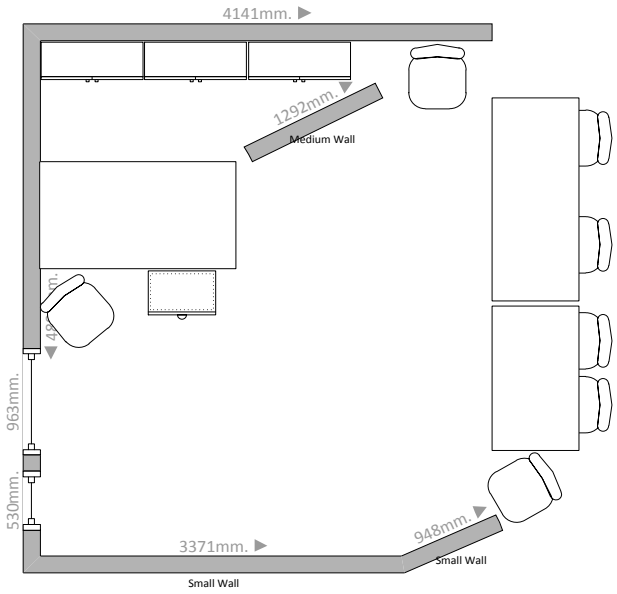

Figure 13 The environment used for training and testing

Table 1 Experimented parameters for Canny edge detector

Threshold 1	**Threshold 2**	**Smoothing ($\sigma = 1.25$)**
1	128	False
1	128	True
1	255	False
1	255	True
30	250	False
30	250	True

The result of these parameters can be seen in Figure 14. Figure 14(a) shows the original image. As can be seen, all images without smoothing, Figure 14 (b), (d), and (f), have more noise compared to their smoothed counterparts. Therefore we compare the smoothed pictures. We can see in Figure 14 (c) that all major edges are preserved. Chairs, battery, small wall, the magazine on the cupboard are all covered. However, the small textures on the mini wall on the left are also presented. Although the textures are detailed, texture information of the mini wall cannot be seen on the cupboard in the back of the screen. Figure 14 (e) shows much less noise compared to Figure 14 (c) but it also suppressed major edges from the chair which can severely affect the clustering

results. Figure 14 (g) is similar to Figure 14 (c). Therefore, to preserve all major edges and unique details, we select 1, 128 with smoothing as the parameters used for the Canny edge detector.

5.3. Clustering Results

Based on the results of the previous section, we smoothed the images with a 5×5 Gaussian filter and standard deviation of 1.25, and used 128 and 1 as the higher and lower threshold of the Canny edge detector. We used hierarchical picture division to extract the histograms of oriented gradients, the maximum resolution selected was 4×4. This means that we calculated the HoG of the original image, then divided the image into 2×2 cells and calculated the HoG, and then again divided the original image into 4×4 cells and calculated the HoG. Therefore, we extracted twenty one (1 + 4 + 16) histograms per image which means the feature vector has 168 dimensions. In order to cluster these images, we implemented two different clustering methods: K-means, and Neural Gas clustering. In addition, principle component analysis was used in our comparisons. We projected the data on twenty eigenvectors with the highest eigenvalues before clustering them.

In unsupervised clustering, the usual method to compare the algorithms is to check the reconstruction error. However, in our case, this alone does not help us. We need to find out which method performs better in putting pictures with similar topological information in the same cluster. Consequently, the best approach is to manually label pictures from the same location and test the clustering methods against them. We took 1400 images from the environment in 14 different states. The results can be seen in Table 2 and Table 3. In order to compute the success rates, for each labeled set, we compute the clustering results. If the images are the same as in the label set, and they are dominant in the cluster (more than 50%), we take them into account. If there are multiple clusters from the same original label, we calculated a weighted success rate. Since the number of clusters was limited, the results are also similar. The neural gas method is the winner with the best clustering and we are going to use this method in the next section. One main problem seen in the results was that a single labeled cluster was clustered as two different clusters or a single cluster contains multiple images from different goal clusters.

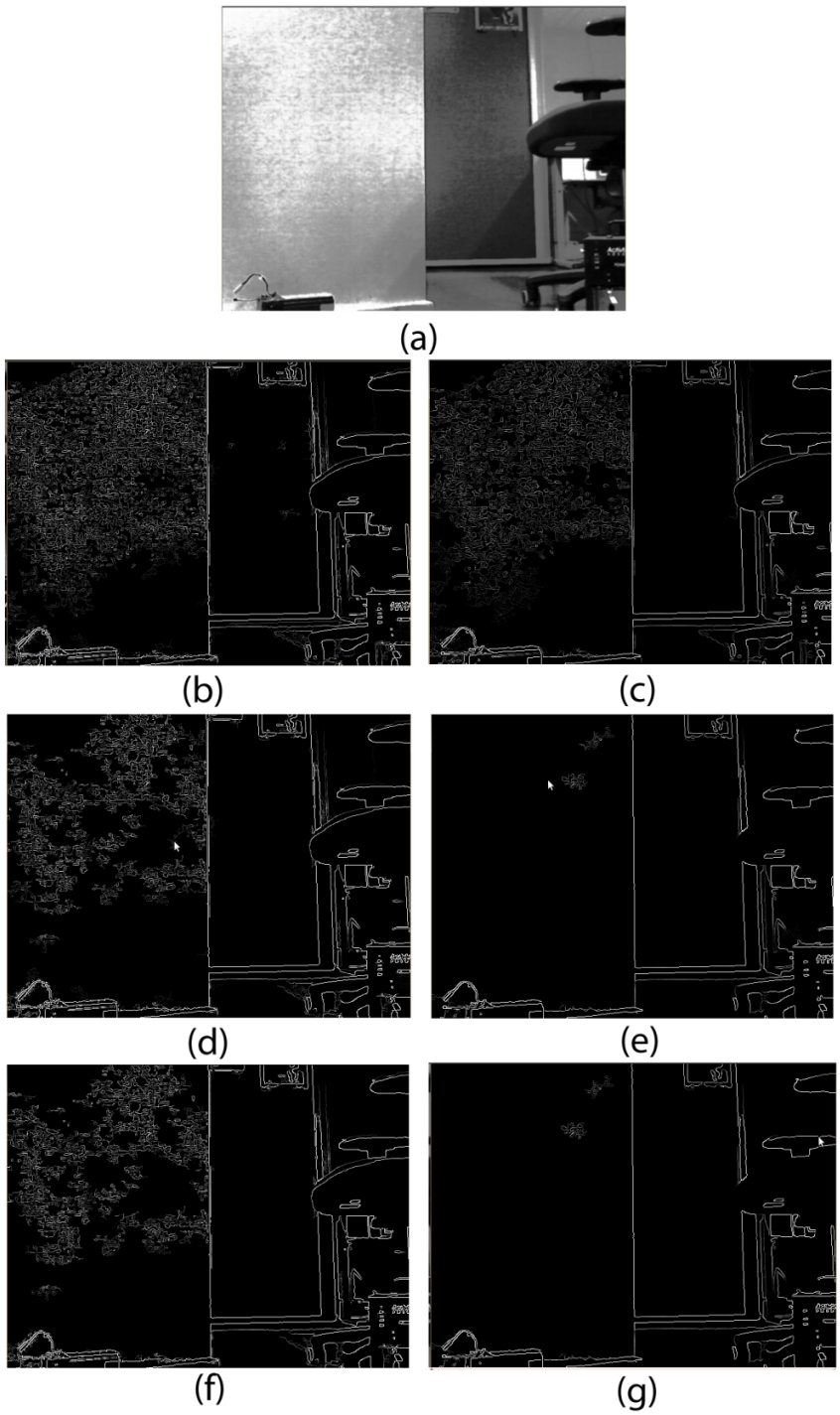

Figure 14 Canny edge detector results with different parameters (a) Original Image. (b) Threshold 1 is 1, threshold 2 is 128 and no smoothing. (c) Threshold 1 is 1, threshold 2 is 128 and with smoothing. (d) Threshold 1 is 1, threshold 2 is 255 and no smoothing. (e) Threshold 1 is 1, threshold 2 is 255 and with smoothing. (f) Threshold 1 is 30, threshold 2 is 250 and no smoothing. (g) Threshold 1 is 30, threshold 2 is 250 and with smoothing.

Table 2 Clustering Result. The methods used are K-means clustering, Neural gas, PCA K-means, and PCA Neural Gas

Cluster Number	Neural Gas Success Rate	K-Means Success Rate	PCA-Kmeans Success Rate	PCA-Neural Gas Success Rate
1	0.92	0.95	0.67	0.91
2	1.00	1.00	0.88	0.88
3	0.54	0.42	0.59	0.51
4	0.76	0.76	0.75	0.81
5	0.50	0.65	0.55	0.40
6	0.45	0.45	0.25	0.40
7	0.85	0.85	0.85	0.45
8	0.57	0.51	0.88	1.00
9	0.80	1.00	0.66	0.91
10	0.50	0.57	0.32	0.58
11	0.57	0.53	1.00	0.80
12	0.22	0.20	0.72	0.40
13	0.45	0.50	0.00	0.47
14	1.00	0.60	1.00	1.00

This problem is well known for K-means and neural gas. However, the neural gas method usually tackles this problem better. The reason is that all prototypes are connected to each other in neural gas. As soon as one prototype changes, its neighboring prototypes also change, thus, reducing the probability of having multi-mode centers and dead units. In our results, we ignored the multi-mode problem if the points with the correct labels inside the cluster were dominant.

Table 3 Final weighted results of clustering

Method	Total Weighted Success Rate
Neural Gas	68.25%
K-Means	68.24%
PCA Neural Gas	68.16%
PCA K-Means	65.57%

We can also conclude from Table 3 that the success rate of projecting the data set onto the first twenty eigenvectors with the highest eigenvalues is lower than the non-

projected method. The result of both neural gas and K-means clustering using principle component analysis is lower than the normal methods. This shows that PCA is not able to factor out dimensions from the feature vector without increasing the final clustering error. We have to mention that the differences are not very significant.

5.4. Navigation Results

In the previous section, we concluded that neural gas is the best method to be used in our navigation system. We are going to compare the results of our method with neural gas clustering and value iteration with random search in which the robot randomly selects actions. Therefore, in the next sections we present scenarios to test our reinforcement learning approach in combination with HoG and clustering. For these experiments, the robot step size was 50cm and we used 10 percent randomness for our action selection. The number of clusters used was 300 and each experiment was repeated 5 times.

5.4.1. Scenario 1, two starting locations

In the first scenario, one goal and two starting positions were selected which can be seen in Figure 15(a). We let each of the methods run for a maximum of 500 seconds to reach the goal. The results for this scenario can be seen in Table 4. The reinforcement learning with neural gas clustering was able to reach the goal from both starting

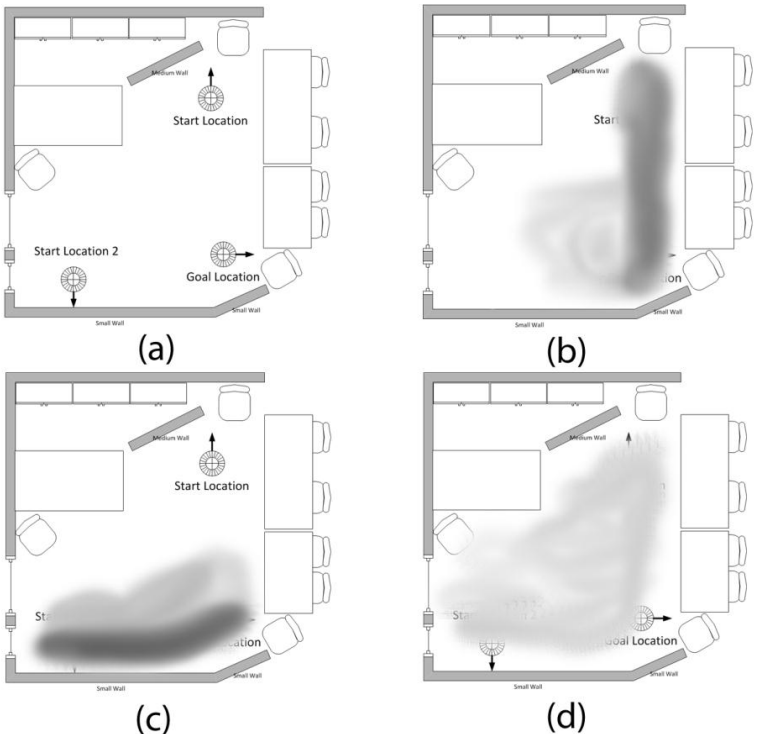

Figure 15 Scenario 1. (a) Start and Goal Loactions. The arrows show the used direction to start or finish. (b) Trail of the robot using neural gas and value iteration from first start location. (c) Trail of the robot using neural gas and value iteration from second start location. (d) Trail of the robot using random search

positions and the performance was superior to that of random search. However, the performance is significantly weaker than human navigation. By looking at the trail of the robot in Figure 15 (b) and Figure 15 (c), we can conclude that the reinforcement learning is correctly distributing the state values, since the robot trail is close to the path to the goal location. Therefore, the problem seems to be the error in clustering and robot orientation. During our observations, we saw that the robot headed toward the goal, moved near the designated location, but because it was not exactly on the right spot (less than a robot step size), it did several loops in order to finally set the "goal reached" flag. Since, the movements are discrete, and the robot odometry has errors, either a less number of clusters should be selected or multiple goal clusters should be marked as the final destination to improve the performance. The interesting point is that in the experiment we did not see a false positive goal that set the "goal reached" flag from the robot. This means that the number of clusters was high enough to avoid false positive results. If the clustering method results are not consistent with the underlying real data, the state value distribution may result in strange robot behaviors during navigation. For example, if an underlying real cluster (a similar scene) is separated into two clusters by our method, there will be no connection between these two states, and it will be an invisible wall during navigation. This behavior was seen when the second starting location was used, and because of this, the results are worse.

Table 4 Scenrio 1 results.

First Start Location	Success Rate	Average Action Numbers	Average Time to reach goal
Random Search	40%	33	225s
Value Iteration with NG	80%	42	243.75s
Second Start Location			
Random Search	0%	N/A	N/A
Value Iteration with NG	40%	37.5	180s

5.4.2. Scenario 2 and 3, one starting location

For the last two scenarios we selected two different goal locations with one starting location. The results can be seen in Table 5. The same arguments that were mentioned in section 5.4.1 also apply here. Another difficulty that we think reduces the

performance is the obstacle avoidance effect. During training, the robot starts from a special heading, and for turning it does a 90^o turn to either direction. When the robot avoids an obstacle, the control will be given to the obstacle avoidance. The problem arises when the obstacle avoidance puts the robot in another type of orientation. For example, the robot starts with the heading of 0 degrees. It turns 90 degrees to the right and recognizes an obstacle, and then the obstacle avoider takes control and avoids the obstacle. The new heading of the robot is now 30 degrees which will result in a complete change in the orientation, and subsequent observations. Therefore, during testing, since it is not possible to turn 30 degrees with our actions, the robot will try to repeat the scenario and hope that the obstacle avoidance leads it to the correct state. This can be fixed by reducing the turn angles, or reduce the number of clusters.

Table 5 Scenario 2 and 3 results.

Scenario 2	Success Rate	Average Action Numbers	Average Time to reach goal
Random Search	40%	45	275s
Value Iteration with NG	60%	32.3	306.33s
Scenario 3			
Random Search	20%	47	477s
Value Iteration with Neural Gas	40%	31.5	290s

5.4.3. Discussion of Results

In this chapter we compared the value iteration method using neural gas clustering to discretize states to random search. Three scenarios with different goal and starting locations were used. In all of the scenarios the proposed navigation method performed better than random search. However, the results are considerably weaker than human navigation. We observed that the robot was moving in the path to the goal, but could not find the final goal cluster because of odometry and unsupervised clustering errors, and changes in the orientation of the robot after observing an obstacle.

Chapter 6

6. Conclusion and Future Work

In this thesis, we first performed a literature study on current research in indoor navigation methods. We found out that most of the methods used in robotics required manual user interference, and standardized or fixed environment during training. Some of these methods are prone to failure if changes to the environment were made during testing. We found out that there is a significant difference between these methods and the human navigation system. Humans navigate mainly based on topological information, and known landmarks. However, adults not only use their experience and knowledge about landmarks to navigate but they also use semantics, understanding of physical laws, and common sense. Therefore, we decided to imitate the human navigation system using only the part which is about visual memory and topological information. To achieve this goal, we presented state of the art reinforcement learning methods to imitate the human's learning loop. We continued by discussing the most applied image processing methods to extract topological information. Furthermore, for our model-based reinforcement learning approach we used clustering on our observations to discretize them.

We separated the navigation system in training and testing phases. During both phases, minor changes in the surrounding environment were allowed and were enforced. The results in our experiment section showed that our proposed method works better than random search but considerably weaker than human navigation. Our observations

approved that the reinforcement learning correctly distributed the state values. The problem emanated from the fact that unsupervised clustering is an ill-defined problem, and the methods used could not guarantee successful clustering of the underlying data. Other issues such as limited turning actions and change in orientation of the robot because of obstacles in the environment deteriorated the performance even further. We will now first answer the research questions and then outline future work for improvement of the proposed methodology.

Since the unsupervised clustering is by definition an ill-posed problem, we require prior information to reduce the errors. One way to moderate these inaccuracies is to take into account the odometry of the robot and connect the visual states to three dimensions of robot movement (x,y, and robot pose). However, the robot odometry itself is prone to errors over time and requires correction. One approach to solve this problem is to do a two way correction of labeling and odometry. This approach is logical since the error of odometry increases over time. For example, the robot should take several pictures on the starting point, and label all of them. Next, it will automatically select an action, and goes to another state. Based on odometry data, new clusters should be labeled. We continuously should train the supervised classifier with introduction of new data. The robot will move back toward previously learned locations to fix its odometry. Supervised classifiers such as support vector machine, learning vector quantization (Bunte, Schneider, Hammer, Schleif, Villmann, & M., 2011) (Schneider, Biehl, & Hammer, 2009), and neural networks can be used to achieve this goal.

In addition we can increase the reliability of our topology information extractor by calculating the relation of edges in each picture cell in addition to HoGs. The number of corners, edge connections, and arrangement of edges are important and cannot be deduced from histograms of oriented gradients.

The robot selects actions after a previous action is complete, and action selection requires small amount of processing time. This results in non-smooth movements of the robot. In order to solve this problem, we suggest using a queue of commands. This means, that the robot optimistically assumes that its selected actions using the value iteration method will be done without any errors. A sequence of actions will be selected based on this assumption using the value iteration method. During the execution, the robot continues perceiving the environment and calculates the correct actions. The robot continuously checks whether the actions are done as planned. If something goes wrong, the robot uses its past history of actions to repair and fix its path.

Bibliography

Alpaydin, E. (2004). *Introduction to Machine Learning.* MIT Press.

Arkin, R. C. (1998). *Behavior-based robotics.* MIT Press.

Booij, O., Terwijn, B., & Zivkovic, Z. (2007). Navigation using an appearance based topological map. *Robotics and Automation* (pp. 3927-3932). IEEE.

Bunte, K., Schneider, P., Hammer, B., Schleif, F.-M., Villmann, T., & M., B. (2011). Limited rank matrix learning - discriminative dimension reduction and visualization. *Neural Networks*.

Canny, J. (1986). A Computational Approach for Edge Detection. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 679-698.

Dalal, N., & Triggs, B. (2005). Histograms of Oriented Gradients for Human Detection. *In Proceedings of the 2005 IEEE Computer Society Conference on Computer Vision and Pattern Recognition. 1*, pp. 886-893. Washington, DC, USA: IEEE Computer Society.

Dissanayake, M. W., Newman, P., Clark, S., Durrant Whute, H. F., & Csorba, M. (2001). A solution to the Simultaneous Localization and Map Building (SLAM) problem. *IEEE Transactions on Robotics and Automation*, 17(3), 229-241.

Folkesson, J., & Christensen, H. (2003). Outdoor Exploration and SLAM using a Compressed Filter. *In Proceedings of the IEEE international Conference on Robotics and Automation*, (pp. 4129-426).

Frese, U., & Hirzinger, G. (2001). Simultaneous Localization and Mapping - A dicussion. *In Proceedings of the IJCAI Workshop on Reasoning with Uncertainty in Robotics*, (pp. pages 17–26).

Gates, B. (2007, January). A Robot in Every Home. *Scientific American*, p. 58.

Goedeme, T., Nutting, M., Tuytelaars, T., & van Gool, L. (2007). Omnidirectional vision based topological navigation. *International Journal of Computer Vision*, 219-236.

Gonzalez, R. C., & Woods, R. E. (2008). *Digital Image Processing.* Pearson Education.

Kaelbling, L. P., Littman, M. L., & Moore, A. W. (1996). Reinforcement learning: A survey. *Journal of Artificial Intelligence Research*, vol. 4, pp. 237–285.

Kohonen, T., & Somervuo, P. (1998). Self-organizing maps of symbol strings. *Neurocomputing*, 19--30.

Kushner, H. (1990). Numerical Methods for Stochastic Control Problems in Continuous Time". *SIAM Journal on Control and Optimization*, 999--1048.

Leonard, J., & Durrant-Whyte, H. (1991). Simultaneous map building and localization for an autonomous mobile robot. *IEEE/RSJ International Workshop on Intelligence for Mechanical Systems, Proceedings IROS*, *3, 3-5*, pp. 1442-1447.

Littman, M. L. (1996). *Algorithms for sequential decision making. Ph.D. Thesis.* Brown University.

MacQueen, J. B. (1967). Some Methods for classification and Analysis of Multivariate Observations. *Proceedings of 5-th Berkeley Symposium on Mathematical Statistics and Probability* (pp. 1:281-297). Berkeley: University of California Press.

Maguire, E., Burgess, N., & O'Keefe, J. (1999). Human spatial navigation: cognitive maps, sexual dimorphism, and neural substrates. *Current Opinion in Neurobiology, 9*, 171--177.

Maguire, E., Frith, C., Burgess, N., Donnett, J., & O'Keefe, J. (1998). Knowing where things are: Parahippocampal involvement in encoding object locations in virtual large-scale space. *Journal of Cognitive Neuroscience, 10*, 61--76.

Martinetz, T., & Schulten, K. (1991). A" neural-gas" network learns topologies. (T. Kohonen, K. Mäkisara, O. Simula, & J. Kangas, Eds.) *Artificial Neural Networks*, pp. 397-402. .

Moore, A. W., & Atkeson, C. G. (1993). Prioritized sweeping: Reinforcement learning with less data and less time. *Machine Learning*, 13:103-130.

Samuel, A. L. (1959). Some studies in machine learning using the game of checkers. *IBM Journal on Research and Development*, 3:210-229.

Schneider, P., Biehl, M., & Hammer, B. (2009). Adaptive relevance matrices in learning vector quantization. *Neural Computation, 21*, 3532--3561.

Simmons, R., & Koenig., S. (1995). Probabilistic robot navigation in partially observable environments. *In International Joint Conference on Artificial Intelligence*, (pp. 1080-1087).

Smith, R., & Cheeseman, P. (1986). On the Representation and Estimation of Spatial Uncertainty. *The International Journal of Robotics Research*, 5 (4): 56–68.

Sutton. (1988). Learning to predict by the methods of temporal difference. *Machine Learning*, 3:9-44.

Sutton, R. S., Precup, D., & Singh, S. P. (1998). *Between MDPs and semi-MDPs: Learning, planning, learning and sequential decision making. Technical Report COINS.* Amherst: University of Massachusetts.

Sutton, R., & Barto, A. (1998). *Reinforcement learning: An introduction.* Cambridge Univ Press.

Thrun, S. (1998). Learning metric-topological maps for indoor mobile robot navigation. *Artificial Intelligence Journal*, 99(1):21–71.

Thrun, S. (2002). Robotic mapping: A survey. *Exploring artificial intelligence in the new millennium*, 1--35.

Thrun, S., Fox, D., & W., B. (1997). A probabilistic approach to concurrent mapping and localization for mobile robots. *Machine Learning and Autonomous Robots*, 31(5) 1-25.

Tomatis, N., Nourbakhsh, I., & Siegwart, R. (2003). Hybrid simultaneous localization and map building: a natural integration of topological and metric. *Robotics and autonomous systems*, 44(1):3-14.

Valgren, C., Duckett, T., & Lilienthal, A. (2007). Incremental spectral clustering and its application to topological mapping. *In Robotics and Automation*, 4283-4288.

Watkins, C. J. (1989). Learning from Delayed Rewards. *Ph.D. thesis*. Cambridge, England: King's College.

Watkins, C. J., & Dayan, P. (1992). Q-Learning. *Machine Learning*, 8:279-127.

Weng, J., McClelland, J., Pentland, A., Sporns, O., Stockman, I., Sur, M., et al. (2001). Autonomous mental development by robots and animals. *Science*, 291(5504):599–600.

Wiering, M. (1999). *Explorations in Efficient Reinforcement Learning. Ph.D. thesis.* Amsterdam: University of Amsterdam.

Zivkovic, Z., Bakker, B., & Krose, B. (2005). Hierarchical map building using visual landmarks and geometric constraints. *Intelligent Robots and Systems* (pp. 2480-2485). IEEE.

Acknowledgements

I would like to thank Dr. Marco Wiering and Prof. Michael Biehl whose guidance and support from the initial phase to the end of my thesis enabled me to develop a perfect understanding of the topic and successfully finish my research.

I would like to further thank Dr. Tijn van der Zant, Prof. Lambert Schomaker, and the graduate school of science. Their support and assistance enables me to continue my Ph.D studies in robotics at the University of Groningen.

I am heartily thankful to my wife, Sara, for her encouragement, and support throughout my whole studies, and especially my thesis. Without her help, this thesis would have not been finished successfully.

Lastly, my deepest gratitude goes to my parents, Aliye and Mahdi, who supported me in every moment of my life and gave me their unflagging love, to my brothers, Ali and Shahram, from whom I learned how to step through life and its mysteries and to my sister, Maryam from whom I understood patience and calmness.