Departamento de Engenharia Electrotécnica e de Computadores
Faculdade de Ciências e Tecnologia
Universidade de Coimbra

Intelligent Robot Navigation Using a Sparse Distributed Memory

Mateus Mendes

November 2010

A Dissertation
for Graduate Study in Ph.D. Program
Doctor of Philosophy in Electrical and Computer Engineering

Intelligent Robot Navigation Using a Sparse Distributed Memory

Mateus Daniel Almeida Mendes

Departamento de Engenharia Electrotécnica e de Computadores
Faculdade de Ciências e Tecnologia
Universidade de Coimbra

November 2010

Research Developed Under Supervision of
Prof. Doutor Manuel Marques Crisóstomo and
Prof. Doutor António Paulo Mendes Breda Dias Coimbra
Departamento de Engenharia Electrotécnica e de Computadores
Faculdade de Ciências e Tecnologia da Universidade de Coimbra, Portugal

Supported in part by grant SFRH/BD/44006/2008 from Fundação para a Ciência e Tecnologia,
and also by the Higher School of Technology and Management of Oliveira do Hospital,
of the Polytechnic Institute of Coimbra. Work developed in the Institute of Systems and Robotics.

Thanks

Unselfish and noble actions are the most radiant pages in the biography of souls.
(David Thomas, American writer)

First of all, I'm in great debt to Cristina, for all the time and attention that she deserved and I took from her to devote to this work over the years, besides my full time job as adjunct professor.

Thanks also to my supervisors, from whom I also have withdrawn so many hours. Thanks to Sascha Jockel, for all the invaluable suggestions and reviews—it was a pleasure to work with you. Thanks Markus Carpenter for proof reading part of this work. Thanks Fernando Coelho, for the help in handling some of the statistical concepts and issues. I must also write a line to thank Paulo Brígida, who promptly gave me so many useless and some few useful suggestions.

I must also leave a word of appreciation to Professor Douglas Hofstadter, as well as Elísio Sousa, Jorge Maciel and Jonathan Murray, for the insightful discussions about the Sparse Distributed Memory and other related ideas. Thanks also Professor David Willshaw for the prompt replies about his work when requested. And thanks to other authors, researchers, colleagues, students and friends to whom I am indebted and whose names I must not list one by one to keep this section short and readable.

Thanks also to the Polytechnic Institute of Coimbra and Fundação para a Ciência e Tecnologia for sponsoring part of this work.

Abstract

It is wise to keep in mind that neither success nor failure is ever final.
(Roger Babson, American educator)

The dream of building intelligent machines dates back thousands of years. However, despite the huge advancements in science and technology, modern robots are still fit only to very specific tasks. There are no general-purpose robots able to learn and perform any kind of task the way humans do. Part of the problem arises from the fact that the concept of intelligence is not clearly defined, implying that researchers are pursuing a moving target. This work reviews some important concepts about intelligence, then presents a system which navigates a robot using visual memories stored into an associative memory. The system is *intelligent*, in the sense that it selects only relevant images and data to store, and infers new routes from the known paths.

Many algorithms have already been proposed to navigate mobile robots. In this work the goal is to use a Sparse Distributed Memory (SDM), a kind of associative memory proposed by Pentti Kanerva, based on the properties of high dimensional binary spaces. Some characteristics of SDMs are that they are very immune to noisy data, can deal with incomplete information, can learn on a single pass and *naturally* forget older memories when close to their maximum capacity. They work very well on high dimensional spaces, and are also appropriate to work with sequences of events, where each event is associated with its successor or predecessor. The SDM naturally confers on the robot many characteristics of the human memory, thus being a powerful and general model, comparable to or better than other navigation methods.

The goal of the work described was to build a system that can navigate a robot using sequences of visual memories stored into a sparse distributed memory. During a supervised learning stage, the robot acquires images from its built-in front camera. Selected grabbed images are stored into the SDM. Each image is accompanied with some additional information, such as the motion the robot performed just after capturing it, the number of the sequence and the position of the image in the sequence. That makes it possible for the robot to repeat the same path autonomously later. A simple algorithm that

uses a block matching technique was also implemented, to correct small horizontal drifts that may occur during autonomous robot navigation. In order to make the system more robust and improve the quality of the raw images, they can be equalised or contrast stretched before being passed on to the SDM.

The proposed approach works very well. However, as soon as the system was implemented and the first tests were run, one major problem arose. The SDM model is suitable to work with random boolean data. Nonetheless, sensorial data is often encoded using the natural binary code. In the case of the images used for navigation, each pixel is represented by an 8-bit graylevel. The dimension of the pixel space is, therefore, the integer interval [0, 255], not boolean. The difference of methods used to capture the information and to process it into the SDM affects the performance of the SDM. In order to get a better insight into the problem and overcome it, four different operation modes were implemented, three using the Hamming distance and one using an arithmetic distance. The performance of the SDM was tested for all four methods, in the domains of robot navigation and manipulation. To the best of the author's knowledge, this is the first time those comparisons were performed, and that is one major contribution of the present work.

Since the SDM was proposed in the 1980s, it has been subject to intense research and many authors have proposed improvements. However, so far it was not clear how robust the SDM would be in practice, specially in the task of robot navigation. In the present work, different versions of the SDM were tested facing different problems that robots typically have to face in order to navigate by using visual information. That includes strong illumination changes, scenario changes and memory overflow. That is another important contribution of this work, since this is possibly the first time such tests have been performed.

Finally, the last major contribution of this work is an algorithm to build a topological representation of the environment using images and a SDM. The SDM was used both to store the images and as a pattern recognition tool to detect overlaps of paths. Using that topological representation it was possible to plan new routes, different from the ones that were taught. The results show that the method proposed is appropriate to successfully navigate mobile robots, based on visual memories, and the arithmetic SDM shows the best performance at the lowest computational cost.

Resumo

É sábio ter em mente que sucesso e fracasso nunca são definitivos.
(Roger Babson, educador Americano)

O sonho de construir máquinas inteligentes data de há milhares de anos. No entanto, e apesar dos grandes avanços científicos e tecnológicos, mesmo os mais modernos robôs ainda são desenvolvidos apenas para tarefas muito específicas. Não há robôs "genéricos", capazes de aprender e fazer qualquer tarefa da mesma forma que os humanos normalmente fazem. Parte do problema decorre do facto de ainda não haver uma definição clara do que é a inteligência. Este trabalho apresenta uma revisão dos mais importantes conceitos sobre o que é a inteligência. Depois apresenta um sistema que um robô navega usando memórias visuais, guardadas numa memória associativa. O sistema é *inteligente*, no sentido que selecciona apenas as imagens e dados mais importantes para guardar na memória, e infere novos caminhos partindo daqueles que já conhece.

Já foram propostos muitos algoritmos para navegar robôs móveis. Neste trabalho o objectivo foi usar uma Memória Esparsa Distribuída (SDM), um tipo de memória associativa proposta por Pentti Kanerva, baseada nas propriedades de espaços binários de elevada dimensão. Algumas das suas mais importantes características são o facto de serem muito tolerantes a informação com grandes quantidades de ruído, poderem lidar com informação incompleta, poderem aprender numa única passagem (*one-shot*) e esquecerem *naturalmente* memórias mais antigas. Além disso, as SDMs são naturalmente apropriadas para trabalhar com vectores de elevada dimensão e sequências de eventos, onde cada evento é associado ao respectivo sucessor ou predecessor. Assim, a SDM confere naturalmente ao robô algumas características da memória humana, sendo um modelo poderoso e generalista, comparável ou potencialmente melhor do que outros modelos de navegação de robôs geralmente utilizados.

O objecto do presente trabalho foi a construção de um sistema para navegar um robô usando sequências de memórias visuais gravadas numa memória esparsa distribuída. Durante uma fase de aprendizagem supervisionada, o robô adquire imagens de uma câmara frontal. Imagens consideradas relevantes são guardadas na SDM. Com cada imagem é gravada também informação adicional, que inclui

o movimento que o robô executou quando a imagem foi capturada, o número da sequência (caminho) e a posição da imagem na sequência. Usando essa informação, o robô é posteriormente capaz de repetir o mesmo caminho autonomamente. Um algoritmo simples, baseado numa técnica de comparação de blocos (*block matching*), permite detectar e corrigir pequenos desvios (*drifts*) que possam ocorrer durante a navegação autónoma. De forma a tornar o sistema mais robusto e melhorar a qualidade das imagens, estas podem ser equalizadas ou normalizadas (*contrast stretched*) antes de serem gravadas na SDM.

O método descrito anteriormente produz bons resultados. Contudo, logo que o sistema foi implementado e executados os primeiros testes, detectou-se um problema importante. O problema tem a ver com o facto da SDM ser apropriada para funcionar em espaços booleanos. No entanto, a informação sensorial dos robôs normalmente não é booleana: é codificada usando o código binário natural. No caso das imagens em níveis de cinza usadas neste trabalho, cada pixel é representado por um número de 8 bits. A dimensão do espaço de um pixel é, portanto, o intervalo inteiro [0, 255], não um espaço booleano. Esta diferença de métodos de codificação usada para recolher a informação dos sensores e para processá-la dentro da SDM afecta significativamente o desempenho da memória. De forma a compreender melhor o problema e resolvê-lo, foram implementados quatro modos de operação da memória, correspondendo a quatro codificações diferentes da informação manipulada. O desempenho do sistema foi testado para cada um dos quatro modos, nos domínios da navegação e da manipulação robótica. Essa é uma das contribuições mais importantes deste trabalho.

Desde que a SDM foi originalmente proposta, na década de 80 do século XX, foram construídos vários modelos e publicadas várias propostas de melhoria do modelo original. No entanto, até ao momento não eram claras as potencialidades e limites da memória, especialmente na tarefa de navegar um robô com base em memórias visuais. Neste trabalho, as várias versões da SDM foram testadas em situações que são problemas típicos da navegação baseada em imagens. Designadamente, situações de variação da iluminação, alterações do cenário e saturação da memória. Essa é outra importante contribuição desta tese.

Finalmente, a última contribuição deste trabalho é um algoritmo para construir uma representação topológica do ambiente usando as imagens e a SDM. A SDM é usada simultaneamente para guardar as imagens capturadas e como ferramenta de reconhecimento de padrões para detectar sobreposição de trajectórias. Desta forma, foi possível usar a representação topológica para planear trajectórias novas, diferentes das ensinadas. Os resultados mostram que o método proposto é adequado para guiar robôs móveis, baseados em memórias visuais, e que a SDM aritmética exibe o melhor desempenho com o menor custo computacional.

Contents

Chapter 1

Introduction

If you have an apple and I have an apple and we exchange apples then you and I will still each have one apple. But if you have an idea and I have one idea and we exchange these ideas, then each of us will have two ideas.
(George Bernard Shaw, Irish dramatist)

Contents

The ultimate goal of robotics is to build intelligent machines, fit for a myriad of different jobs. In the last 50 years, the emergence of a new research area, called Artificial Intelligence (AI), brought new light and opened new perspectives to this field [13, 89, 78]. This chapter briefly reviews some importand historic aspects and the main problems encountered during the quest towards intelligent machines. It also gives an outline of the proposal described in this thesis and presents the thesis structure.

1.1 A dream thousands of years old

Robotics and Artificial Intelligence are usually regarded as modern research areas. Indeed, the term "Artificial Intelligence" was coined just in the 20^{th} century, as the main subject of a conference that was held in Dartmouth in 1956 [68]. But the key ideas of artificial beings can be traced back thousands of years [13, 89, 78]. This section briefly reviews some important historic landmarks, from ancient to modern history, more related to the theme of intelligent robot navigation.

1.1.1 The origins

The modern word "computer" was derived from an old job. The concept was adjusted to be applicable to machines. But it's not only the word that is hundreds of years old. The dream of building intelligent machines, and some important contributions to the field, date back to at least the ancient Greeks, Egyptians and Chinese.

The word computer

It is a popular saying that "while humans don't build intelligent robots, humans are the robots". Actually, that was literally true with computers. The term "computer" was used to refer to a human job that consisted in performing routine calculations on a daily basis. Those human computers spent days calculating navigational tables, tide charts, planetary positions for astronomical almanacs and other tasks which are currently performed by modern digital computers. However, humans get bored and tired with ease, and inevitably start making more and more mistakes as time goes by. Besides, they also take their time—usually more than a machine for that kind of jobs. As soon as the technology was developed, machines were built to make those routine tasks automatically. The process was sped up specially during the Second World War, when top scientists were called to break German military's cipher [41]. Those first machines were called "automatic computers", in order to be differentiated from the "human computers". That started happening as early as the 1920s [79].

Credits to ancient civilisations

The quest for intelligent machines is even older than the birth of the modern computers. As early as 2500 B.C., the Egyptians developed the concept of "thinking machines," which were regarded with some mysticism. There are reports that back in the ancient Egypt, 800 B.C., a statue was built that could move its arm and speak to onlookers. However, its "intelligence" was that of a man that could fit inside the husk [34]. Nonetheless, those statues that could move their limbs were the first automata, which

are the predecessors of the modern robots.

The concept of robots is also very present in the Greek mythology. According to the legends, Hephaestus, the God of the smiths, discovered ways of working iron, copper and other metals. Using that expertise, he built mechanical servants to serve him[1]. One of those amazing creatures made by Hephaestus was Talos_1. Talos_1 was a robot made of bronze, who guarded Crete by running round the island three times every day. When intruders appeared at the island, he pelted them with stones. There are many different versions of the legend, but the most interesting is perhaps that in which Talos_1 is a *man*, the last of a generation of men made of bronze [91].

Besides the mythological concepts, another important contribution to building intelligent machines took place in the ancient Greece. The Greek philosopher Aristotle, in the 5^{th} century B.C., created the "syllogistic logic". The syllogistic logic is the origin of the deductive reasoning and the basis for implementing intelligent behaviours in artificial agents.

Those concepts of robots and intelligent reasoning had to be combined with some technology in order to produce the first machines (and even the most advanced modern robot is not even close to the mythic Talos_1). The oldest ingenious machine is possibly the abacus. It's not clear when the first abacus was created, but it is usually credited to the Babylonians, who might have developed the first abacus circa 2400 B.C. [95]. The abacus is just an auxiliary memory to the human users who have to perform the calculations. It helps and speeds up the process of making basic arithmetic operations, namely sums and subtractions. Division and multiplication are more complicated to perform using abaci.

There are also reports of an ingenious device, called the "South Pointing Chariot," which may have been invented in China as early as 2600 B.C [19]. The South Pointing Chariot was possibly the first mechanical device that used differential gearing. The mechanism was built in a way that a figure (figure of a hand, dragon, etc.) atop the chariot maintains its heading, always pointing the South, regardless of the path travelled by the chariot. Different versions of the device have been built. The worst versions accumulated a large error in just a few turns. The best versions were able to handle hundreds of turns with just a minimum error, thus being useful navigation devices for those who could afford such an equipment.

1.1.2 Modern history

Modern history can be described as an exponential line of successes in various subfields. Nonetheless, it is made of ebbs and flows in each subfield. From the first mechanical calculator built in the 17^{th} century to the modern humanoid robots that can play soccer [7], there have been periods of great optimism,

[1] http://homepage.mac.com/cparada/GML/Hephaestus.html (last checked 2010.03.16).

Figure 1.1: Picture of a replica of the first commercial calculator, which is in exhibition at the Science Museum in London[4].

but also "winters" of doubt which are only broken when there's a new breakthrough.

The first calculator

Many centuries passed since the first attempts to build intelligent machines. Developments in mathematics and physics were very slow, but in the 17^{th} century the first mechanical calculators were built. The Pascalina[2] (see Figure 1.1), built by Blaise Pascal, was the first commercial calculator. It had only a very limited ability to sum and subtract. Therefore, it was just a more sophisticated abacus, based on gears instead of movable stones. During his youth, Pascal built about 50 prototypes and sold about 12 units of those Pascalinas, before loosing interest in the commercialisation of the device.

Robots and Artificial Intelligence

The term "robot" was used for the first time by Karel Capek, in a science fiction novel published in 1921 in the Czech Republic [15]. Capek envisioned machines created as mechanical slaves to work for human masters, and called those machines "robots". In the novel, the robots are treated poorly, but they are "happy" until they are programmed to incorporate emotions. At that point, the robots rebel in order to take over the world and get free from their human masters.

Out of the science fiction world, many innovations took place in the early years of the the 20^{th} century. The first and second World Wars sped up the technological developments. The first modern computer was built in the 1940s, mostly with technology developed during the war time. The end of the Second World War was actually a period of great optimism, in what concerns technological development.

[2]http://www.computernostalgia.net/articles/pascalina.htm (last checked 2010.03.16).

[4]http://www.sciencemuseum.org.uk/images/I032/10302630.aspx (last checked 2010.03.16).

The technologies developed in the military industry were released for civil applications, for the benefit of the public in general. Science was making widely available artifacts which a few decades before were just visions of science fiction writers. Therefore, almost everything seemed possible to accomplish in a very short time frame. It was expected that in a few decades intelligent robots would be everywhere, performing not only repetitive computing tasks, but also tasks which required high-level reasoning, learning and ability to perform intelligent decisions, as well as other skills which usually can only be achieved by human workers. In the 1950s many authors were optimistic enough to predict that in 10–20 years computers would be the world chess champions[5] and uncover new mathematical principia [89].

The term "Artificial Intelligence" (AI) was coined in 1955, by John McCarthy [68]. AI was meant to be the subject of a scientific summit in Dartmouth. That summit brought together some of the most important researchers at that time, in order to discuss the new scientific and technological developments. The subject of the meeting actually became the name of a whole new area of research.

Ebbs and flows

The initial optimism and faith in the achievements of science and technology encountered unexpected barriers, when the ideas were to be put into practice. Apparently simple tasks turned out to be incredibly complicated to perform using robots or computers. The first huge problem spotted was that computers were actually performing simple algorithms, with little knowledge of their own about the context. They had no *awareness* and no *common sense*, which seem to be basic human skills that guide many decisions and thought processes. That lack of *awareness* is in the origin of the still open question launched by John Searle later in the 1980s, which is discussed in more detail in Chapter 2.1.2.

In the 1970s AI has gone through what is known as the first "AI winter"—about six years of disappointment with the results obtained and extensive cuts in research funds. In the early 1980s, the rise of expert systems caused another wave of optimism. Expert systems are simple computer programs that, using logical rules which are inferred from the knowledge of human experts, mimic their actions and answer questions correctly. That approach was very welcome by companies, which could then use machines to replace human experts, saving money and reducing the risk of human failure and boredom. In the late 1980s, though, the enthusiasm faded again, in part due to the realisation that expert systems are not a general solution to all kind of reasoning problems. Expert systems are only fit for a small number of decision-making applications. By that time, a new debate arose, with many authors questioning if intelligence was possible without a body. That question is in the origins

[5] Actually, it was only in 1997 that the supercomputer Deep Blue was able to beat the world chess champion Garry Kasparov. However, Deep Blue was not a mere "general purpose" computer. Deep Blue was a supercomputer designed specifically for the purpose of defeating Kasparov.

of the "Behaviour-Based Artificial Intelligence" (BBAI), which builds on the idea that intelligence is not possible without a body. BBAI is tightly related to Behaviour-Based Robotics (BBR), which builds on the same idea. According to BBR, intelligent behaviour arises from a number of small modules, which react to stimuli in a semi-autonomous way. A popular approach is the subsumption architecture, proposed by Rodney Brooks, which is explained in more detail in Chapter 4.1.

How far can machines evolve?

Robotics and AI have known many successes. Numerous techniques, such as expert systems, neural networks, data mining and many others, have found large-scale commercial applications. However, up to this day, few of the initial promises of AI have been fulfilled. The debate is still open, on how intelligent will be the most intelligent artificial machines, or the ultimate robot. The optimists say that it's just a matter of time. With increasing computer power and memory, the problem is just to put everything together. The pessimists say that artificial machines will have to be radically different from the ones currently built if they are meant to reach levels of intelligence and competence comparable to that of human beings, because it's not possible to capture the true essence of intelligence using an algorithmic approach. The problem may be that research is following the wrong track: intelligence is the result of a process radically different from computing. Claude Shannon's discussion on "What Computers Should Be Doing" is often quoted in this aspect [16]. Shannon states:

> Efficient machines for such problems as pattern recognition, language translation, and so on, may require a different type of computer than any we have today. It is my feeling that this computer will be so organized that single components do not carry out simple, easily described functions. ... Can we design... a computer whose natural operation is in terms of patterns, concepts, and vague similarities, rather than sequential operations on ten-digit numbers?

If the pessimists are right, current research may be to a great extent on the wrong path. However, the history of Artificial Intelligence and Intelligent Robots has been made of ebbs and flows, much like the economic cycles. Occasionally, a new discovery spikes a wave of optimism, which soon fades and gives room to a "winter" of pessimism and only small advances. The current sentiment among researchers is that new approaches are still needed to renew AI, if the goal of building intelligent robots is to be achieved in the mid term. One of the most ambitious projects is the Mind Machine Project, launched in 2009. It is a 7-year and multi-million dollar project, involving some of the most renowned researchers in the field[6]. The Mind Machine Project intends to rethink AI from scratch, including fundamental

[6]For more information on the MIT Mind Machine Project see http://mmp.mit.edu (last checked 2010.03.10).

assumptions such as the nature of the mind and memory. That is one major step towards maintaining AI's original goal of building machines of human-level intelligence, while many authors agree that the current research is mostly split into a wide range of sub-areas. In those sub-areas, the focus is on building intelligent machines for specific tasks where a limited level of intelligence is needed, without caring about the big picture or cognitive insight into the nature of the problems[7]. The Mind Machine Project intends to shed new light onto the problem, using a holistic approach.

1.2 The problems

As explained in Section 1.1, the dream of creating intelligent machines is thousands of years old. However, the most significant developments in the field date back only a few tens of years, and it's still not clear if the goal can be achieved at all. The difficulties start with the very definition of the problem, which is still not clear. Besides, there are very high expectations, and those expectations are raised continuously. However, the knowledge and technologies available are still rudimentary, and may even be insufficient to achieve the final goal.

1.2.1 Poorly defined problem, or wrongly defined problem

The very definition of the goal is not clear at all. What is intelligence? Is it unique, or are there different levels and types of intelligence? What types of intelligent behaviours can be mimicked and are suitable to implement in artificial beings?

The goal

The first problem that is encountered when the goal is to build intelligent machines is that of understanding what intelligence is. Chapter 2 discusses this problem in detail. The definition of intelligence may just encompass the success of the intelligent agent, or also care about *how* the agent does it. In the first case it is only the performance that matters. The latter case is more complicated, and involves other concepts which are also poorly defined. The agent is expected to *understand* or *comprehend* its actions and its environment. However, there are no clear definitions of understanding and comprehending, as explained in Chapter 2.

Hence, the problem in its general formulation is poorly defined and comprises a goal which is not clear. Additionally, there is no standard way to measure the intelligence of a machine or a human being.

[7] For a more complete discussion, see for example http://www.itweb.co.za/index.php?option=com_content&view=article&id=30637 (last checked 2010.03.18).

Of course, there are the Intelligence Quotient (IQ) tests, but those are still subject to many criticisms and are not a standard measure of intelligence ready to be applied to artificial minds—and even more so when the goal is just to implement *intelligent* navigation.

Is the goal reachable?

Actually, the problem of reasoning and making *intelligent* decisions in general may not be solvable at all using currently known technology, as suggested in Section 1.1.2. For example, it is known that a sentence or word may cause the release of a chemical substance in the brain. That chemical substance may then cause a feeling or change into a mood. Then, after the sentence is forgotten, the feeling continues [50] until the chemical substance that causes it fades. This means that in the human brain the feeling is separated from thought. However, the feelings influence behaviours and decisions. That aspect of the human way of action is completely alien to the machines, which are built upon a completely different anatomy. Even the most modern machines are highly rational and algorithmic, while humans rely on a complex amalgam of emotions, memories and sensations to judge and act. Of course, even the release of a chemical substance and its fading can be simulated in software, although it's an awkward way to program decision-making processes.

Another important aspect is that the brain seems to have a natural ability to do mind reading. It is now well known that humans in general are very keen on mind reading, and that ability plays an important role in normal human communication and social relations. In a normal conversation, most of the information is exchanged non-verbally, and even unconsciously. Humans easily spot if someone is sad or happy, confident or unsure, nervous or relaxed. A simple look at a face, even at a picture of a face, is enough for the brain to unconsciously interpret a myriad of signals and spot a lot of emotions. Those non-verbal and unconscious analysis also play an important role when two people don't share a common language, or when someone is cheating or lying. The ability is natural in humans and to a great extent independent from race and culture. Paul Ekman and Tomkins [31] identified a set of about 300 facial expressions that are responsible for the ability to express about 30 different emotions. Those basic expressions are called Action Units (AU). An AU is, e.g., narrow lips or nose. Those 300 AU combined make around 10000 useful combinations, allowing humans to express emotions such as anger, happiness and disgust, among others. That ability plays a very important role in human communication—sometimes even more important than verbal communication. Computers are in a certain way like autists. There's a long way to go to program them to express non-verbal language and also to understand non-verbal signals. Computers will surely still lack the abilities to read someone's mind and to express non-verbal signals for quite a long time. The ability to automatically read emotion from a facial expression may not be impossible to program, but it seems to be very far away. The most

promising approach might be using sophisticated computer vision algorithms, able to distinguish an AU in an image of a face. Actuators strategically placed in an artificial skin might provide the necessary flexibility to imitate such facial expressions. Such abilities will make it possible to build a robot with very good conversation skills[8].

Paving a new way

An important consideration is that imitating humans may not be important at all to build intelligent machines. It is often said that men didn't learn to fly by imitating birds. The human solution to the problem of flying is completely different of that used by the birds and insects that fly. Yet, the human way is suitable for human needs, safe and efficient[9]. Hence, it may happen with robots the same that happened with the ability to fly. A completely different approach may be developed, resulting in the construction of machines which are different from humans, but equally smart and fit for many different purposes. The result may be a new, non biological, form of life. Current researchers tend to either imitate nature, being strongly tied to the natural models, or to neglect it completely, not taking advantage of what can be learnt from the natural world. But many different approaches may be valid to achieve the goal of building intelligent machines.

1.2.2 High expectations

There have always been high expectations, when the theme is "intelligent machines". Machines have to be smart, at least as much as humans, but cannot fail. And when a particular subgoal is reached, it looses its mysticism and is not considered a sign of intelligence any longer.

Better than humans

From the ancient Egyptians to the Dartmouth summit, and now to the modern think tank that is the Mind Machine Project, the dream has always been the same: to build intelligent machines, that can think and act like humans. However, the fact is that those intelligent machines have to be better than *average*

[8] See for example the Feelix project at http://www.feelix-growing.org/about (last checked 2010.03.30). Feelix has this goal.

[9] Despite the success of the flying devices that use different approaches, the construction of machines able to fly by flapping wings, like birds, is more energy efficient, and it is still an open area of research. The website www.ornithopter.org contains a thorough compilation of information about such devices, called "ornithopters". Todd Reichert was the first human to fly on an ornithopter, on August 2, 2010. That ornithopter had wings that spanned for 32 meters, and it was made of light materials, such as carbon fibre, foam and balsa wood. It weighed only 43 Kg and was moved by pedalling. Todd Reichert flew the aircraft for 19.3 seconds. (Source: http://www.ottawacitizen.com/dreams+flies/3581043/story.html, last checked 2010.09.26)

human beings. Everyone is, to a great extent, complacent with an occasional, apparently unexplained, human failure. The Roman philosopher Seneca wrote the famous sentence "errare humanum est" (to err is human), and it is still valid and popular nowadays. But an intelligent machine is not expected to make a mistake, under "normal" circumstances. And, depending on the kind of machine, a machine error may endanger people or property, perhaps more than a human failure in general. Therefore, researchers can hardly be satisfied with a machine with human-like abilities. Actually, the goal is to produce biologically inspired machines, but flawless in the range of problems they are designed to solve and the jobs they are made to accomplish.

A moving target

Another problem is that Artificial Intelligence is actually pursuing a kind of "moving target". The concept of intelligence is poorly defined. Hence, it is hard to classify any system as intelligent or unintelligent. A popular example is ELIZA, the first chatter bot, written in 1966 [106]. ELIZA is able to maintain a conversation with a human in a limited context, the most famous of which is the emulation of a Rogerian psychotherapist[10]. Many people mistook ELIZA for a real person, and attributed it superb intelligence. However, ELIZA is just the implementation of a few clever tricks. The key idea is to pick some selected keywords in the user's sentence and reply with a selected question based on that keywords. For example, to the sentence "I'm sad", a very probable ELIZA answer is "How long have you been sad?". That creates an illusion of intelligence, while in fact ELIZA does not even have a memory or context of the conversation. It works solely based on pattern matching. The question, then, is: is it intelligent? Despite the initial enthusiasm, nowadays it is widely accepted that intelligence has to be much more than those simple ELIZA-like tricks. This subject is discussed in more detail in Section 2.1.2.

1.2.3 Environmental difficulties

As presented in Section 1.2.1 (and discussed in more detail in Section 2.1.2), the concept of intelligence is poorly defined. Therefore, researchers are trying to find their way along unknown terrain, without a clear understanding of the localisation of the goal or the best way to find it. And there is also another problem: the environments that machines have to face can be quite hostile and complex, while the robots in general still possess only poor sensing abilities.Russell & Norvig classify the environments according to the following properties [89]:

[10]To test a working implementation of ELIZA see, for example, http://www.manifestation.com/neurotoys/eliza.php3 (last checked 2010.03.21).

- **Accessible/inaccessible**—Accessible is that environment which can be fully sensed by the robot. The robot must have access to all the information that is relevant to its choices. The robots usually have limited sensorial information, so they have to work with an only partially accessible environment.

- **Deterministic/nondeterministic**—In a deterministic environment, the next state is completely determined by the present state and the robot's actions. Real environments are, therefore, nondeterministic, from the point of view of the intelligent agent. Environments with inaccessible properties are always nondeterministic, for the agent cannot determine exactly the state of the variables it cannot sense.

- **Episodic/nonepisodic**—In episodic environments, the agent's experiences can be divided into sequences of perceptions and acts, which are self-contained episodes. What happens in an episode does not interfere with other episodes. Realistic scenarios are nonepisodic, so the robot actually has an advantage in planning and keeping a memory of the past, in order to have an acceptable performance, in a wide range of environments.

- **Static/dynamic**—static environments cannot change while the robot deliberates. Dynamic environments can. Again, robots usually have to face dynamic environments.

- **Discrete/continuous**—Discrete environments have only a limited number of percepts and actions. Those are, obviously, very rare. Robots in the real world (not simulated) have to deal with continuous variables.

According to these properties, the problem of robot navigation is the worst possible: the environment is not accessible, nondeterministic, nonepisodic, dynamic and continuous. Of course, in simulations or laboratory conditions, some or all of these properties can be controlled. In some particular applications, such as industrial robots in factory environments, the range of expected scenarios can also be limited. But, in general, the environments faced by real robots are the worst possible, and the robots must be robust enough to behave well under those conditions.

1.3 Motivation

Fifty years of research have produced unsatisfying results, in what concerns building intelligent machines. Modern robots are still designed for specific, limited tasks, and they cannot perform tasks other than those they are designed for. Every few years a new approach spurs a new wave of optimism, but researchers are still in search of the tipping point to start building intelligent robots. Therefore, it

is necessary to try new approaches, apply existing models into different areas, experiment and either propose new models or extend the existing ones in radically new different directions. This work has the intent of being an exploratory work, which applies an existing memory model, the Sparse Distributed Memory (SDM), to the problem of robot navigation using visual information.

The SDM is a kind of associative memory, proposed in the 1980s by Pentti Kanerva [52]. The SDM is suitable to work with sequences of high dimensional vectors, can learn and forget in a natural way, is highly immune to noise and can work with incomplete data. It is associative in the sense that an incomplete pattern can be used retrieve the complete datum. Therefore, it shows many characteristics which are also found in the human brain.

The SDM *per se* can mimic many functionalities of the human brain. But the brain is highly hierarchical, and the original model of the SDM is not hierarchical (however, a hierarchy of SDMs can be built. See, *e.g.*, [47]). For example, it is known that a rough model of the memory of a fear/dreadful situation is stored into the amygdala, a part of the brain which works below the level of awareness [50]. This explains the immediate reaction of some people to some situations which they fear, or their uncontrolled reactions. The rational explanations for the fear (the reason, for instance) are stored into the cortex. The rational part of the memory is somehow separated from the emotional behaviour. The former is dealt with higher in the hierarchy than the latter.

J. Hawkins proposes a hierarchical model [37] of the human memory, as described in Section 2.2.5. However, Hawkins' model is just a theory, and it is still not clear how to implement it. On the other hand, Kanerva presents thorough implementation details of the model. Many models of the SDM have actually been built and tested, as presented in Section 3.6. Therefore, the SDM is a natural candidate to use in the field of robotics. Actually, there is already some research on using SDMs in robotics. Some relevant approaches that were identified are described in Section 4.4.

Popular navigation approaches are reviewed in Chapter 4. In general, the SDM naturally confers on the robot characteristics of the human brain which are hard to achieve using other methods. It is a very powerful model, built on a strong mathematical basis, simple to implement and general enough to use to store different types of data.

1.4 Objectives

This work encompasses four major goals: implement a view-based navigation approach using a SDM; study of the encoding problem; test of a path planning algorithm using the SDM; and study of the characteristics of a robot equipped with the memory.

View-based navigation

The main goal of this work is to implement intelligent robot navigation based on visual memories and a SDM. In this approach, a robot learns paths during a supervised learning stage. Using the learnt data, the robot is later able to follow those paths autonomously.

During learning the robot captures images and odometric data using a front camera and other sensors. Those images are processed (equalised and/or contrast stretched) to improve the quality. Then, the relevant images are stored into the SDM, along with relevant navigation information. Images which do not provide information useful for navigation (for example, because they are too similar to other images already learnt), are discarded. In the memory, the images are organised into sequences which can later be retrieved from beginning to end, from end to beginning, or from inbetween.

During its autonomous run mode, the robot uses the grabbed views to try to localise itself in a path. Then it follows the path by performing the same actions it performed when it was learning and storing the learnt images. Small drifts are corrected using an image matching technique.

The tests were performed using a small mobile robot, as described in Section 5.1.1, which was used to learn and follow paths in scenarios that were reconstitutions of urban and countryside environments.

Path planning

By following only learnt sequences, the capabilities of the robot are very limited. To implement a more advanced behaviour, an algorithm was implemented to build a topological representation of the environment. The topological representation is a graph of the learnt paths. Intersections of paths that have been detected are represented as nodes in the graph. The intersections and common segments can be detected by the occurrence of many consecutive images matched with consecutive images of another sequence.

Once a topological representation of the environment is built, it is possible to implement goal-directed navigation and path planning using search algorithms such as A*. This way, the robot can follow the shortest route that leads to the goal, choosing the right path at intersection points in function of the goal point it is heading to. The algorithm and results are described in Chapter 9.

Encoding problem

Once the basic navigation algorithm was implemented, it was noticed that the performance of the system depends on the architecture of the SDM, and also on the method of encoding the information that was being stored and used. The problem is that the original SDM model is planned to work in assorted high

dimensional binary spaces. However, sensorial data is usually encoded using the natural binary code, which is a way of grouping the bits into higher dimensional spaces, taking advantage of the positioning of the bits. Part of this work consists of a comparative study of the impact of the different methods of encoding the information on the performance of the system, as described in Chapter 7. The problem was studied in the domain of robot navigation, but also in the dissimilar domain of robot manipulation, as described in Chapter A.

The limits of the memory

Another important contribution of this work is the study of the limits of the memory, as described in Chapter 8. It consists of several experiments, in which the robot follows learnt paths under challenging situations, such as dim illumination, scenario changes or memory overflow. To the best of the author's knowledge this is the first time those tests are performed and the results are reported, despite the fact that the SDM is a model more than 20 years old. The conclusions show that the SDM is able to withstand huge illumination changes, overflows of about 30% and occlusions of more than 12.5% of the image.

1.5 Structure of this thesis

This chapter intends to give the reader an idea of the historical background behind intelligent machines, and also introduces the work developed. Chapter 2 compares human and artificial intelligence. It briefly reviews the most up to date definitions of intelligence, and briefly describes two expected sources of intelligent behaviour: the human brain, which is probably the source of human intelligence, and the Turing machine, which is able to run any computable algorithm, and so is the natural tool to use to try to build intelligent machines. This chapter also briefly reviews the State Of the Art (SOA) in terms of associative memory models.

Chapter 3 is devoted to the SDM model, as proposed by Pentti Kanerva and later improved by many different authors. It summarises the mathematical principles behind the SDM and Kanerva's original proposal to build the sparse distributed memory. Then it reviews the SOA in terms of improvements and applications of the original Kanerva's model and lists important advantages and disadvantages of the SDM.

Chapter 4 is a review of the SOA in terms of robot navigation approaches. It describes the more important paradigms that have been used in recent years to map and navigate robots, focusing specially approaches that are based on the use of visual memories. It also describes previous applications which

have used sparse distributed memories. Section 4.4.3 contains an important comparison of the SDM, when likened to similar techniques, such as Hidden Markov Models and Support Vector Machines.

Chapter 5 describes the experimental platform that was used, including the robot and laptop used to perform the tests, as well as the testbed where almost all the tests were performed. It also describes the architecture of the software that was developed, including the image processing steps and collision avoidance algorithms.

Chapter 6 describes the architecture of the implemented SDM. That includes the use of a dynamic radius, adjusted in function of the amount of noise that is found in the images. It also describes the different operation modes used, as well as the performance assessment measures that were defined to compare the results obtained using the different operation modes. It ends with the conclusion that the performance of the original SDM model is impaired because it is based on the use of a Hamming distance, which does not consider the position of the bits, while the sensorial information is based on the use of the natural binary code, in which the value of each 1 (one) depends on its position (this problem is called throughout the thesis as the "encoding problem").

Chapter 7 is dedicated to the study of the encoding problem, and is one major contribution of this thesis. The encoding problem arises from using different criteria to encode the information that comes from the sensors and to process it inside the SDM. Therefore, it can be solved either by encoding the sensorial information in a different way, or by changing the structure of the SDM to work differently. Both approaches were tried and the results are compared in this chapter.

Chapter 8 is a study of the performance of the SDM under typical robot navigation problems and is another major contribution of this work. Problems such as that of the kidnapped robot, illumination changes and scenario changes are simulated in a controlled environment, to assess the tolerance of the system facing those situations. The performance of the memory was also measured facing problematic situations which are predicted by the theory, such as memory overflow and forgetting over time. The tests and results are described in this chapter.

Chapter 9 describes a method that was implemented to build a topological map from the sequences of images stored into the SDM. The SDM is used both to store images and as a pattern matching tool to detect points where paths come together or split apart.

Chapter 10 summarises all the work done, the main contributions and advantages and disadvantages of this approach. It also opens perspectives of possible future work to further research in these topics.

Finally, Appendix A describes a comparison of the performance of the SDM in two different domains: the domain of robot navigation using view sequences, and that of robot manipulation using position sensors. The former domain is characterised by long vectors and large amounts of noise that are found

in the images. The latter domain is characterised by relatively short vectors and very small amounts of noise. The appendix describes the experiments and compares the results obtained.

1.6 Summary

The dream of intelligent machines is thousands of years old: it dates back to the ancient Egypt and to the Greek mythology, at least. The knowledge to build those intelligent machines comes from old and recent scientific discoveries. The 20^{th} century has been specially prolific. The problem is naturally very complex, because the goal is to build machines which have to make successful decisions in very challenging environments. But it is even more difficult due to the fact that the goal is not clear, because the main characteristic that is sought to those machines—intelligence—is still poorly defined.

This work follows an approach which consists of programming a robot to perform intelligent navigation based on visual memories. The robot stores memories of places it visits and it is later able to visit them again by following the same paths, or inferring new routes by detecting connection points between paths. It discards memories which are, with high probability, not useful, and detects connection points and common segments between paths. Those behaviours are achieved through the use of a Sparse Distributed Memory—a sophisticated associative memory proposed back in the 1980s, which confers on the robot characteristics such as high immunity to noisy data, the ability to work with incomplete data and natural learning and forgetting. The limits of the system under changing and unfavourable conditions are also studied and reported.

Chapter 2

Intelligence and the Brain

Yesterday, upon the stair,
I met a man who wasn't there
He wasn't there again today
I wish, I wish he'd go away...
(William Hughes Mearns, in *Antigonish*)

Contents

This chapter reviews up to date concepts about intelligence, how it most probably relies heavily on the use of a sophisticated memory, and how it can be mimicked using artificial systems. It tries to make the bridge between human and artificial intelligence.

2.1 Human and Artificial Intelligence

Perhaps the biggest problem researchers face when building *intelligent machines* is that of understanding what intelligence is. The goal of Artificial Intelligence is *clearly* to build *intelligent* machines, and the standard for comparison is human intelligence. This means that the roadmap includes: 1) Understanding what intelligence is; 2) Understanding how it can be tested or quantified; and 3) Finding ways to simulate it in artificial agents (in this case, robots). This section briefly reviews some definitions of intelligence, proposals to measure it and the first tool behind the simulation of intelligent thinking.

2.1.1 The Turing machine

Intelligent agents must mimic *intelligent behaviour*. It is known that humans achieve intelligent behaviour because they possess sophisticated brains that are capable of storing valuable information and use them in a way to make the right decisions. Robots are expected to achieve the same or better results by performing the necessary computations, possibly using modern computers, which can be regarded as implementations of the Turing Machine (TM), proposed by Alan Turing [100].

The Turing machine is an abstraction of the concept of computation. It has the capability to simulate any form of computation, thus being able to solve any problem that is computable. The basic model is not an architecture of a machine to be implemented. It is a thought experiment, which illustrates how a machine can compute any algorithm. (Nonetheless, it can be implemented, and there are working implementations of the TM that prove the feasibility of the concept.[1])

Figure 2.1 shows a simplified model of a Turing Machine. The TM has the following parts: a long, possibly infinite, tape; a head, that can read and write symbols in the tape; a finite table that contains rules or instructions that the machine must execute; and a state register, that stores the state of the machine. The function of the machine is to process the tape. The tape is divided into squares, and each square has a printed symbol. The machine head can be moved to any point of the tape and read or write the symbols one at a time. At any moment, there is one symbol in the machine. This symbol and the previous state of the machine determine its behaviour. Operation of the machine is defined by

[1]For instance, http://www.turing.org.uk/turing/scrapbook/tmjava.html (last checked 2009.11.21).

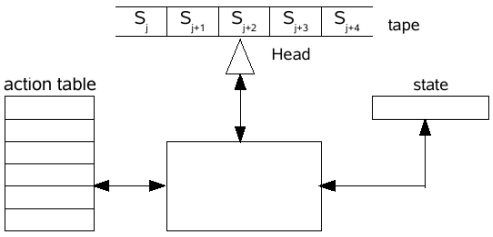


Figure 2.1: Simplified model of a Turing Machine.

a finite set of rules, stored into the table. The rules are such as "in state A, if the symbol read is X, then write symbol Y".

Since the machine tape can be infinite, it gives the machine unbounded storage space, which can be used as a working memory. Given also enough time, the machine has the potential to execute any algorithm or solve any problem that is computable.

A Turing Machine that is also able to simulate any other Turing Machine on arbitrary input is called a Universal Turing Machine (UTM). The UTM can achieve this behaviour by reading from its tape the description of the machine it simulates, thus mimicking its behaviour. This is basically a model of a modern computer program, where the computer running the program works as a Universal Turing Machine, by reading from its disk (*tape*) the description of the behaviour it must simulate. The modern computers are, in this sense, Turing Machines. If the intelligent behaviour is the result of a computable algorithm, it should, therefore, be possible to write a computer program to mimic it.

2.1.2 Definitions of intelligence

Just like many other concepts, until very recently the concept of intelligence was neither clear nor questioned or challenged. Definitions in the dictionaries are similar to this one: "the ability to comprehend; to understand and profit from experience"[2]. Although understandable for a vast majority of readers, this definition is clearly insufficient for the researcher who wants to build intelligent machines. It is based on other concepts which are defined only informally. Namely, it relies on the definitions of "comprehending" and "understanding", which are equally poorly defined.

Until very recently, the most solid ground on this subject was a series of sparse and informal writings from psychologists and researchers from related areas. Although there seems to be a fairly large common ground, among all the definitions the boundaries of the concept are still very cloudy and roughly defined.

Moreover, it is in general accepted that there are several different "intelligences", responsible for several different abilities, such as linguistic, musical, logical-mathematical, spacial and other abilities.

[2]Definition taken from Wordnet, version 3.0. Online at http://wordnet.princeton.edu (last checked 2009.11.15).

However, in many cases individuals' performance levels in all these different fields are strongly correlated. Spearman [96] calls this positive correlation the g-factor. The g-factor must, therefore, be a general measure of intelligence. The other intelligences are mostly specialisations of the general one, in function of the experience of the individual.

Gottfredson definition

In 1994 Linda S. Gottfredson published a fairly complete review of the mainstream opinion in the field [32][3]. Gottfredson wrote a summary of her personal definition of intelligence, and submitted it to half a dozen "leaders in the field" for review. The document was improved and then submitted to 131 experts in the field, who were then invited to endorse it and comment on it. A total of 100 experts responded: 52 endorsed the document; 48 did not endorse it for various reasons. Not all the reasons were technical: some were personal or political, for example. Of those who did not endorse the definition, only 7 stated that it did not represent the mainstream opinion about intelligence. Therefore, it is reasonable to assume that a representative number of experts agree with this very definition of intelligence:

> Intelligence is a very general mental capability that, among other things, involves the ability to reason, plan, solve problems, think abstractly, comprehend complex ideas, learn quickly and learn from experience. It is not merely book learning, a narrow academic skill, or test-taking smarts. Rather, it reflects a broader and deeper capability for comprehending our surroundings—"catching on,", "making sense" of things, or "figuring out" what to do.

Gottfredson clearly emphasises some key aspects: **problem solving**, **learning** and **understanding**. Yet, there is no concern or explanation on how these tasks can be accomplished. Also, there is little concern with the performance of the intelligent agent. At most, that is part of the "problem solving" assessment: it is implicit that the agent must solve problems successfully, but unclear if it can fail, or what amount of failure is tolerable. On the other hand, this definition strongly depends on the ability to "understand". However, there is no definition of what is "understanding". In summary, this definition of intelligence has to be taken into account, because it is presented as representative of the mainstream opinion of the experts in the field. Nonetheless, it is of little use in practical terms, since it still leaves the boundaries of the concept undefined and still leaves two important questions unanswered: how to assess the performance of an intelligent agent; and how to simulate intelligent behaviour.

[3]The reference [32] states the article was first published in the *Wall Street Journal*, December 13, 1994.

Legg's formal definition

S. Legg and M. Hutter [59] also present a thorough compilation of interesting definitions, both from psychologists and AI researchers. And they end up with a shorter and very pragmatic definition:

> Intelligence measures an agent's ability to achieve goals in a wide range of environments.

This very definition has the merit of being much shorter and clearer from the point of view of an engineer, as it is very pragmatic. Legg starts from this informal definition towards a more formal one, and proposes what is probably one of the first formal definitions of intelligence.

According to Legg, an intelligent agent is the one who is able to perform *actions* that change the surrounding *environment* in which he or it exists, assess the *rewards* he or it receives and thus *learn* how to behave and profit from his or its actions. It must incorporate, therefore, some kind of reinforcement learning.

In a formal sense, the following variables and concepts can be defined:

- o observation of the environment
- a agent's action
- r reward received from the environment
- π an agent
- μ an environment
- E the space of all environments

The agent π, at time k, is defined as a probability measure of its current action, given its complete history: $\pi(a_k|o_1r_1...o_{k-1}r_{k-1}), \forall \quad k \in \mathbf{N}$.

The environment μ is defined as a probability function of its history: $\mu(o_kr_k|o_1r_1a_1....o_{k-1})$

Legg also imposes the condition that the total reward is bounded to 1, to make the sum of all the rewards received in all the environments finite:

$$V_\mu^\pi := E\sum_{i=1}^{\infty} r_i \leq 1 \tag{2.1}$$

One important point to consider when evaluating the performance of the agent is also the complexity of the environment μ. On this point, Legg considers the Kolmogorov complexity, or the length of the shortest program that computes μ:

$$K(\mu) = \min_p\{l(p) : \mathcal{U}(p) = \mu\} \tag{2.2}$$

where $\mathcal{U}$ is the Universal Turing Machine.

Additionally, each environment, in this case, is described by a string of binary values. As each binary value has two possible states, it will reduce the probability of the environment by 1/2. Therefore, according to Legg, the probability of each environment must be well described by the *algorithmic probability distribution* over the space of environments: $2^{-K(\mu)}$.

From these assumptions and definitions, Legg proposes the following measure for the universal intelligence of an agent π:

$$\Upsilon(\pi) = \sum_{\mu \in E} 2^{-K(\mu)} V_{\mu}^{\pi} \tag{2.3}$$

The intelligence Υ of an agent π is, therefore, a measure of the sum of the rewards it is able to receive in all the environments—a formal definition that is not against the informal ones described before. Unfortunately, the interest of this definition up to this point is mostly from a theoretical point of view, because Equation 2.3 is not computable. It is, nonetheless, an interesting approach to formalise intelligence. And it is still interesting from a practical point of view, as a general demonstration that intelligent agents need to be versatile to perform well in a wide range of environments, as well as profit from past experience by learning from previous actions or sensor readings.

Discussion

So far, two important definitions of universal intelligence have been presented:

1. Gottfredson's definition of intelligence as an ability to learn, understand and solve problems;
2. Legg's formal definition of intelligence as a measure of success in a set of environments.

These definitions are not incompatible, but if the first is to be accepted as the standard, an additional definition of what is understanding is still lacking. Moreover, it is still not clear what is understanding or if it is important for an intelligent agent. John Searle proposed an interesting thought experiment which shows that performance is different from understanding [93]. Searle's proposal is popularly known as the "Chinese room experiment," and it is as follows. Imagine Searle in a room where he is asked some questions in Chinese. Searle knows nothing of Chinese, but he has a book with all the possible questions and the correct answers, all in Chinese. For every question, Searle searches the book and sends the correct answer. Therefore, Searle gives the correct answer 100% of the times, without even knowing a single word of what he is saying. This is not much different from today's intelligent agents, which can perform quite well in the tasks they are programmed to, without having a clear understanding of the *words* or *tasks* they actually are using and performing. According to Legg, these agents may

be intelligent up to some degree, depending on the complexity and quantity of environments they can operate on. According to Gottfredson, they cannot, for they are not supposed to be exhibiting the abilities to "comprehend their surroundings" or "make sense of things".

Anyway, the definitions seem to agree that successfully solving problems in a variety of environments is a key ability to intelligence and intelligent behaviour. Dimiter Dobrev [22] goes even further, proposing that an agent that correctly solves 70% of the problems (i.e., makes the correct decision 7 out of every 10 times), should be considered an intelligent agent. The author does not make it clear why he proposes 70%. This number is just presented as a possibly good compromise between the rates of success and failure. It makes some sense, in the light of the scales used to assess the performance or "intelligence" of humans, for example when grading students' tests. However, there is no clear evidence of the scientific or statistical validity of this number. Moreover, Dobrev makes no distinction in the kind of decisions assessed. That may be an important weakness of the proposal, because an intelligent agent may have to decide about a wide range of subjects. For instance, some of the decisions may be about its own safety, or the safety of humans or other artificial agents. In that case, a failure rate of 30% may be unacceptable, because the agent might be damaging itself or others. There are decisions in which a single wrong choice is not acceptable. It is clear that humans make wrong decisions from time to time, and that is accepted as *natural*. However, humans and animals hardly make wrong decisions in life-threatening situations. A biologically-inspired agent should, at least, achieve a very high performance in those life-threatening situations, and a 70% success rate is apparently too low.

2.2 On the brain

It is accepted that the brain is the source of "intelligence" and "intelligent behaviour". Therefore, it is of interest for researchers willing to build intelligent robots to look at how the brain is organised and how it works. Despite fairly encouraging discoveries in the field, little is already known about the brain structure and method of operation that is of interest from a practical point of view, for building intelligent machines.

2.2.1 The neuron

The basic building block of the brain is the neuron. A neuron is a cell as represented in Figure 2.2. The main body of the neuron is called the soma. The soma has several dendrites, which are the cell's receptors. Those dendrites receive signals from other neurons. The soma also has a kind of long tail, that is called the axon. The axon terminates in a tree of axon terminals, that can connect to other cells. The axon terminals of one neuron can make connections with the dendrites of another neuron. Those

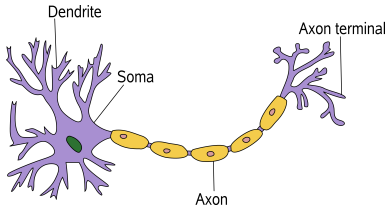


Figure 2.2: Basic structure of a neuron.[4]

connections between the axon of one neuron and the dendrites of another are called synapses. That is how the brain learns—establishing new synapses between the neurons.

The signals received by the dendrites may be inhibitory or excitatory. If the neuron receives enough excitatory signals and the inhibitory signals are below a given threshold, the neuron enters into a process of electrochemical reactions and fires a signal through its axon and synapses. That signal will possibly cause other cells to fire too. Other neurons will be stimulated to fire, and/or actions will be performed, such as muscle cells' contraction.

Although the neurons are the basic building blocks of the brain, they are also present in other parts of the body. They are also the building block of the spinal cord, and they are present throughout all the nervous system as sensorial receptors and information transmitters.

2.2.2 Brain structure

The human brain is composed of grey matter and white matter. The former consists mostly of neurons, while the connections between them, the axons, make up the white matter inside.

The outer layer of the brain is grey matter, and it is called the cerebral cortex. Inside that layer there is a lot of white matter, some gray matter and some spaces filled with cerebrospinal fluid.

The brain is usually regarded as the source of intelligent behaviour[5]. Non-human animals have brains which are relatively small and specialised in functions which are essential for their own survival, such as controlling the breathing impulse, feeding and reproduction. The human brain is responsible for the very same behaviours, thus being comparable at this level with other animals, namely primates, which are very close to humans in the evolutionary path. The difference arises at another level, where intelligence may appear. Mammals' brains also evolved an additional structure, called the neocortex.

[4] Based on http://en.wikipedia.org/wiki/Neuron (last checked 2009.11.21.).

[5] Although it may seem indisputable that the brain is the source of intelligent behaviour, it is still an open question "how much" responsibility for that intelligent behaviour can be credited to the brain. As explained in Section 2.2.1, there are neurons all over the human body. This means that the sources of "intelligent behaviours" may be more distributed than it was previously thought. On the other hand, it is still disputed what are those "intelligent behaviours", as explained in Section 2.1.2.

That neocortex is mostly a very large structure of grey cells (neurons), on top of the cerebral himespheres. In humans it sums up to about 76% of the total brain [17]. It is a smooth surface in small mammals, but a very corrugated one in humans. The grooves and wrinkles increase the overall area of the neocortex. It contains about 23 billion neurons in males and 19 billion in females, distributed over a 2–4 mm thick layer. That layer can be divided in six sublayers, labelled from I (the outermost) to VI (the innermost).

Along with the archicortex and the palecortex, the neocortex is part of the overall system called the cerebral cortex, mentioned above. The neocortex is the most recent part of the mammalian brain, in the sense that it appeared later in the evolutionary path. It is usually considered as responsible for playing the most significant role in generating intelligent behaviour. It plays a central role in functions such as sensory perception, memory, generation of motor commands, spatial reasoning, conscious thought and language. It is also able to interfere with older structures of the brain, eventually superimposing to them and inhibiting their natural functions. That mechanism allows humans to control instinctive behaviours, such as stopping breathing until one eventually faints. When one faints the "old brain" assumes control of the breathing instinct again.

Despite the old interest on how the brain works, it was only recently that researchers were able to get some insight into the subject. The problem is hard for a number of different reasons. Dissecting a body and looking at a dead brain is not of much use—it is just an almost random piece of gray and white cells. It is like trying to understand how computers work looking at failing hardware pieces which can no longer work. Looking at a human brain, it is possible to visually identify different regions of the brain, such as the ones represented in Figure 2.3. But the most important pieces of information arose when modern brain imaging techniques were developed. Techniques such as magnetic resonance lead researchers to identify regions which are activated when a person performs actions such as speaking, deciphering an image, moving, etc. Brain scanning is actually a very active area of research nowadays. Experiments are being carried on, in order to make accurate maps of the brain, that show where certain brain functions typically reside. The study of the brain functions *one by one* is a task that is very complex, very time consumming and also very expensive because of the required equipment and skills.

However, mind mapping as described above is still not enough for engineers trying to build intelligent machines. Figuring out where task x or y is usually performed still does not provide any clue on how those tasks are actually performed, for the brain building blocks are essentially the same neurons and the brain structure shows only small differences from region to region.

Back in the 1970s, it was already known that there are many brain functions, many brain regions and many connections between them. Additionally, although there are noticeable physical differences between brain regions, those differences are only but small. Based on those observations, V. Mountcastle [74] proposed that *all* the brain might be performing basically the same algorithm, the result being

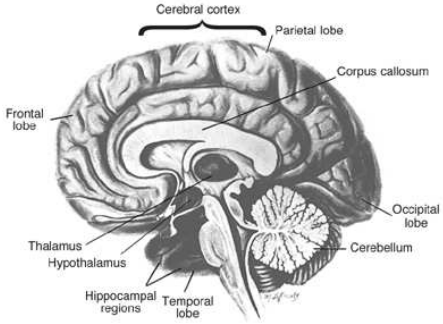


Figure 2.3: Most important regions of the brain [80].

different only depending on the inputs. Even the physical differences that are noticed could be a result of the brain wiring connections. Although that may seem an unrealistic proposal at first sight, many scientists currently endorse Mountcastle's theory, as it can not be proven wrong and explains phenomena which would be harder to explain assuming that the brain is an enormous conglomerate of specialised neurons. One important observation is probably the fact that the brain is not static. It adapts to its environment and changes when necessary. For example, people who are born deaf process visual information in areas where other people usually perform auditory functions. Another example are people with brain injuries of particular brain regions: in many cases other brain areas undertake the information processing of the damaged areas. Even more convincing, neuroscientists have surgically rewired the brains of newborn ferrets, so that their eyes could send signals to the areas of cortex that should process auditory information. In result, the ferrets developed visual pathways in the auditory portions of their brains [37]. The evidence, in summary, shows that most probably the intelligent behaviour is not a result of complex processing abilities, but the result of a possibly simple *universal* algorithm.

2.2.3 The brain as a memory system

The underlying reason for human intelligence may be the fact that humans have a large neocortex, able to process large quantities of information up to a high level of abstraction. How those huge amounts of information are processed is still a *mystery*. The *mystery* is yet more intriguing considering that the brain performs incredibly *complicated* tasks at an incredibly fast speed. It is known that neurons take about 5 ms to fire and reset. This means that the brain operates at about 200 Hz—a frequency fairly below any average modern personal computer.

Moreover, it is known that the brain takes about 1/10th of a second to perform tasks such as language understanding or visual recognition. Considering that neurons take about 1/1000 of a second to send a signal, this means that, on average, those tasks cannot take more than 100 serial steps, because no

more than 100 neurons had time to fire in the time gap. However, it is not possible to write a computer program to process any of those tasks with just 100 instructions. Besides, the neurons perform tasks much simpler than the modern computer instructions. This problem is often addressed as the "one hundred steps rule" in the literature [35].

One possible explanation for the awesome behaviour of the human brain is that it performs many tasks in parallel. Many neurons working at the same time would contribute to the overall final result. The brain can be, therefore, a massive parallel super-computer. That explanation, though, is not satisfactory for all the problems the brain seems able to solve in fractions of seconds. The theory of the brain working as a massive parallel super-computer, though attractive, is not likely to explain all the phenomena. The conclusion arises from the observation that many actions the human brain seems to perform in just fractions of a second cannot be done in parallel, for some steps of the overall process depend on the result of previous steps. An example from Jeff Hawkins [37] is the apparently simple task of catching a ball moving at some speed. The brain needs to process visual information to identify the ball, its speed and direction, and compute the motor information needed to move all the muscles which have to be stimulated in order to catch the ball. And more intriguing, the brain has to be repeating all those steps several times in a short time interval for better accuracy, while at the same time controlling basic impulses such as breathing and keeping a stable stance and equilibrium. In the case of football players, good goalkeepers sometimes even predict the direction of the ball before it is actually kicked in penalties. To build a robot to perform that apparently simple tasks, using current technology, is a nightmare, no matter how many processors can be used. The most difficult part of the problem is that motor information cannot be processed while sensory information is not available. No matter how many processors are used, there is always a number of steps which cannot be performed in parallel. A simple analogy, also from J. Hawkins, is that if one wants to carry one hundred stone blocks across a desert and it takes a million steps to cross the desert, one may hire one hundred workers to only cross the desert once, but it will, nonetheless, take one million steps to get the job done.

Based on the one-hundred step rule, J. Hawkins proposes that the human brain must not be a massive parallel computer, but mostly a memory system. The brain does not have time to compute solutions, so it has to be retrieving them based on analogies with learnt experiences from past situations. That also explains why practice and experience lead humans closer to perfection—their *database* of cases, problems and solutions is enriched, allowing them to retrieve better solutions to problems similar to the ones they have already captured.

2.2.4 The memory-prediction framework

According to the previously explained principle that the brain is mostly a memory system, J. Hawkins [37] also proposes a new framework to explain the human brain's behaviour. According to that framework, the neocortex is essentially a tool to store sequences of patterns. Those patterns can be representations of sensorial readings, or artificial representations created by the brain itself. The brain has the ability to recall those patterns auto-associatively to make predictions of what may happen or may have happened. This way, it is able to follow any previously stored sequence, from any given point onwards, and up to some degree, also backwards. This happens when a person relives some event, or sequence of events. The brain is also able to create novel sequences anew. That ability is used to dream, to foresee or to carry on creative acts. According to J. Hawkins, the human ability to make those predictions is the basis of what is called intelligence. A more acute ability to make accurate predictions with less data corresponds to a higher degree of intelligence and results on a more intelligent behaviour.

The brain is protected from the outside world, inside the skull. The skull is a large and hard mass that acts as a protective shield, isolating the neural mass from external actions. Besides, the brain has no sensory elements of itself. All the neurons are connected just to other neurons. Hence, the brain is not *aware* of what is happening inside the skull. The only information it receives comes from the sensory inputs and feedback from the brain itself. When there is no information from the outside world, no sensory input, the brain is still able to process information generated by itself. Patterns of memories previously stored are fed back to the neocortex through the many feedback loops inside the brain, eventually firing other memories which are associated. That is the mechanism that makes it possible to remember past events and visualise some sequences of memories (or parts of them) from the beginning to the end.

The brain is also permanently receiving information from the outside, through sensorial inputs corresponding to the five human senses. That information is firstly processed in the lower layers of the cerebral cortex. There occurs a form of *filtering*, that filters the relevant information out of the other one. A large amount of the information received is not considered relevant. Ambient sounds or smells, familiar landscapes, etc., are often simply discarded. They usually do not follow up to higher layers of the cortex, thus not capturing the attention of the person experiencing them. However, the senses may also feed information considered relevant enough to capture one's attention. If it is something new, such as an unseen landscape, or a very loud music or sound that are not expected, those patterns are not recognised in the lower layers of the cortex. They have to be passed on to upper layers, for further processing and analysis. Eventually, something new and relevant may be found out in those patterns. The experience may carry new and useful information. If that is the case, it requires some neurons to be rewired and new representations are retained to be stored into the brain. In other words, the person

has stored a representation of the new experience, which means to have learnt something new. This idea is supported by recent findings by Axmacher et al. [6]. Axmacher reports that unexpected stimuli trigger brain activity that lead to subsequent processing in order to promote memory encoding.

When the brain stores new patterns to learn something, those patterns that are stored are not exact representations of the sensorial inputs. The patterns are stored in a way that Hawkins calls *invariant forms*. Those invariant forms are more suitable to represent the objects or entities they refer to, and easier to process. An invariant form is a kind of concept. For instance, the concept of a "perfect circle" is something which can hardly exist in the real world. No one has probably ever seen anything like a "perfect circle". Nevertheless, possibly all the human beings that do not suffer any kind of severe impairment are able to think of and visualise a "perfect circle". Then, when they notice even a rough representation of a circle in an image, picture or landscape, that rough representation triggers the invariant representation of the "perfect circle" that is stored into the brain. The same happens for every entity humans deal with. The representation of someone's face in one person's brain allows the person that holds the memory, to recognise that familiar person, even in very different conditions. The illumination conditions, the angle, the scenario, may change a lot and be quite different from the conditions where the memory was stored, but the person holding the memory can successfully make the connection in very harsh circumstances.

According to the Memory-Prediction Framework (MPF), the brain is essentially a large memory system, constantly predicting what will happen next and acting according to those predictions. In a known environment, one person's actions are processed almost without that person being aware of them. For example, many people drive every day to work based uniquely on old memories of what happened the days before. When a sequence of actions becomes routine, it can be performed without one paying attention to what is being done. The sensorial inputs are *automatically* triggering actions at the lowest levels of the brain hierarchy, without the need of capturing one's attention. When something unusual happens, such as a car accident, or someone crossing the street in an unexpected place, that violates the brain's predictions. When a prediction is violated, the input that caused the violation captures the attention of the bearer. One stops acting based on the previous *sequences of cases* and starts searching the memory for the closest similar situation, in order to know how to behave in the new situation. When people are experiencing new environments, they are doing that all the time. Their senses keep feeding their brains patterns which keep them busy. Those sequences of patterns are then stored as new sequences, which may be retrieved later. That is how humans learn, so that they can later predict with better accuracy in similar situations.

2.2.5 Invariant forms and hierarchical temporal memory

One of the most difficult tasks performed by the brain, according to the memory-prediction framework explained in section 2.2.4, is that of creating invariant forms that can be used by computer programs as humans use concepts and mental models. Modern computer models in general are quite inefficient dealing with variations. Even the smallest variations can be troublesome, if not carefully programmed in advance. Actually, a modern computer or robot is most of the times little more than a calculator, solving thousands of mathematical equations which have been painfully programmed one by one. Computers have no understanding of what they are doing and can hardly improve their performance over time beyond the expected frontiers they have been programmed to. All the models of the world which are the basis of their hard work had to be programmed in advance. Computers are unable to construct their own models of the world, or to create invariant forms as human brains seem to do in a very natural and easy way. At most, they can extract features and create models of the environment with expected features.

Hawkins has also extensively elaborated on the problem of creating invariant representations, the basis of his theory of the memory-prediction framework. To create those invariant representations, D. George and Hawkins [30] propose the model of a Hierarchical Temporal Memory (HTM). That HTM, according to the theory, once exposed to some patterns of sensorial input, must be able to automatically create its own model of the world, much like what is believed the human brain does. That internal model must guarantee the memory the ability to recognise familiar objects and predict the future in a very human-like way.

An HTM is composed of several nodes, organised in a hierarchy, as shown in Figure 2.4. Each node can be linked to several nodes below and one single node above, in a tree structure with one or more nodes on the top layer. The nodes at the lowest level receive input from the outer world. The nodes at other levels receive input only from the level below—level two receives input from level one, level three from level two and so on. The flow of information is controlled by conditioned probabilities: an event triggers a given node if it is the most likely node to have learnt information related to that event.

When the system is shown an image, such as the one depicted in Figure 2.5, it is split in several parts. Each low level node receives only one part of the image (say, a 4×4 pixel square). Those low level nodes then pass the patterns they see up to the next levels, where they are concatenated to form the shapes. The top level nodes compare the shapes against a library of objects, in order to select the best match. Figure 2.5 shows an example of the various stages during the recognition of an helicopter. The raw input is divided in smaller windows which are fed to the bottom nodes. Those nodes are eventually able to recognise small parts of the image and feed the upper ones. In the top layer the image can be

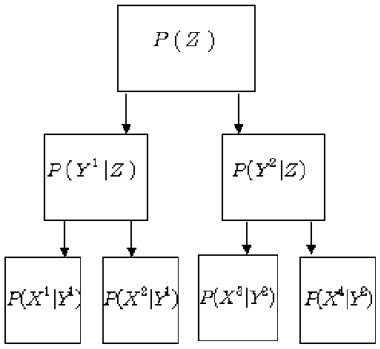


Figure 2.4: Example of probabilistic hierarchy in an HTM. The bottom nodes receive raw sensor data, the top nodes are related to more abstract concepts. Source: [30].

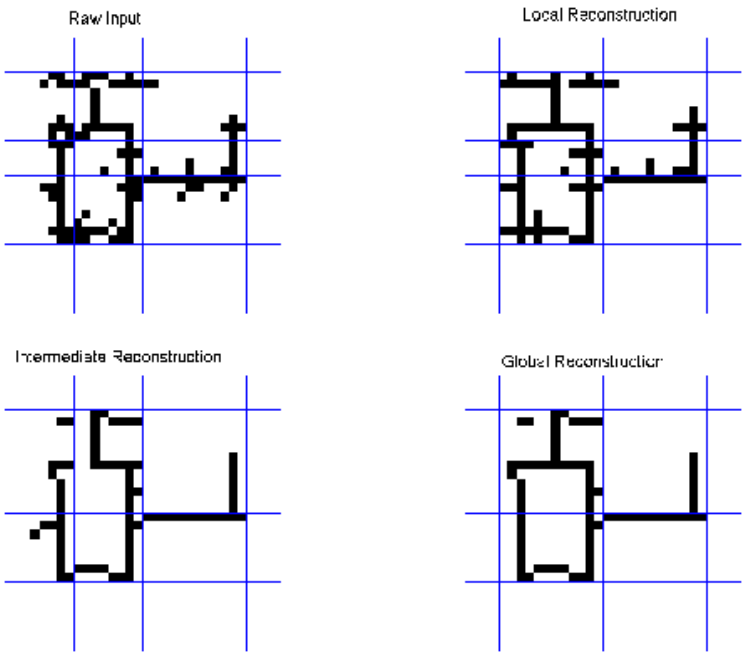


Figure 2.5: Example of various stages of recognition of an helicopter in an HTM, from the raw sensorial input to the activation of the abstract concept. Source: [30].

recognised as an helicopter. Recognising the image as an helicopter means to establish a link to the invariant concept of "helicopter". Hence, the noise, and possibly some details, have been removed from the image, to make it match the top concept. The information from the above nodes is then fed back to the lower levels, causing an adjustment of the probabilities, so that all the nodes can predict better what they are supposed to see in future occurrences of the same pattern, thus refining the prediction ability of the model.

Every node in the structure is able to learn to recognise a given pattern. When the overall system has learnt to recognise a given pattern, the tree can be seen as a Bayesian network of probabilities, as shown in Figure 2.4. The probabilities at the lowest level represent quantisations of the input patterns. The variables at intermediate levels represent object parts, while the ones at the top nodes represent complete objects. During the learning process the values stored at each node are changed according to the input pattern and the data fed back by the upper nodes. During recognition, the goal is to find the most likely set of states at each module, so that the combination of all the states best explains the input pattern.

2.2.6 A critical view on Hawkins' work

Hawkins' work is undoubtly a new approach to renew AI. Some enthusiastic commenters even consider Hawkins' theories a breakthrough and say he has reinvented AI and the way engineers think about the brain. D. George has achieved some promising results [30] on producing invariant representations using an HTM. Additionally, the HTM theory has already made its way into the first commercial application in 2009[6]. It is the helm of a video monitoring system, called Vitamin D Video. The HTM implementation was developed by Numenta[7], and was trained to recognise humans against other moving objects, such as animals, swaying trees or even moving lights. That problem is one example among many, where humans in general perform much better than traditional computer approaches.

Nonetheless, the history of AI has been made of many failures and quite few successes. Many "new approaches" of the past have proven just adequate solutions for specific problems, but still have not uncovered the basic principles behind intelligent behaviour. The successes of D. George so far are still hardly enough to prove the qualities or potential of the HTM as a general-purpose memory system. In the case of the first commercial application of the HTMs, Vitamin D Video, it is not clear if this new approach to the problem of telling humans from other moving objects is better than traditional approaches, or how much better it is compared to them.

So far, it is still not clear if the memory prediction framework and the hierarchical temporal memory

[6] http://www.vitamindinc.com (last checked 2009.11.24).

[7] Numenta is a company founded by Jeff Hawkins himself. http://www.numenta.com (last checked 2009.11.24).

models are the desired ultimate, general framework of intelligence. Despite the recent successes, there are still some phenomena which may not be clearly explained by the models, and a few questions remain unexplained.

One problem not addressed by Hawkins is that of old memories which are suddenly recalled and keep demanding one's attention. How to explain the behaviour of being attached to a song, for example? That behaviour is very common among humans. If one does not hear a song that he likes for several years, he will possibly forget it. However, after several years, if he listens to that song again, he can not stop paying attention to it. According to the memory prediction framework only, that behaviour can not be explained. Actually, the song is a known sequence. The brain's predictions are correct. Hence, the sensorial inputs should be ignored rather than devoted all the attention[8]. According to the theory, once a sequence is learnt, the brain stops paying attention when it occurs and simply processes it according to the previous experience. Humans only pay attention if or when something unusual happens. However, it is not so when the brain is presented a song he had not heard for a long time: it is familiar, and it did not receive any attention when it was fashion. But after a long time the brain simply cannot avoid devoting it all the attention, and possibly recalling remarkable emotional states associated with the song. That problem is not clearly dealt with in the theory.

One possible explanation to the urge to pay attention to some old sequence happening again after a long time may be that the old sequence was in risk of *vanishing*. Being unused for a long time it was a candidate to be forgotten. However, Hawkins also does not clearly explain the mechanism the brain uses to *select* what to *forget*. The MPF gives no insight into that issue. It is known that old memories tend to vanish, and some of them may be recalled even long after they were considered forgotten. However, it is important to note that this cannot be pointed out as an error in the model. At most, it proves that the model is incomplete.

P. Almond [2] proposes a memory model that solves the forgetting issue. Almond's model is very similar to the HTM model, but it comprises a possibly unlimited number of levels in the hierarchy. Each level of the hierarchy receives data from the lower levels, just as in the HTM model. But Almond also adds a mechanism to forget that is just based on a measure of age. The lower levels of the hierarchy erase data after a period of time. The higher levels erase data after longer periods. The time gap increases with the ranking of the level. The idea is biologically inspired, since humans also seem to forget quicker the smaller details. The higher levels abstractions remain in memory for longer periods before being forgotten.

Another open question is that of how to make a connection between the various parts of the brain.

[8] This problem is devised by Michael Sippey, in his blog http://sippey.typepad.com/filtered/2006/12/jeff_hawkins_an.html (last checked 2009.11.24).

As explained in Section 2.2.2, the brain comprises several parts which are responsible for different behaviours. The neocortex is just the region usually associated with *intelligent behaviour*, but it depends on the other parts of the brain to work. The question of how to merge together the *knowledge*[9] in the *old brain* with the intelligent features stored in the neocortex is not addressed in the MPF. However, it is known that human intelligence is a mix of high level reasoning, emotions, intuition and a set of nate behaviours stored into the old brain. Hawkins foresees the MPF as a tool to produce intelligent machines with a new and different kind of intelligence, free from desire, from the will for power or the urge for mating. However, there is no evidence so far that intelligence can exist without those low level issues. Indeed, it is known that most of the times the perceptions and decisions taken in a fraction of a second are more accurate than those taken after careful thinking [31]. The brain very quickly captures the "big picture" and proposes a solution that is often correct. After careful thinking and paying attention to some details, the brain comes up with a different decision. However, it turns out that the second decision is often incorrect. An example of this behaviour is very well known by the students. Possibly all the students have learnt that the first idea that first comes to the mind is the correct answer to an exam question. After careful thinking, the brain comes up with something distorted and incorrect. It is also like that in many other situations. Again, those issues are not addressed in the MPF, and cannot be explained by the original model.

Yet another open question is how the brain handles associations. It is well known that humans learn better by associating concepts to some other previously learnt concepts. It is fairly easier to learn something after an analogy with something already learnt than to retain something completely new, unless that *something new* carries a strong dose of emotion (fear, awe, etc.). One possible explanation for this phenomenon is that by relating one concept to another, the brain is able to create the invariant form in a quicker and more efficient way, by using previously established synapses of the related concept or model. If the experience carries a strong dose of emotion, it may then be totally or partially stored into another region of the brain (namely, the amygdala), in which case the process is a little different. However, those features are also not addressed by the HTM model.

Moreover, Hawkins still does not uncover the basic brain algorithm. He may have uncovered part of the algorithm to produce invariant forms, but the ability to create invariant forms and models of the world is just one part of the whole system. Storing sequences, following those sequences, reasoning over those sequences, are just some of the problems that still need to be solved in the quest towards intelligent machines.

However, up-to-date research seems to point out that the MPF model is correct. A very recent study

[9] According to [86], J. Hawkins states that "When you are born, you know nothing.". Therefore, he seems to consider knowledge only what is learnt. Nate behaviours are, in consequence, not true knowledge, just natural abilities.

by Hunt and Cavanagh shows that the brain actually perceives a change in the direction of gaze before the eyes move [43]. Observers shifted their gaze to a clock with a fast-moving hand and reported the time perceived to be on the clock when their eyes first landed. The reported time was 39 ms earlier than the actual time the eyes arrived. In a control condition, the clock moved to the eyes, mimicking the retinal motion but without the eye movement. Here the reported time lagged 27 ms behind the actual time on the clock when it arrived.

2.3 Some memory models

Long before Hawkins' work on the human memory model, other researchers proposed models which somehow tried to mimic specific characteristics of the human brain. The brain works mostly as an associative memory, in which one small clue is enough to pull a complete datum, and from it possibly a complete sequence and associated concepts. Willshaw and Hopfield Networks are two remarkable approaches in that aspect, and they are briefly described below.

2.3.1 Associative memories

An associative memory is one which is content-addressable. Knowing part of one input vector, the remainder can be retrieved from the memory using the known part as a cue (as a memory address). The brain works like that. For instance, the first name of a familiar person is enough to retrieve the full name. Associated with the name it is also possible to retrieve a plethora of information about that person, as well as other people with the same or similar names. This behaviour is very tricky to implement in traditional computer models. However, the concept of associative memory is actually very easy to define in formal terms. In formal terms, such a memory is a system that always tends to a finite set of states.

Let's consider such a system, described by an N-dimensional vector of possible states X:

$$X = \{X_1, X_2, ..., X_N\} \tag{2.4}$$

Let's also assume that the system has only a finite number of M stable limit points $X_a, X_b, ..., X_M$. If the system is put in a state close enough to the stable state X_a, say at point $X_a + \Delta$, then it will tend over time to the stable state X_a.

The input $X_a + \Delta$ can be regarded as the cue to the memory, and X_a will be, with high probability, the datum retrieved. Under the specified conditions, the system must work as a content-addressable memory, where the contents stored correspond to the stable states of the system. If the set of stable

states can be easily updated, then the system is versatile enough. It can be used to learn new memories, thus being of practical use as an associative memory.

2.3.2 Willshaw network

In 1969 D. Willshaw et al. [107] proposed a memory model that later became known as the "Willshaw Network" (WN). The Willshaw network is based on the idea of a holographic memory. A hologram can be created by filtering light through planes with pinholes. Analysing the rays of light emerging back from different pinholes, it is possible to filter out the undesirable interferences and reconstruct the pattern of pinholes in one of the planes, knowing the other one. That is similar to the working principle of an associative memory: knowing one vector, it is possible to retrieve another one from the memory, filtering out the noise below a given threshold.

Willshaw proposes a memory model based on a neural network which can be represented as shown in Figure 2.6. The memory consists of rows and columns, which compose a matrix of points in the intersections. In those intersection points there may or may not be connections (synapses). In the beginning there are no connections at all, i.e., the neural network has only neurons, but no synapses. The matrix represented is all zeros. The rows represent addresses, the columns are data to be read or written. During a write operation, connections are established in the points of the matrix where an active row (address bit set to 1) and an active column (column receiving a bit set to 1) intersect. During a read operation, the rows which receive ones are activated again, and the number of points where those lines have established connections are summed columnwise. The output vector d is composed of zeros where the sum is below a given threshold T and ones where the sum is above it. The threshold is a parameter of the system, defined by the user.

For a data vector of size N, Equation 2.5 describes the output vector, where S_{ij} is the binary value stored at the connection between row j and column i.

$$d_i = f\left(\sum_{j=0}^{N-1} S_{ij}a_j\right), \quad f(x) = \begin{cases} 1 & if \quad x \geq T_i \\ 0 & if \quad x < T_i \end{cases}, \quad 0 \leq i < N \tag{2.5}$$

Figure 2.6 illustrates a Willshaw network where the pairs (1001) → (0011) and (1010) → (1001) have been stored. For a threshold of 2, reading is exact. However, reading errors will happen as soon as the memory reaches saturation, which may happen very fast if the memory is not to be erased or reset. Henson and Willshaw [40] propose a method of *synaptic decay*, which can make the memory forget old associations and, therefore, prevent it from exceeding its capacity. In that case, in each learning episode some synapses will be erased in a random manner. One problem of that approach is that there is no guarantee about which information is corrupt or not. The probability of the most recent data still being

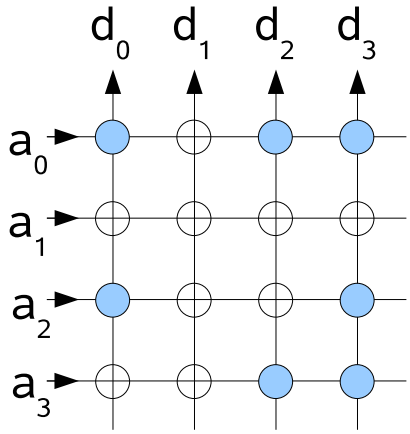


Figure 2.6: Example of a Willshaw network with some data. The blue circles represent connections where data have been stored.

accurate is, however, higher than that of the older data, because of the fact that older information has been exposed to the random deletion mechanism for a longer time. A better alternative analysed by Henson is to explicitely select synapses to erase in function of their age, or of their recent usage. In that case, the probability of affecting very recent data, or data that has been used recently, is considerably reduced. However, that means the memory will fail completely recalling older associations which have not been used recently, thus being only suitable to work as a short term associative memory.

2.3.3 Hopfield network

Another kind of associative memory is the Hopfield Network (HN) [42]. The Hopfield network is a model very robust to noise and failure of individual units, strongly relying on heavy parallel processing and feedback, as shown in Figure 2.7 for a network of four neurons. That is basically a neural network, where each neuron asynchronously adjusts its state V by evaluating the sum of its inputs as shown in Equation 2.6:

$$\begin{matrix} V_i \to 1 \\ V_i \to 0 \end{matrix} \quad if \sum_{i \neq j} T_{ij} V_j \quad \begin{matrix} > U_i \\ < U_i \end{matrix} \quad , i, j \in [1, .., N] \tag{2.6}$$

T_{ij} is the strength of the connection between neurons i and j—and it might be zero, if those neurons are not connected. U_i is the firing threshold, fixed for each neuron, and N is the number of neurons.

Typically, no neuron must be connected to itself, i.e., $T_{ii} = 0$, and the connections are symmetric, which means $T_{ij} = T_{ji}$.

Hopfield analyses the properties of such a system, and proves that it tends to a finite number of stable states, coincident with assigned memories. The idea can be modelled in terms of Energy of the

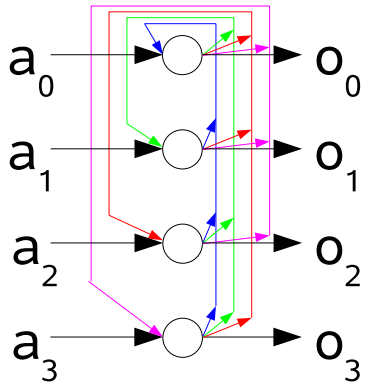


Figure 2.7: Example of a Hopfield network: neural network where the outputs of each neuron are connected to the inputs of other neurons.

system, described by the equation:

$$E = -\frac{1}{2}\sum_{i<j} T_{ij} V_i V_j + \sum_i U_i V_i \tag{2.7}$$

The system will converge to points corresponding to local energy minima. Learning is, therefore, equivalent to train the neural network by adjusting connections' strengths so that the energy function of the network has a local minimum at the desired point.

States close to one another are confused and tend to merge in the memory. Unknown starting points tend to the stable state 0 (state vector composed only of zeros), which means that unknown memories will most likely lead to a null output. Under normal circumstances, it also suffices to know just part of the contents—if the input state is closer enough to the desired datum, it will eventually converge to it.

It should also be noticed that learning requires training the neural network, and that often can not be done in a single pass. The latter constitutes an important drawback of the model if the goal is to build a biologically inspired memory. Humans have the ability of one-shot learning, and it is used on a regular basis.

2.4 Summary

This chapter briefly reviews the modern mainstream ideas of what is human intelligence, from Gottfredson's informal definition to Legg's formal definition. Gottfredson defines intelligence as an ability to learn, understand and comprehend, without defining the concepts of understanding and comprehending. Legg presents a more pragmatic definition, considering intelligent an agent that makes the right decisions in a variety of environments. Dobrev proposes that an agent that makes the right decision 70% of the times should be considered intelligent.

It is generally accepted that the brain is the source of intelligent behaviour. The brain comprises several different areas, but the neocortex seems to play the most important role in intelligent behaviour. Hawkins and other authors propose that intelligence is the result of using a sophisticated memory, more than huge processing power. The brain is probably an associative memory, divided in different levels of abstraction. Based on that idea, Hawkins proposes the Hierarchical Temporal Memory and the Memory Prediction Framework as biologically inspired theories of the mind. Other simpler associative memory models were also revised, namely the Willshaw and the Hopfield networks.

From the present analysis, it seems reasonable to assume that an intelligent robot must rely on the use of a sophisticated memory, and that memory will most probably be associative in nature.

Chapter 3

Sparse Distributed Memory

Some men see things as they are and ask why. Others dream things that never were and ask why not.
(George Bernard Shaw, Irish dramatist)

Contents

Chapter 2 showed how intelligence is most probably the result of using a sophisticated memory, and described the hierarchical temporal memory and some other associative memory models. This chapter describes another memory model, the Sparse Distributed Memory (SDM). The SDM was proposed in the 1980s, based on the same idea that the brain is mostly a memory system. Contrary to the other models, the SDM is proposed with a very complete and sound mathematical basis, proving that the memory inherits the characteristics of a high dimensional boolean space.

3.1 Introduction

In simple learning tasks, algorithms such as look-up tables, where each table entry is associated with a state or state-action pair, show reasonable performance. That approach, though, is only suitable for a small number of states and actions. More powerful approaches are needed for continuous or large discrete spaces, where good generalisation skills are demanded for a better learning experience.

Neural Networks (NN) are another method widely used to perform learning tasks, where layers of artificial neurons, connected by relations which have an associated variable weight, are trained to learn different functions and associations or recognise patterns. Those neural networks, though, are more suitable to work on low-dimensional domains. They produce good results on small networks, but scaling up to thousands or millions of nodes is not a trivial problem [87]. Scaling up a system requires more computational power, due to the increased number of variables and the increased number of interactions between them, and those requirements may be unsuitable for tasks such as robot navigation based on large amounts of sensorial data. Besides, NN in general are very hard to tune and may suffer catastrophic forgetting under normal circumstances.

In the late 1980s, Pentti Kanerva proposed the Sparse Distributed Memory model, a kind of associative memory based on the properties of high-dimensional binary spaces [52]. According to P. Denning [20], J. Albus [1] and D. Marr [63] independentely developed similar theories, based on the observation of the structure of the human nervous system. The underlying idea behind a SDM is the mapping of

a huge binary memory onto a smaller set of physical locations, so-called *hard locations*. As a general guideline, those hard locations should be uniformely distributed in the virtual space, to *mimic* the existence of the larger virtual space as accurately as possible. Every datum is stored distributed by a set of hard locations, and retrieved by *averaging* those locations. Therefore, recall may not be perfect, accuracy depending on the saturation of the memory. Kanerva's proposal is based on four basic ideas, as presented by the author:

1. The space 2^n, for $100 < n < 10^5$, exhibits properties which are similar to humans' intuitive notions of relationships between the concepts. This means that it makes sense to store data as points of the mentioned space, as will be described in more detail in Section 3.2.3.
2. Neurons with n inputs can be used as address decoders of a random-access memory[1].
3. Unifying principle: data stored into the memory can be used as addresses to the same memory.
4. Time can be traced in the memory as a function of where the data are stored, if the data are organised as sequences of events, as explained in Sections 3.6.8 and 8.4.2.

Those four principles are the guidelines behind the whole model. The mathematical foundations and the model itself are described in the next sections.

3.2 The space 2^n

As previously stated, Kanerva's Sparse Distributed Memory model is based on the properties of the boolean space $\{0,1\}^n$. For clarity, that boolean space will also be referred to as 2^n, or simply N, following Kanerva's notation. Below is a summary of the most important concepts, relations and properties of N. A more detailed analysis can be found in [52] or [8].

3.2.1 Concepts and relations

Before analysing the properties of the space itself, some preliminary concepts must be defined:

n is the number of dimensions of the space
2^n is the number of points in the space

In theory, the memory can store up to 2^n different memory items: one item per point in the space. Any point in N can be represented by an n-tuple of binary digits $< b_0, ..., b_{n-1} >$, or a binary number,

[1] Kanerva proposes that a SDM can be built based on a neural network. Apart the fact that the NN model is biologically inspired, which is not a significant advantage *per se*, there is no other clear reason to prefer the connectionist approach. Indeed, many authors prefer to model the SDM based on linked lists or similar data structures.

$b_0...b_{n-1}$. Those binary numbers have no particular order—it is not possible to tell whether 001 comes before or after 000. Bearing that in mind, additional concepts may be defined. Let's assume that x is a random point of space N. Then, by definition:

- O point $< 0, ..., 0 >$, origin of the binary space
- $`x$ complement of $x$. The complement $`x$ has 0 where x has 1, and 1 where x has 0
- $|x|$ norm of x. The norm of x is the number of 1 (ones) in the binary representation of x

Building on the definitions above, some operations, characteristics and additional concepts may also be defined for the space 2^n, as follows.

Difference

Difference between two points x and y is defined as the bits in which the points differ, or the exclusive OR. The difference thus defined is commutative:

$$x - y = y - x = x \oplus y \tag{3.1}$$

Example 1:

Let's consider $x = 1010$ *and* $y = 1001$.

$x - y = y - x = x \oplus y = 0011$ ■

Distance

There are several metrics to express the distance between two points in a binary space. However, since the space is unsorted, the position of the bits is irrelevant. Hence, Kanerva uses the Hamming distance[2]. The Hamming distance is defined as the number of symbols in which two strings are different. In a binary space, the distance between two points x and y is equal to the number of bits in which x and y differ, or the norm of the difference between x and y:

$$hd(x, y) = |x - y| \tag{3.2}$$

A quick way to compute the Hamming distance is to count the number of ones resulting from an exclusive OR operation, as represented in Equation 3.3. In the equation, x_i is the i^{th} bit of vector x, and y_i is the i^{th} bit of vector y.

$$hd(x, y) = \sum_{i=0}^{i=n} x_i \oplus y_i \tag{3.3}$$

The distance between two points is also a measure of similarity of two memory items. Two points close to each other are considered *similar*, while points very distant from each other represent very different patterns.

[2]The problem of computing distances is discussed in more detail in Section 3.4 and Chapter 7.

Example 2:

Consider $x = 1010$, $y = 1001$ *and* $z = 1011$.

$hd(x, y) = |x - y| = |0011| = 2_{(10)}$

$hd(x, z) = |x - z| = |0001| = 1_{(10)}$

Since $hd(x, z) < hd(x, y)$, *it can be affirmed that* z *is more similar to* x *than* y. ■

Betweenness

If point y is between x and z, it is represented as $x : y : z$. It is said that point y is between points x and z if and only if the distance from x to z is the sum of the distances from x to y and y to z:

$$x : y : z \quad \text{iff} \quad d(x, z) = d(x, y) + d(y, z) \tag{3.4}$$

A consequence of the definition is that every single bit of y coincides with the corresponding bit either in x or z:

$$x : y : z \quad \text{iff} \quad y_i = x_i \vee y_i = z_i, \quad i = 0, ..., n - 1$$

Some characteristics of betweenness are symmetry, recursivity and no transitivity:

Symmetry	$x : y : z \Rightarrow z : y : x$
Recursivity	$w : x : y : z \Rightarrow w : x : y$ and $x : y : z$
No transitivity	$w : x : y : z \not\Rightarrow w : x : z$

Example 3:

Consider $x = 010$, $y = 110$ *and* $z = 101$.

Computing the distances: $hd(x, y) = |100| = 1_{(10)}$, $hd(y, z) = |011| = 2_{(10)}$ *and* $hd(x, z) = |111| = 3_{(10)}$.

Since $hd(x, z) = hd(x, y) + hd(y, z)$, *it is possible to affirm that* $x : y : z$. ■

Orthogonality

Orthogonality is one of the most important concepts in the boolean space. Two points x and y are orthogonal (also said perpendicular or indifferent) if and only if, the distance between x and y equals half the number of dimensions:

$$x \perp y \quad \text{iff} \quad d(x, y) = \frac{n}{2} \tag{3.5}$$

$\frac{n}{2}$ is called the *indifference distance* of 2^n.

A noticeable property is that if x is indifferent to y, it is also indifferent to its complement 'y. x is halfway between y and its complement 'y.

Example 4:

Consider $x = 1010$ *and* $y = 0110$*, in the space* 2^4*.*

$hd(x, y) = |1100| = 2_{(10)}$*, therefore* $x \perp y$*.*

$hd(x, `y) = |0011| = 2_{(10)}$*, therefore* $x \perp `y$*.* ■

Circle

The concept of a circle is similar to that of a circle in an Euclidean space. However, the measure of distance here is the Hamming distance. Hence, a circle of radius r and centre x is defined as the set of points whose distance to x is less than or equal to r:

$$C(r, x) = \{y | d(x, y) \leqslant r\} \tag{3.6}$$

Example 5:

Consider $x = 1010$*, in the space* 2^4*.*

$C(1, x) = \{1010, 1011, 1000, 1110, 0010\}$ ■

3.2.2 Properties of 2^n

The properties of N play a fundamental role in the theory, for they are the reason that the Sparse Distributed Memory will exhibit some characteristics similar to the human brain. Here is a list of the most important of those characteristics.

2^n as a spherical space

An n-dimensional unit cube can have its vertices placed into a sphere of diameter $\sqrt{n}$, the diameter being an Euclidean distance. Based on that observation, Kanerva establishes an analogy of the space 2^n as a spherical one. 2^n is, according to the analogy, the surface of a sphere with radius $\frac{\sqrt{n}}{2}$, exhibiting remarkable properties:

- Any point of N can be considered the origin of the space. Since it is spherical, it does not matter if the origin is $(0, 0, ..., 0)$ or any other point;
- A point and its complement are like two poles of the sphere, at distance n and with the entire space in between them;
- Any points perpendicular to the poles and halfway between them, are like the equator.

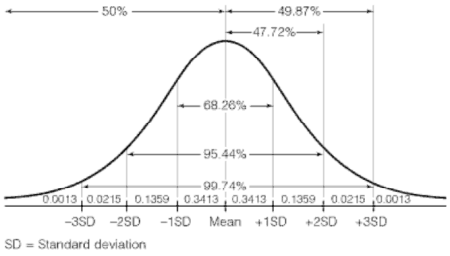


Figure 3.1: Distribution of values of a normal distribution. Source: [88].

Distribution of the space 2^n

Since any point in the space can be considered the origin O, the number of points which are at distance d from point x is the number of combinations of d of n bits, given by the binomial coefficient:

$$\binom{n}{d} = \frac{n!}{d!(n-d)!} \tag{3.7}$$

N can, therefore, be described by the binomial distribution $B(n,p)$, where p is the bit probability $p = \frac{1}{2}$ and n is the number of dimensions of the space. Following Kanerva's notation, that probability distribution function will be denoted as $N(d)$. According to the probability theory, $N(d)$ has mean value $E(x) = np = \frac{n}{2}$, and variance $Var(x) = np(1-p) = \frac{n}{4}$. Therefore, from the statistics theory, a good normal approximation F for the distribution function $N(d)$, with mean $\mu = \frac{n}{2}$ and standard deviation $\sigma = \sqrt{\frac{n}{4}}$, is as shown in Equation 3.8:

$$N(d) = Pr\{d(x,y) < d\} \cong F(d;\mu,\sigma) = F\left(d; \frac{n}{2}, \sqrt{\frac{n}{4}}\right) = F\left(\frac{d-\frac{n}{2}}{\sqrt{\frac{n}{4}}}\right) \tag{3.8}$$

In the equation, $Pr\{d(x,y) < d\}$ is the probability that the distance between points x and y is less than d bits.

Tendency to orthogonality

Since the distribution of points in N can be approximated by the normal distribution, as stated above, an interesting property arises from that fact. As Figure 3.1 shows, more than 68.2% of the points lie within one standard deviation from the mean, in the interval $[\mu - \sigma, \mu + \sigma]$. 95.44% of the points are within two standard deviations. 99.74% of the points lie within 3 standard deviations of the mean. Actually, the interval $[\mu - 5\sigma, \mu + 5\sigma]$ accounts for *almost all* of the points in the space, namely 99.9999% of them. Only a *residual* number of points lie out of that interval. The consequence of this characteristic is that for any given point x, most of N lies at approximately the mean distance $\frac{n}{2}$. As Kanerva states, *"most of the space is nearly orthogonal to any given point"*. Considering x and 'x the poles, almost all

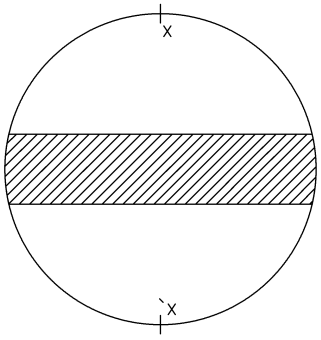


Figure 3.2: The tendency to orthogonality of the space 2^n means that most of the space lies close to the equator for any given point x.

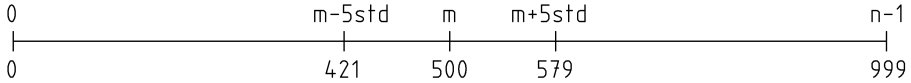


Figure 3.3: 99.9999% of the space 2^{1000} lies in the interval [421, 579] bits from any given point.

the points of N lie close to the equator, as Figure 3.2 shows. The larger the n, the more pronounced is the effect.

Example 6:

Consider $n = 1000$, and point x the origin of the space. The mean value is $\mu = \frac{n}{2} = 500$.

The standard deviation is $\sigma = \frac{\sqrt{n}}{2} = 16$ bits.

$5\sigma = 79$ *bits.*

Therefore, 99.9999% of the space lies in the interval [421, 579] bits far from x and ‘x, as Figure 3.3 illustrates. ■

Distribution of a circle

Another interesting property is that of a circle $C(r, x)$ in the space 2^n. $C(r, x)$ is the set of points no more than r far from x, as defined in Section 3.2.1. The area of the circle is defined as the number of points enclosed in the region, which can be determined as the sum of the first $r+1$ binomial coefficients. Kanerva studies the properties of an arbitrary circle $O(r, x)$, and defines the area of $O(r, x)$ as:

$$|O(r,x)| = \binom{n}{0} + \binom{n}{1} + \cdots + \binom{n}{r} \tag{3.9}$$

According to the previous definition of $N(d)$, in Section 3.2.2, and the normal approximation expressed in equation 3.8, the area is given by:

$$|O(r,x)| = N \cdot N(r) = N \cdot F\left(\frac{r - \frac{n}{2}}{\sqrt{\frac{n}{4}}}\right) \tag{3.10}$$

Again, the area of circle $O(r, x)$ follows approximately a normal distribution. Therefore, the circle only includes a small portion of N if r is small enough. When r gets close to $\frac{n}{2}$ (i.e., $r = \frac{n}{2} - \varepsilon, \varepsilon \to 0$), it will enclose almost half of N, and once r surpasses $\frac{n}{2}$ (i.e., $r = \frac{n}{2} + \varepsilon, \varepsilon \to 0$), the circle will contain *almost all* the space.

Another important point to notice is that, if $r \leq \frac{n}{2}$, most of the points in the circle lie close to its periphery. That is expectable, considering that the area of the circle follows a normal distribution. Kanerva demonstrates it as follows. Point x has n neighbours 1 bit away, $\binom{n}{2}$ that are two bits away and, in general, $\binom{n}{r}$ that are r bits away. For small values of r compared to n, the number of neighbours grows nearly exponentially with r. The growth rate is the ratio of two binomial coefficients, as given by Equation 3.11, so that when $r < \frac{n}{3}$, the size of the circle, at least, doubles at each unit increment of the radius r:

$$Growth(n, r) = \frac{\binom{n}{r+1}}{\binom{n}{r}} = \frac{n-r}{r+1} \tag{3.11}$$

3.2.3 Memory items as points of 2^n

In the implementation of a Sparse Distributed Memory, each memory item is stored as an n-bit vector. Each n-bits vector can be considered a point in N. Therefore, the SDM will inherit the properties of the binary space.

Some characteristics

From the analysis of the properties of N, important characteristics can be pointed out for this memory.

- Distance between two points is a measure of similarity between two memory items. The closer the points, the more similar the stored vectors.

- As shown in Section 3.2.2, almost all of the space is indifferent to every single point x[3]. Most of the space lies at a Hamming distance of about $\frac{n}{2}$ bits away from x. Only a small portion of N lies close to x. That seems close to what happens in human memory, where every concept seems *close* (related) to a small number of other concepts, and *indifferent* to the vast majority of the other concepts in the memory. Nonetheless, skipping from concept to concept (point to point), it is possible to reach any concept (point) from any other concept (point) stored into the memory, using the appropriate access radius.

[3] A point is indifferent if it is orthogonal, as defined in Section 3.2.1.

Concept boundaries

The idea of a concept boundary is very important, for it sheds some light on the problem of retrieving the right datum from the pool of information stored in the memory. It is best explained using an example.

Let's consider a million random points of 2^{1000}, representing a million memory items of 1000 bits each. A circle C of radius $r = 425$ bits will encircle $\binom{1000}{425} + \binom{1000}{424} + \ldots + \binom{1000}{1}$ points. That is a portion less than 1.18×10^{-06} of N, which is about one millionth of the space. An arbitrary memory item can be considered as the centre of C. The points inside C can be considered as belonging to the same concept, while the ones outside can be considered as not belonging to it. Therefore, Kanerva suggests that a radius of 400 bits, which encircles a portion of about 10^{-10} of the space (more exactly, 1.36×10^{-10}), should provide a satisfactory probability of selection of the exact concept. Hence, under normal circumstances, operating with 600 out of 1000 bits of each datum correct should provide satisfactory results. A probability of being correct of 0.6 should suffice for normal operation.

Incomplete data

As shown above, a 0.6 probability of being correct should suffice to retrive the correct datum with high probability. Therefore, it is enough to know precisely only 200 out of the 1000 bits (20% of them) to, possibly, retrieve the correct datum. The unknown bits could be filled at random. In that case, the probability of filling in any given bit with the correct value is 0.5. The probability of getting the correct datum is, therefore $(0.2)(1) + (0.8)(0.5) = 0.6$. Querying the memory for such a datum, one should be able to retrieve the correct output most of the times.

Kanerva proposes that this property can mimic the human ability to recognise an object in different contexts—with only 20% of the information accurate, humans are keen on retrieving the correct datum from the memory.

3.3 A sparse, distributed memory

As explained in the previous sections, the space 2^n exhibits properties and characteristics in many aspects similar to the human brain. It is, then, an interesting model to work with in order to build an associative memory. However, implementation of a memory based on the principles of 2^n requires further analysis, for there are some practical problems that need to be dealt with.

3.3.1 Memory size and sparseness

The first practical problem one has to solve before implementing a SDM is the vastness of the space. Indeed, N can become extremely large as the number of dimensions n increases. For a robot dealing with state vectors of 1000 bits $N = 2^{1000} \approx 10^{301}$. As a comparing measure, the number of neurons in the human nervous system[4] is about $2^{36} \approx 10^{10}$, or a fraction of 10^{-291} of N. Another comparison, pointed by Kanerva, is that a century contains less than 2^{32} (4.3×10^9) seconds, or a portion of 10^{-292} of N.

A memory with 2^n locations may, therefore, be infeasible to implement, for a large n. To overcome that problem, Kanerva proposes that the memory be *sparse*: it must contain only a small portion of physical addresses, much less than the addressable space. Those physical addresses are called *hard locations*, and they are just a small subset of all the addressable space N. According to the original proposal, the hard locations are given right from the start, before the memory starts being used. Their addresses are distributed randomly in the space 2^n, and cannot be modified during normal operation of the memory—only the contents may be updated. Following Kanerva's notation, N' denotes this subset of N, and x' denotes a hard (physical) location.

Kanerva proposes that N' should be just a small portion of N. His example is that 2^{20} (about one million) should suffice to represent a space of 1000 dimensions. N' is, therefore, a portion of about $2^{980} \approx 10^{-297}$ of N, which makes the memory a very sparse one. Indeed, every hard location of such a memory represents, on average, a fraction of 2^{-980} addresses of N, or one millionth of the whole space N. Due to the tendency to orthogonality (explained in Section 3.2.2), a circle of radius 425 contains less than one millionth of N, as shown in Section 3.2.3 when dealing with concept boundaries.

In conclusion, any subset of hard locations N' should appropriately represent the whole space N, if one accesses the memory looking in an appropriate radius. For $n = 1000$, a subset of 2^{20} hard locations and an access radius of 425 bits should produce satisfactory results.

3.3.2 Writing

Since every hard location x' accounts for a small amount of N, Kanerva proposes the concept of distributed storage. Writing a datum x may affect more than a single hard location. x must be written in all the hard locations within an access circle, as illustrated in Figure 3.4. When a write operation is performed, the contents of all the hard locations at distance $d < r$ are updated using some algorithm. The new datum may replace the previous datum, or be summed, for example, as explained further. Storage, in consequence, is distributed in an access circle.

[4]The number of neurons of an adult human varies a lot, but common figures are in the range 10–100 billion.

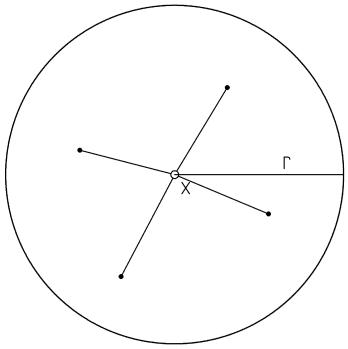


Figure 3.4: Reading and writing affect all the hard locations in an access circle.

3.3.3 Reading

Reading back datum x means taking the representative of x from N. This can be done, again, pooling all the hard locations within the access circle and using some algorithm to recover the desired information. The set of words retrieved from the access circle, when retrieving data stored at x, will be denoted as $D(x)$.

Reading, therefore, may not be exact. Depending on the algorithms used to extract the word from $D(x)$ and the degree of saturation of the memory, the error may be marginal or significant.

Various approaches can be used to retrieve the representative of x from $D(x)$. One method is to select the datum that is closer to the desired one, or the most similar to x. Another approach is to select the most frequent word in $D(x)$—a method that may be computationally expensive. A third possibility is to randomly select a word from $D(x)$. The latter is computationally attractive, but the probability of selecting the best element is very low. The best method, Kanerva proposes, is to compute an element of N that may adequately represent $D(x)$, even though it may not be one element of it. The *natural* candidate to be that representative element is the average value of the words of $D(x)$. The ith bit of the average value can be computed by summing all the ith bits of all the words in $D(x)$, and thresholding with half the number of words, as shown in Equation 3.12:

$$W_i(x) = \begin{cases} 1 & iff \quad \sum_{\xi \in D(x)} \xi_i \geq \frac{|D(x)|}{2} \\ 0 & otherwise \end{cases} \tag{3.12}$$

The above described approach is adequate in most circumstances, and has been widely used in several applications, as further detailed in Section 3.6. However, its behaviour is more appropriate when dealing with random data, where zero and one are equally probable. In that case $P(0) = P(1)$, and the threshold 0.5 is a perfect choice. If the probabilities $P(0) \neq P(1)$, as happens with non-random data, the threshold 0.5 may be inadequate. In that case, Ryan and Andreae [90] suggest to set the threshold equal to the probability $P(0)$. That small change shall significantly improve the performance of the SDM with non-random data, according to the author. However, there is still the need of setting

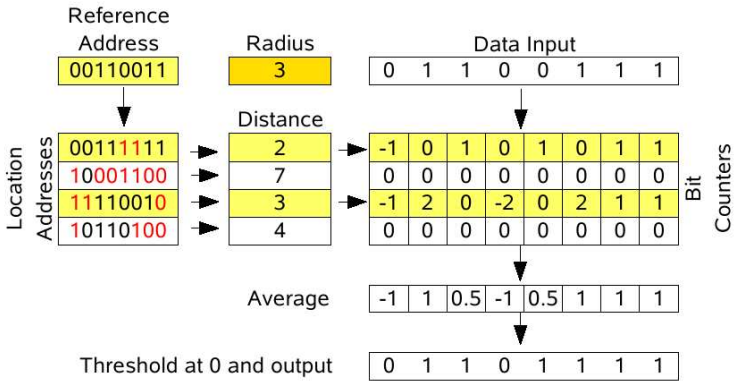


Figure 3.5: Architecture of Kanerva's model of a SDM.

up a threshold, which in some applications may be a problem (for example, if $P(0)$ is unknown, dynamic or unpredictable).

3.4 Implementation of a SDM

The SDM model is suitable to be implemented using several different techniques, including linked lists and neural networks. The former model is the one implemented in the present work. The latter is the most important from the point of view of the biological interpretation, but has no significant practical advantages over the non-connectionist model.

3.4.1 SDM using linked lists

One possible model for a SDM is as shown in Figure 3.5: an array of addresses which may or may not be activated by the reference address, in function of the distance between the reference address and the very memory addresses; an array of bit counters which store the data; and columnwise adders and thresholders that calculate the output datum when reading.

It should be mentioned that in the auto-associative version of the memory the same array could simultaneously be used as address and input data, overcoming the need of the second array. An auto-associative version of the SDM was implemented in the present work—it is described in Section 6.1.2 and shown in Figure 6.1.

Addressing

The reference address is the input address, where data is to be stored, or to be pooled. In *traditional* memories the input address would only activate a single memory location. In the SDM, though, it will

activate all the hard locations within the activation radius. To build the set of active hard locations, the Hamming distance between the reference address and every other physical address is computed. For every hard location, if the Hamming distance to the input address is less than or equal to the activation radius, that location is added to the set of active locations, otherwise it is ignored. In Figure 3.5, the first and third locations are activated, since they are within the selected activation radius. They differ 3 bits or less from the input address. In the picture, they are marked in colour.

Note that addressing is a very critical feature of the SDM. Kanerva proposes the Hamming distance shall be used, as defined in Section 3.2.1. However, several other metrics may be used.

Many authors do not use bitwise distances, as described in Section 3.6. In general, distances may be calculated using Equation 3.13, where x and y are vectors of m coordinates (possibly integers or floats, the formula is not specific for binary vectors):

$$d_n(x, y) = \sqrt[n]{\sum_{i=0}^{m} |(x_i - y_i)^n|} \tag{3.13}$$

For $n = 1$, d is the L1 norm, or the so called "city block" or "Manhattan" distance, where all the coordinates contribute proportionally to the distance. If x_i and y_i are bits, it is the same as the Hamming distance. If the bits are grouped (for example, as integers), then d will be different from the Hamming distance originally proposed by Kanerva. $n = 2$ gives the Euclidean norm. Higher values of n give a hypercubic metric norm. It should be noted that as n increases, the relevance of the largest differences increase proportionally, thus being an effective way to possibly deal with noise. For $n \to \infty$, only the largest difference contributes significantly to the distance, the other differences contribute only marginally to the distance and can possibly be discarded.

Hard locations

In a *traditional* memory every location consists of a memory element able to store a set of bits. In the SDM, every location consists of a set of counters, one per bit, which are incremented or decremented as ones or zeros are stored. Those counters are usually referred to as "bit counters", or "data counters". Initially, all the bit counters are set to zero, since the memory stores no data. Then, the counters are pooled in every read operation, and incremented or decremented in every write operation.

One drawback of the Sparse Distributed Memory model becomes clear at this point: while traditional memories store one bit per bit, in the SDM every bit requires a bit counter. Nonetheless, every counter stores more than one bit at a time, making the solution not as expensive as it might seem (i.e., the value of an arbitrary counter may contribute to readings and writings of many different addresses, as explained above—therefore, it may store, *on average*, more than one single bit of a single datum).

Kanerva calculates that such a memory should be able to store about 0.1 bits per bit of traditional computer memory. That value has been surpassed in other implementations, as described in Section 3.6.

Data writting

Data is written to the memory by incrementing or decrementing the selected counters. If a 0 is to be written, then the corresponding counter is decremented. If a 1 is to be written, then the counter is incremented.

In the example of Figure 3.5, if the input datum is to be written, the consequences for all the active hard locations are: 1 is subtracted from bits 0, 3 and 4; 1 is added to bits 1, 2, 5, 6 and 7.

Analysing the way data are stored, it is possible to infer the impact of rewriting the same datum many times. Data that are rewritten are less likely to be forgotten, for the counters affected will have been incremented or decremented more times, thus storing larger absolute values. They will, therefore be less likely to be affected by interference from other data. Every rewrite reinforces *something* in the memory, much likely what happens with the human brain.

Data reading

Since the data are stored in various hard locations at the same time, reading can be done by *averaging* the values of the active hard locations, as explained in Section 3.3.3. All the bits are first summed columnwise. If the sum of the bits for any given column is above a given threshold, then the corresponding bit is considered 1. Otherwise it is considered 0. Zero is a good threshold if, when writing, the counters are decremented to store zero and incremented to store one.

As the theory predicts, there is no guarantee that the data retrieved are exactly the same that were written. They should be, providing that the hard locations are correctly distributed over the binary space, and also that the memory has not reached saturation.

3.4.2 SDM using a neural network

A SDM as a linked list is a very straightforward model. However, the implementation as a Neural Network is also very easy to visualise and is biologically more plausible. The SDM can be modelled as a three-layered neural network. The first layer works as the addressing layer. The second is the hidden layer, and the third is the threshold and output layer.

The first layer can be built of perceptrons, with fixed weights. The perceptron works as a mapping

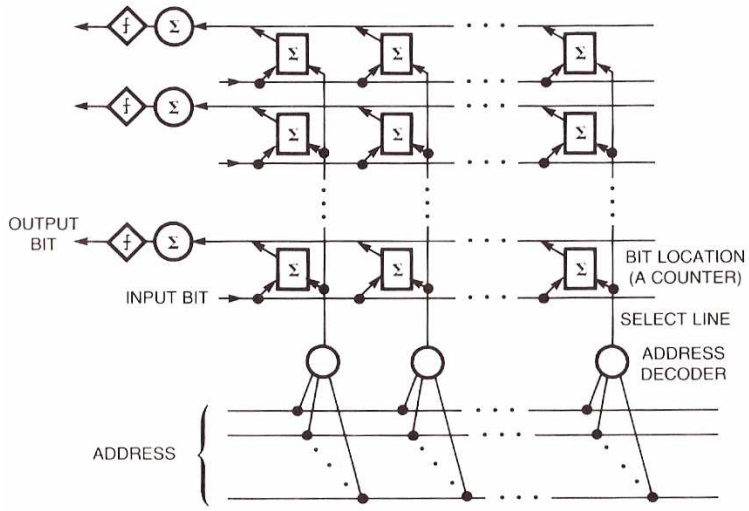


Figure 3.6: Model of a SDM implemented using a neuronal network. Source: [52].

function, mapping $\{0,1\}^n \rightarrow \{0,1\}$. It will fire if the number of ones in its input is above a given threshold. That is the same as firing if the input address is within the activation radius, or firing if the Hamming distance between the address decoded by the perceptron and the input is below the activation radius. Kanerva proposes that the weights of these neurons be fixed from the beginning, forcing the addresses to be distributed at random in the addressing space. A good random distribution of hard locations should produce satisfactory results. The output of each neuron is connected to a column of neurons in the hidden layer, so that firing of one neuron is equivalent to activating one memory address, as shown in Figure 3.6. Each row of neurons in the hidden layer is connected to a single neuron in the output layer.The weights of the synapses between the hidden and the output layer should work as bit counters. Learning (storage) is done by updating those weights, which is a kind of Hebbian learning.

Neurons in the output layer work as adders and thresholders, producing output one if the sum of the outputs of the corresponding row is above the given threshold, and zero otherwise.

3.5 Characteristics of the SDM

Being implemented based on the properties of the space 2^n, a Sparse Distributed Memory constructed following the models described in Section 3.4 should exhibit characteristics and behaviours, to a great extent, similar to those of the human memory. Namely, the following characteristics can be pointed out:

High dimensionality — Contrary to many popular approaches, SDMs are suitable for high-dimensional systems, due to the large number of address and input lines. Kanerva studies the example of 1000-bit vectors.

Associativity — SDMs are associative memories. It is possible to associate data patterns with other patterns, use data vectors as address vectors and the contrary, as well as work with partially correct information.

Error tolerance — SDMs exhibit good tolerance to noisy data and errors, due to distributed storage, associativity and thresholding techniques. They must perform at least as well as typical neural networks.

Natural learning — Depending on the saturation level, the SDM must be able to learn in one pass. Also, successive writing of the same datum will make it less likely to be forgotten.

Natural forgeting — Data in a SDM cannot be erased. It must be "forgotten," as time goes by and the memory becomes full.

Graceful degradation — In case of malfunction, the SDM must behave as a typical neural network. The output does not collapse all of a sudden, but becomes less accurate as more and more neurons fail. Output becomes noisy until it eventually becomes only garbage.

Incomplete data — SDMs can be operated with incomplete data. About 20% of the bits accurate and the remaining 80% set at random is enough to retrieve the correct datum with a high probability of success.

Parallel processing — The SDM can operate in parallel, just like it is thought the human brain operates, possibly achieving high speeds of operation for large or simultaneous inputs.

From the previously described characteristics, many behaviours usually considered to be typically human can be mimicked by the memory, making it strikingly close to the human memory's functional behaviour. P. Denning [20] points out the following behavioural similarities (to a great extent also outlined by Kanerva):

Knowing that one knows — From incomplete data it is possible to retrieve a pattern that should be close to the desired pattern. The retrieved pattern can be used to query the memory again, and eventually retrieve a pattern closer to the desired pattern. Repeating that procedure it is possible to either reduce the error iteratively and converge to the desired pattern, or, if the error increases between successive iterations, infer that one does not know.

Discover new relations — A SDM is a sophisticated tool to match and relate patterns. It can be used to discover relations and similarities between previously unrelated concepts.

Sequence storing — It is possible to store a pattern and associate it with another pattern, and the second with a third pattern, and so on. Starting from any but the last pattern, it is possible to retrieve the remainder of the sequence. The memory can, therefore, be used to store sequences and retrieve the continuation of a sequence of events when given a cue.

Reinforcement of an idea — Reinforcement of an *idea* which must not be forgotten can be achieved by successive writings of the corresponding pattern. Just like humans do with *ideas* they do not want to forget. It is possible to write the same input many times, if it is very important, or "relive" a sequence to avoid it being forgotten. Reliving can be implemented as retrieving a sequence from the memory and feeding it back for rewriting, at the same time, thus reinforcing it and making it less likely to be forgotten.

Retrieve data with incomplete information — Humans can retrieve data by matching similar but distinct patterns, in what is called analogy or methafor. That behaviour is used on a daily basis. However, typical computer memories require the use of exact addressing. To implement analogy or methafor-based reasoning, it is necessary to use sophisticated algorithms to process the data. In the SDM, addressing needs not to be exact: it is possible to retrieve the *closest* datum, or a list of vectors that differ less than a given radius from a given pattern. That makes the SDM suitable to work with incomplete information, and/or implement analogy and methafor in a natural way.

3.6 Previous applications and improvements

Sparse Distributed Memories have been studied and used since they were originally proposed by Kanerva in the 1980s. The earliest applications, such as described by Anwar and Franklin [4] and [5], were plain implementations of the original proposal, with minimal or no improvements. Other authors, such as Keeler [54] and Hely [38], have done thorough theoretical studies of the memory and compared it to other techniques, such as neural networks. The applications that are more relevant for the present work are the ones described below. This section contains a comprehensive review of those systems, which are briefly described, along with the problems that were found and the solutions that were adopted or proposed. This list does not cover all the SDM implementations. Although the SDM is not a mainstream memory model, it has been subject to many research work and there are many other applications.

3.6.1 The Stanford model

One of the most remarkable attempts to build a large SDM was that by Flynn, Kanerva and Bhadkamkar [25] at the Computer Systems Laboratory of the Departments of Electrical Engineering and Computer

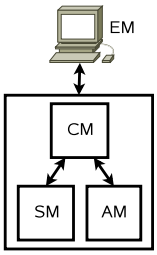


Figure 3.7: Blocks diagram of the Stanford Model of the SDM.

Science of the Stanford University, in 1988. That was later to be known as the "Stanford model," or the "Stanford prototype".

The Stanford model was a large prototype memory, with 256-bit addresses and from 8 K to 128 K hard locations for 256-bit words. It was built all on dedicated hardware, to avoid the practical problems of high processing times when running software simulations. The model was built as a first large-scale implementation of a SDM, and it intended to be a flexible and low cost playground where algorithms could be tested with the goal of developing better applications and making the most of or improving the original SDM model.

The Stanford prototype has four major hardware modules, as tentatively represented in Figure 3.7: 1) Executive module (EM); 2) Control module (CM); 3) Address module (AM); and 4) Stack module (SM).

The EM corresponds to the "focus," as defined by Kanerva. It is a workstation, responsible for the user interface. That workstation communicates with the other modules via a SCSI (Small Computer Systems Interface) port. The SDM is, to a large extent, working as a smart disk drive attached to the workstation. Therefore, any workstation supporting the SCSI protocol might be able to use the SDM as an additional smart peripheral.

The remaining modules were in a custom card cage, linked to each other by a VME bus[5]. Note that the addresses are stored into the AM, while the contents are stored into the SM. Therefore, to maintain the association between the address and the content, a 13-bit tag is used. That tag is unique for each pair address/contents, thus meaning that the memory is able to carry $2^{13} = 8192$ different hard locations.

The Control Module was implemented on a single board, containing a Motorola MC68030 microprocessor. It is responsible for controlling both the Address Module and the Stack Module, as well as interfacing the Executive Module. Its function is mostly to receive requests from the EM, translate addresses into 13-bit tags and address the corresponding modules.

[5]The VME bus is a computer bus, standard ANSI/IEEE 1014-1987.

The SM is the one where the hard locations' contents are stored. It is made of several submodules, called "folds". Each "fold" is an array of 16-bit counters, one counter for each of the 256 bits, and an adder to quickly sum the vectors that are to be stored into the SDM.

The Address Module is custom-designed hardware. It had to be custom-designed because of the requirement of high-dimensionality of the SDM. In that case, the SDM was developed to work with 256 bits, and there was no commercial hardware for this number of bits at the time. The authors then designed the module and optimised it to compute the Hamming distance and decide what hard locations shall be activated.

The Stanford model was, in summary, a huge and powerful hardware system at the date. It is able to perform about 50 read/write operations per second, based on about 25 hits per operation. But the Stanford model is also an expensive and complex model, suitable only as an academic prototype. It is the proof of a concept, not a practical or commercially viable implementation of a memory system.

3.6.2 Weather forecasting

Weather forecasting is a high-dimensional problem, where a large number of variables has to be considered to make a single weather prediction. That characteristic makes the problem of weather forecasting a good one to be dealt with using a Sparse Distributed Memory. D. Rogers [87] has built such a system. Weather data used consisted of features such as date, air pressure, cloud cover and temperature, among others, sampled over a 20-year period on the Australian coast. Those sample features were encoded into 256-bit vectors. Each 16 bits of the vector represented a different feature. Hence, a maximum of 16 features was taken into account. The feature vectors were then fed into the SDM as addresses to store a single 1-bit datum. That bit denoted whether it rained in the next 4 hours or not. According to the theory, similar weather conditions would lead to similar patterns of features. Therefore, for any given input, the SDM would then be able to match similar patterns and retrieve an accurate prediction of whether it was likely to rain or not.

Rogers used a neural network to implement the SDM and a Genetic Algorithm (GA) to evolve the best weights, thus leading to the system he calls a "Genetic Memory" (GM). The GM is a three-layered neural network, just as Kanerva's original proposal. The main difference between Kanerva's proposal and Rogers' implementation lies in the way Rogers deals with the first layer's weights. Kanerva describes a model where the weights are fixed. The memory's hard locations are kept unchanged. Rogers, nonetheless, uses a GA to update them in function of the input data. The hard locations are, in consequence, moved around the space in order to best reflect the input vectors. According to the author, the GM exhibits a performance superior to the performance of the original version with fixed

weights for the input layer, producing up to 50% less errors. Indeed, randomly chosen hard locations may be appropriately spread over the binary space, but there is no guarantee that they will be placed in positions that will suit any application in particular. Updating their addresses and pushing them to where they best suit one's needs, may produce better results in particular applications.

Another advantage of adjusting the addresses in function of the input data is that it is possible to analyse their final positions, possibly grabbing extra information from their motion over time. Since in Rogers' application the addresses (input vectors) were compositions of the different independent variables, Rogers scans the space and identifies regions where it is more likely to rain or not. Then, the author infers that some values of some features are more likely to cause rain than others. Those conclusions would not be possible if the locations were kept unchanged, as in Kanerva's original model.

3.6.3 Object indexing

R. Rao and D. Ballard [81] used a SDM to index objects. Each object is represented by an iconic description and assigned a reference label. The iconic description comprises a set of photometric features, which are obtained taking the responses of derivative Gaussian filters at a range of orientations and scales. An object is described as a set of filter response vectors, from different loci within the object, for a small number of different views.

A modified version of Kanerva's SDM was used to facilitate interpolation between views and learning the associations between an object's appearance and its identity reference label. Rao and Ballard point out SDM's immunity to noise as a remarkable advantage. That is due to its tendency to orthogonality, explained in Section 3.2.2. Additionally, the fact that addressing needs not to be exact allows one to localise the closest object with just minimum information. According to the authors, that approach showed a good tolerance to occlusion, illumination changes, scale changes and rotations in three dimensional worlds.

Rao and Ballard's approach differs from Kanerva's proposal in various aspects. One aspect is that Rao and Ballard do not use binary addresses, but numbers whose range is determined by the output from the filters. Another difference is that they do not use the Hamming distance to compute the distance between memory items. They use the normalised dot product, as shown in Equation 3.14. In the equation, d is the distance, and r_1 and r_2 are two data items:

$$d(r_1, r_2) = \frac{r_1 \cdot r_2}{\|r_1\| \cdot \|r_2\|} \tag{3.14}$$

Finally, Rao and Ballard, like Rogers (whose work is summarised in Section 3.6.2) also do not use randomly placed hard locations. They pick addresses according to the distribution of the data. The

initial set of hard locations is selected in function of the training vectors. Once all the locations are filled, the space is re-organised using a soft competitive learning rule [54].

3.6.4 Jaeckel's selected-coordinate design

In 1989 L. Jaeckel proposed an interesting improvement to the original Sparse Distributed Memory [46], which is intended to reduce the addressing time at the cost of possibly reducing the maximum number of hard locations that can be represented. That cost is not a big disadvantage of Jaeckel's approach, since the memory has to be very sparse anyway. The idea is that, given the limited number of hard locations, there is no need to use all the bits to select the active locations. Instead, Jaeckel proposes that only 10 bits may be used, which he calls the *selected coordinates*.

According to Jaeckel's design, regardless of the number of bits in the address vector, a much smaller number of bits, possibly 10, are selected from the whole vector. Only those 10 *selected coordinates* are relevant to compute the set of active locations, and the remainder bits are ignored. Hard locations are activated if those 10 selected bits have the same values than the corresponding bits in the address vector, regardless of the values of the other unselected bits. Figure 3.8 shows an example, for 8-bit vectors. In the figure, the input 1001 1010 activates the second hard location, for it is activated by all the vectors in which the second bit is zero, and the fourth and seventh bits are one.

Depending on how the addressing algorithm is implemented, addressing in Jaeckel's model may be several times faster than in the original Kanerva's. There is no need to compute the Hamming distance between possibly lengthy vectors. All that is required is that the XOR of 10 selected bits is zero, and that is much faster to compute than the sum of a large number of bits.

Another consequence of using Jaeckel's design is that the overlap of the access circles is considerably increased. Using 10 selected coordinates it is only possible to define $2^{10} = 1024$ different access circles, regardless of the total number of bits. All the vectors which share the same values for those selected coordinates will overlap. If $N = 1000$, the access circle of each hard location will have 2^{990} points. Hence, that makes it possible to retrieve any vector from the memory, if only 10 bits are accurately known, providing those known bits are the *selected coordinates*.

Jaeckel also describes two possible hardware embodiments for the model, which are, as could be expected, much simpler than the Stanford prototype described in Section 3.6.1. The Stack Module and tag cache remain similar, but the addressing process is completely different in Jaeckel's design. Therefore, those modules contain one or more specialised address decoders which are responsible for selecting the active hard locations by comparing the 10 *selected coordinates*. The Executive Module is also similar to the one used in the Stanford prototype: a workstation running a program where the

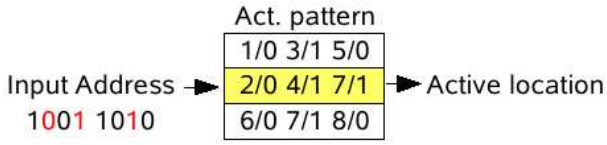


Figure 3.8: Jaeckel's design of a memory with 3 selected coordinates.

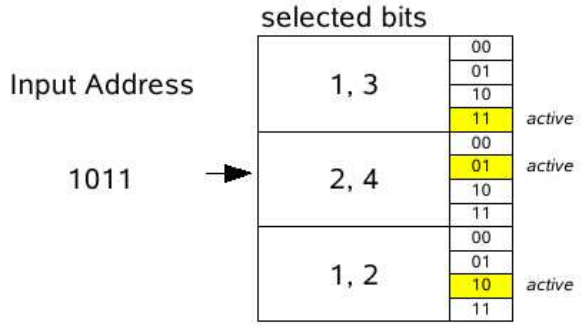


Figure 3.9: Karlsson's SDM design of a memory using three blocks and two selected coordinates.

user may control the memory parameters. In this case, however, the user can also define the selected coordinates.

The disadvantages of Jaeckel's model are also obvious. One is that the number of hard locations is limited to the number of different binary numbers that can be represented using only the selected coordinates. Another disadvantage is that the number of prediction errors may greatly increase, due to the large overlap of access circles. And yet another disadvantage is that the memory, though highly tolerant to errors in the non-selected coordinates, is very vulnerable and prone to failure due to possible errors in the selected bits. In summary, Jaeckel's design is a different approach, possibly better in particular domains than the original model. But it is not a general-purpose improvement to Kanerva's model, for it looses generality and robustness and requires yet another parameter to tune (selection of the *selected coordinates*).

Karlsson's design

Inspired by Jaeckel's design, R. Karlsson proposed yet another alternative design to Kanerva's original model [53]. Figure 3.9 shows Karlsson's model. Karlsson proposes that, starting with Jaeckel's design, each hard location is divided into A blocks. Each block contains 2^k hard locations, where k is the number of selected coordinates, and A the number of hard locations in the original Jaeckel's design. Karlsson points out two main advantages of the newly proposed model: 1) it is faster, compared to Jaeckel's approach; and 2) the number of activations in each access is fixed to A, which may reduce the probability of errors.

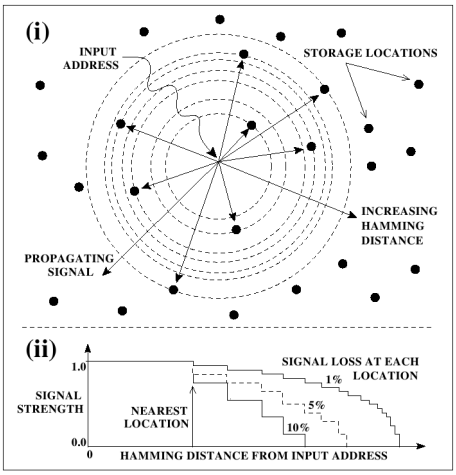


Figure 3.10: Hely's signal propagation model for the SDM. Source: [38].

Karlsson also proposes a rule to choose the selected coordinates: a bit cannot be chosen if there exists another bit which has been used less times. The goal behind the proposed rule is to get a number of activation patterns that is as random as possible.

Karlsson states that his model is much faster than the original Jaeckel's design, and exhibits a similar signal-to-noise ratio. However, it is not clear how much better Karlsson's model is for general purpose applications. Besides, the disadvantages of the original Jaeckel's design, outlined in Section 3.6.4, still hold for Karlsson's model.

3.6.5 Hely's signal model

In 1995 Tim Hely et al. proposed another variant of the SDM [38], based on the ideas of signal propagation. Addressing in Hely's model is inspired by the idea of a signal, which loses strength as it travels away from the input address. Figure 3.10 illustrates the concept.

To implement Hely's model, it is required that the arrays of bit counters are replaced by arrays of floating point numbers. For each write operation, all the arrays (all the hard locations in the memory) may be updated with a weighted value of the input pattern to be stored. The weight may vary between 1.0, for the closest hard location, and 0.0, for the farthest location. To read from the memory, the hard locations shall contribute proportionally to the final value, according to the weights used during writing: the closer the location is to the input address, the larger the weight of its contribution.

One clear advantage of Hely's model is that it will not need a sophisticated module to select the subset of hard locations. It can be implemented in a way that all the hard locations are automatically selected for each read or write operation, even if the signal strength at that point is very low.

Hely also proposes another improvement to the SDM: the hard locations which constantly receive low

signals should be pruned, so that in the long run only fit locations will remain. In the initial setup, the memory is loaded with excess hard locations. Those hard locations are created at points where signal is being received, and assigned addresses identical to the input pattern. After the initial setup, the excess hard locations are removed. The first locations to be removed are the ones which have received lower signal intensity.

Hely's model creates, therefore, a dynamic memory. The structure of the memory adjusts over the time to the data that is being stored. It is, therefore, suitable to work with non-random data, due to the use of a signal model to propagate the data, instead of an activation radius. The authors claim that the performance of such a model, tested with non-random data, is, in general, better than Kanerva's original proposal with the same data. However, it is not clear if the improvement in performance is worth the increased complexity of the system.

3.6.6 On-line value-based reinforcement learning

B. Ratitch and D. Precup [85, 84] propose the use of a SDM for on-line, value-based, Reinforcement Learning (RL). According to the authors, SDMs can be advantageous over other RL methods. They provide a linear, local function approximation scheme suitable to work in high dimensional spaces. Indeed, SDMs were designed to map a very large input space onto a much smaller physical memory, thus being suitable by design to deal with high-dimensionality.

Ratitch et al.'s implementation is not as proposed by Kanerva. The input and output vectors are arrays of floating point numbers, instead of plain binary vectors. The bit counters are also replaced by floating point numbers, as in Hely's design described in Section 3.6.5. The memory is, therefore, an implementation of a function that maps $f(x) : R^n \longrightarrow R^n$, to suit the particular needs of RL, and not an implementation of the original SDM: $f(x) : 2^n \longrightarrow 2^n$.

Another difference is the similarity measure. Since Ratitch et al. work with floating point numbers, they do not rely on counting bit differences to compute the distance between two points in space. Considering input vector $x =< x_1, ..., x_n >$ and hard location $h =< h_1, ..., h_n >$, similarity $\mu(h, x)$ is given as follows:

$$\mu(h,x) = \min_{i=1,...,n} \mu_i(h,x) \tag{3.15}$$

$$\mu_i(h,x) = \begin{cases} 1 - \frac{|x_i - h_i|}{\beta_i} & \text{if} \quad |x_i - h_i| \leqslant \beta_i \\ 0 & \text{otherwise} \end{cases} \tag{3.16}$$

where β_i is the activation radius. Similarity is, therefore, a measure of the *degree of activation* of the hard location being tested, normalised to fit in the continuous interval [0, 1].

Prediction (reading) from the memory, is the weighted mean value of all the active locations. Considering H_x the set of hard locations, μ^k the similarity between input x and the k^{th} location and w_k the actual value stored in the very k^{th} hard location, prediction is as follows:

$$\hat{f}(x) = \frac{\sum_{k \in H_x}(\mu^k w_k)}{\sum_{k \in H_x} \mu^k} \tag{3.17}$$

Learning is done using a standard gradient descent algorithm for linear approximation. The value being learnt is, therefore, *proportionally distributed* over the set of active hard locations. Considering the input sample $< x, f(x) >$, the update is performed as follows:

$$w_m = w_m + \alpha \frac{\mu^m}{\sum_{k \in H_x} \mu^k}[f(x) - \hat{f}(x)], \forall m \in H_x \tag{3.18}$$

where α is the learning rate.

Another difference between Ratitch's application and Kanerva's proposal is that Ratitch also does not use randomly placed hard locations. Instead, Ratitch proposes a *dynamic allocation algorithm*. The dynamic allocation algorithm starts with an empty memory. When there is a datum that needs to be stored in a region of the memory that is empty or too sparse, one or more hard locations are added as needed. Considering A the number of locations in the neighbourhood of x and M the minimum number of active hard locations for any datum, when x is to be stored into the memory and $A < M$, then $M - A$ hard locations need to be added. For better performance, the new locations must be evenly distributed in the neighbourhood of x. Therefore, for any pair of locations (h^i, h^j) in the neighbourhood, a condition on their similarity must be enforced:

$$\mu(h^i, h^j) \leq \begin{cases} 1 - \frac{1}{M-1} & , \quad if \quad M \geq 3 \\ 0.5 & , \quad if \quad M = 2 \end{cases} \tag{3.19}$$

Equation 3.19 guarantees that the fewer locations in a neighbourhood, the farther apart they should be placed. The authors propose two rules to add new locations, according to the *dynamic allocation algorithm*:

1. For datum x, if $A < M$, add a new location centred at x, if Equation 3.19 is not violated.

2. After applying rule 1, if $A < M$, add the necessary locations, sampling uniformely from the interval $[x - \beta, x + \beta]$, where β is the activation radius, enforcing Equation 3.19.

Once the memory has reached its maximum allowed number of hard locations and new locations need to be inserted, the authors propose an algorithm they call Randomised Reallocation Algorithm (RRL). Considering the new location x to be added, it is necessary to find location x', in the set of active locations, that is closest to x. The location x' is then removed, and a new location is inserted

midway between x and x'. If the datum is different from the address, the contents of the new location are also placed midway between the contents of x and x'. The procedure is repeated for each location that needs to be removed.

3.6.7 SDM using M-of-N codes

Furber et al [29, 9] implemented yet another variation of a Sparse Distributed Memory. Their implementation is based on the use of a Neural Network. What is new in Furber's proposal is the use of a code known as "M-of-N"[6]. In the M-of-N code, it is guaranteed that for every input or output there are exactly M active bits (or firing neurons), out of N possible bits. N is the total number of neurons in each layer. M is a user-defined number of firing neurons, in the interval [0, N[. For every read or write of a binary vector, the output or input vector must have exactly M active bits. The activation of M address decoders is guaranteed choosing a suitable activation threshold, and sorting the addresses to select the M top decoders.

The advantage of using an M-of-N code is that the whole system is even more immune to noise than Kanerva's original approach, or traditional Neural Networks. That additional immunity, though, has a cost. The first drawback is that additional processing is needed, to perform selection of the M firing neurons. The second is that the portion of usable addressing space is greatly reduced, for only combinations of M active locations are allowed. While the whole binary space comprises 2^N points, enforcing the use of M active bits one can only address $\binom{N}{M}$ positions—a relatively small sphere inside the original Kanerva's 2^n. Considering the space 2^n with $n = 1000$, for example, it contains about 10^{301} points. Enforcing the use of M=200, or 200 active bits, it is only possible to address 6.7×10^{215} points, a fraction of 10^{-86} of the space.

Another difference between Kanerva's proposal and Furber's memory is the use of simple binary weights instead of bit counters. Writing datum x to a location h is equivalent to updating h using a simple bitwise OR function:

$$h_i^{t+1} = h_i^t \vee x_i, \quad i \in 0, \ldots, n \tag{3.20}$$

Once one bit is set up to one, it can not be set down to zero again. Additionally, subsequent writing of the same value has no effect. That makes it possible to implement such a memory using pulsing neurons. Furber states that, according to numerical simulation, binary weights seem to perform better than up-down counters, in terms of the ratio of the number of bits of information stored, to the number

[6] In the literature it is possible to find different literals and namings for the code. Some authors refer the code with those characteristics as M-of-N (for example, [18] is an article by the same author), others refer N-of-M. Despite [29] and [9] using "N-of-M", "M-of-N" is used here, since that looks more according to Kanerva's literature. Kanerva uses N to refer to the whole space 2^n.

of bits in the storage locations—and they require less processing. Another advantage is that the memory always learns in a single pass, although at the possible cost of definitely mixing up any previously stored information. One drawback of that simplification is that there is no advantage in writing the same data more than once to emphasise information that is considered more important, or seen with a higher frequency (a characteristic of the human memory). Also, the original SDM memory and many other implementations (such as Ratitch's approach, described in Section 3.6.6), exhibit the characteristic of *natural forgetting* over time. Furber's model can not forget bits which have been set to "1", which is a major drawback. When saturation is reached, the memory will never recover.

3.6.8 Working with sequences

Sequence learning is a *natural* human ability which is not trivial to implement using traditional computer memories. Associative memories, though, have contributed to solve that problem. Using an associative memory it is possible to associate a given pattern (sequence element) with its successor, in the correct order. Following that procedure for each element of the sequence, it is possible to store the complete sequence from beginning to end. Later, starting from any element, it is possible to retrieve all the sequence iteratively from that element to the end. If the memory is versatile enough, it may also be possible to retrieve the sequence elements in reverse order. That also makes it possible to follow a sequence backwards, from any given point to the beginning.

Nonetheless, the method described above only works for simple cases. Sequence storing is not as trivial if it is necessary to store complex sequences, or multiple sequences at the same time. For example, sequences in which a given element occurs more than once may be a problem. Or if it is necessary to store different sequences sharing some common elements, that is also a problem that cannot be addressed using a simple associative memory.

A simple sequence, for instance $ABCD$, should be correctly stored and retrieved if input A is associated with output B, then input B is associated with output C, and so on. Any SDM, or other associative memory, should provide the necessary support for that task. If every single element of all the sequences was unique, and there was no mixing up of the data, there would be no ambiguous situations and all the sequences could be stored in the memory at the same time. However, that uniqueness cannot be guaranteed most of the times. It often happens that one element of the sequence occurs more than once, or that it is shared by two or more sequences. Considering a sequence of vectors where each unique vector is represented by a letter of the alphabet, under normal circumstances one can expect the random occurrence of sequences like these ones:

1. ABC

2. XBZ

3. DEFEG

There are two different problems with the three sequences above:

1. Sequences 1 and 2 both share one common element (B);

2. One element (E) occurs in two different positions of sequence 3.

In the first case, different sequences sharing common elements means that those sequences cannot be retrieved from the memory. If the input is A or X, the successor B is correctly retrieved. But if the input is B, there are two possible answers: C for the first sequence, and Z for the second one. In the basic version of the SDM, the output would be a mix of the two, since all the hard locations addressed by B will be activated during the write and read operations.

Concerning sequence 3, the problem is even more complicated. In that sequence, element E occurs twice. If the input is D or F, the output will be E, but if the input is E, there are two possible correct answers: either F or G. Again, in a normal version of the SDM the output is a mix of the two, which means an incorrect prediction in both cases.

k-folded memory

To solve the problem of ambiguities in sequences, Kanerva proposes the use of a k-folded memory. A k-folded memory is one in which the input is composed of the last k inputs. Figure 3.11 shows an example of such a memory. In that case it is a 3-folded memory, which stores the two last seen inputs. When a new datum D_k is present to the memory, it is juxtaposed to the data D_{k-1} and D_{k-2} to form a new vector. The input to the SDM is then the new vector. D_{k-2} is thereafter lost, replaced by D_{k-1}, which is in turn replaced by D_k, in time for the next iteration with a new datum D_{k+1}.

In the examples shown above, a 2-folded memory is sufficient to correctly predict all the sequences. For sequences 1 (ABC) and 2 (XBZ), the input to the SDM is, respectively, AB or XB. Therefore, the correct answer is C to the former and Z to the latter, which means the ambiguity does not arise in that case. As for sequence 3, input DE will predict F, and input FE will unequivocally refer to G.

The k-folding technique can be regarded as the use of a short-term memory, to store a certain amount of recent context. The robustness of the memory will depend a good deal on the number of *folds*, which is the same as the amount of history that can be remembered. The longer the memory can remember, the less mistakes it will make. Nonetheless, for a general-purpose memory it is impossible to determine

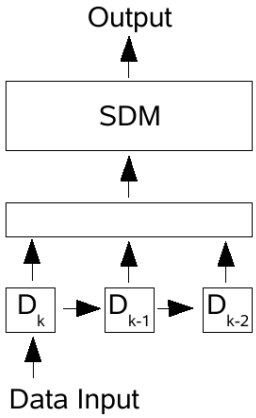


Figure 3.11: 3-folded memory.

exactly what is the appropriate number of folds that will never fail—there is always a probability, however small, that there will occur a sequence with more repeated elements than the number of folds in the memory. The use of folds will also result in a loss of flexibility. The higher the number of folds, the longer the memory will take to respond to sudden changes, as is the case of the kidnapped robot. Another disadvantage of this approach is that each fold requires additional length of the input vector. The SDM works well with high-dimensional vectors, and it is easily scaled up, but nonetheless it is necessary to account for that problem, as it may result in additional processing time and/or hardware requirements.

Sequence machines

B. Ans et al. [3] use two neural networks (NN) to build a *sequence machine*. One NN is used to store the sequences (NN1), the other is used to store a representation of the past behaviour (i.e., context) (NN2). When a new datum (address) arrives, NN2 is queried to retrieve the present context. That context is juxtaposed with the incoming address, and the resulting vector is then used to address the memory, either to store a new datum into NN1 or to retrieve the desired datum.

In a different approach, using a SDM, J. Bose [9, 11] points out at least four components for a sequence machine: input, output, main memory and context memory. The context memory will provide the main memory with context information, thus allowing to select the correct successor of the recurring symbol in function of the context where it happens. The input to the SDM is not the data one wants to store, but the composition of that data with the context, so that the association is made between the pair data+context and the desired output.

One way to represent the context is to use a shift register to store a fixed-length time window (as described for the k-folded memory). The disadvantage of such procedure is that the maximum number

of common symbols may be greater than the size of the shift register.

Another, more powerful solution, is to store the whole past behaviour of the sequence for every input, i.e.: $A \rightarrow B$, $AB \rightarrow B$, $ABB \rightarrow C$, $ABBC \rightarrow C$, ... That can be achieved, e.g., through the use of a neural network, used to learn the entire history of the sequence. The influence of the old context is modulated by multiplying the old context outputs by a constant and feeding them back as inputs, thus creating an infinite look back in time. A special character can be used to signal the end of a sequence, when the old context may be cleared in order to start a new sequence [10].

In summary, while the k-folding technique keeps track of only the last k inputs, the sequence machines by Ans and Bose store only a binary word that is a representation of all the past states. That way, the dimensionality of the input vector is drastically reduced, and the length of the time window can be stretched.

Remarks on storing sequences

Although the approach of adding folds to the memory, each fold representing a past situation, is adequate to deal with many real-world situations, the idea of using a representation of a "context" looks more powerful and may be more according to what really happens in the human brain. According to Burgess and Hitch [14], a variety of memory models converge to the idea that to store sequences of events the human brain associates an abstract "context signal" with each datum. That context signal evolves over time.

In a study cited by Burgess and Hitch, subjects were asked to memorise and recall two lists, of length m and n, respectively. In that situations, humans tend to make what is known as "intrusion errors". An intrusion error happens when items of the two sequences become intermixed, so that the person does remember items of sequence m in place of items of sequence n. The interesting point here is that the intrusive items in one list are usually from the same relative position in the other list. For example, if list m is $\{x, y, z\}$, x is more likely to be intrusive in the first position of the list n. That is consistent with the idea that each list item must be tagged with its relative position in the list, instead of associated to the previous or next items in the sequence.

Another interesting and well known characteristic is that it is easier for humans to remember sequences if they are presented in a rhythmic way [31]. Rhythm facilitates the process of memorising sequences, such as happens in songs or other musics. In that case, the elements of the sequence must be tagged with some temporal information, and cannot be simply chained to one another.

3.6.9 Image recognition

J. Vanhala et al. [101] show that a Sparse Distributed Memory can be used to store and retrieve images with good performance. Vanhala's work consisted in using the memory to store sequences of 100 images each. The images were 52 x 78 pixels, 256 graylevels. Therefore, each pixel should be represented by one 1-byte integer. The memory had 1024 storage addresses. The authors were mostly concerned about the presence of noise in the images, which is quite common and a problem very difficult to solve in most vision-based applications. To minimise the impact of noise, Vanhala truncated the pixels and used only the most significant bit of each byte. That is equivalent to work with binary images. The least significant bits, according to the authors, contain a "large relative proportion of noise". Therefore, their contribution to the result is irrelevant in the best case, and undesirable in the worst case. That means that each byte can be reduced to a single bit, thus making calculations much easier to carry.

Vanhala also proposes a method of *minimal reading*, which is supposed to improve the results by reducing the interference between the data written to the memory. In that method, each hard location is required to have a write counter, which is incremented every time it is written. When reading, the locations which have the lowest write count are preferred. A location with write count one should hold a perfect copy of the datum. Locations with higher write counts do not guarantee the fidelity of the data. That approach adds a little complexity to the system, but should improve the performance of the memory in the case of saturation.

3.7 SDM advantages and disadvantages

The SDM model exhibits some characteristics which make it look attractive to apply in many fields of research in general, and in mobile robots based on computer vision in particular. Namely, some characteristics can be summarised as follows:

- Many authors claim that the SDMs can be used in pattern (image) recognition, where they have shown to be naturally tolerant to occlusions, illumination changes, scale changes and rotations in 3D [81, 69, 71].
- SDMs are naturally immune to noise up to a high threshold. Using coding schemes such as m-of-n codes, their immunity is even increased [29], at the cost of reducing the addressable space.
- SDMs are very robust to failure of individual locations, just like neural networks.
- SDMs degrade gracefully, when some locations fail or the memory approaches its maximum capacity.

- One-shot learning is possible. If the memory is not close to saturation, it will learn in a single pass. That is also a desirable feature for a robot, where, in general, long learning periods should be avoided.

- Unlike traditional neural networks, SDMs can be "open" and subject to analysis of individual locations. That is specially important for debugging purposes, or to track the learning process.

- The memory is scalable. At any point in time, new hard locations can be added or removed without interfering with the other hard locations which may contain useful data.

- It is possible to change a memory's structure without retraining all the memory [85]—an important characteristic to build modular robots.

- Controlling the access radius, it is possible to find the *closest* match to any given input. That makes it possible to make even remote connections between items such as objects, locations or events.

- The SDM is suitable to work with sequences, which is in many aspects as the human brain probably works.

- Under normal operation it is only possible to retrieve a sequence in the order it was stored, not in reverse order. Nonetheless, it is possible to invert the process and, using data as addresses, retrieve the reverse sequence. That characteristic allows a robot to learn one path or task and be able to follow or perform it from the beginning to the end or from the end to the beginning.

- The SDM is a versatile system. It is suitable to work with high dimensional vectors, which can be any type of sensorial reading: images, sonar data and laser range finder outputs, among others, can be used.

- In a SDM it is possible to segment the addressable space, in order to hasten the process of addressing or decrease the probability of prediction errors [46]. That is an important advantage of SDMs over more traditional neural network architectures, in which the addressable space is unified in a way that it is impossible or very difficult to segment.

The SDM also has some drawbacks. The most important drawbacks are:

- Once a datum is written, it cannot be erased, only *forgotten* as the time goes by. Under certain circumstances that may be an undesirable feature, because unnecessary data may interfere with more recent and important data. However, the model can be modified so that selected hard locations may be removed, thus freeing space for new empty locations.

- Storage capacity may be only about 0.1 bits per bit of traditional computer memory.
- If implemented (simulated) in software, a lot of computer processing is required to run the memory alone. In hardware specially designed to implement the memory, however, the problem may be minimised or overcome.

3.8 Summary

This chapter describes the Sparse Distributed Memory, proposed by Pentti Kanerva in the 1980s. Kanerva proposed a sound theory and offered the mathematical support to prove it, as well as the implementation details. The SDM is based on the properties of a high-dimensional boolean space, which are to a great extent those exhibited by the human brain. The mathematical details were summarised in Section 3.2. The architecture of a working SDM was described in Section 3.3.

Many researchers have implemented working systems based on the SDM, and the results were those predicted by the theory. Some of them offered contributions that may improve the performance of the original model, with only marginal costs. Relevant works were described in Section 3.6. The SDM is, therefore, a robust model of an associative memory, whose characteristics, described in Section 3.5, make it suitable to be put at the helm of a mobile robot.

Chapter 4

Robots navigation

If we knew what it was we were doing, it would not be called research, would it?
(Albert Einstein, German physicist)

Contents

One of the goals of the present work is to navigate a robot using sequences of views stored into a Sparse Distributed Memory. There are many other navigation techniques, and there are also many robot architectures that can be implemented. This chapter briefly reviews some system architectures and robot navigation methods, as well as previous work on robot navigation using Sparse Distributed Memories.

4.1 Brooks' subsumption architecture

The control system of earliest robots were based on a pipeline architecture, which is neither robust nor easy to scale for more complex systems. Brooks proposed a "subsumption architecture," which consists of a hierarchy of levels of competence. That architecture is more robust and easy to scale than the pipeline model.

4.1.1 Architecture of competences

The architecture of the earliest robots was based on simple task-oriented control systems. Those control systems had little hierarchical decomposition, if they had any hierarchy at all. They were strongly based on a pipeline architecture, such as the one shown in Figure 4.1. In those systems, the operation is sequential. Each step that is performed after another step builds on information and results gathered in previous executional steps. Data is collected from the sensors, processed in the intermediate stages, and motor commands are generated in the final stage. In such an architecture every module of the system strongly depends on the adjacent modules. The failure of a single module or component may seriously affect the whole structure and even result in a complete failure of the system. The coupling is very strong, so every individual module has little autonomy to produce useful results in case of failure of any other. The consequence is that the system as a whole turns out to be very vulnerable.

In an attempt to improve the traditional robot control architecture, in 1985/6 Brooks [12] proposed an alternative design. The idea is to design the system in a hierarchical manner. The control tasks are divided in well defined levels of competence, as shown in Figure 4.2. Each competence is implemented in an independent layer. Those layers are then juxtaposed, one on top of another. The main point here is that the lower layers can operate independently. Every layer can work even if an upper layer fails. However, upper layers are superimposed on the lower layers and can subsume their functions. That means that a robot whose top layer fails, for example, may still keep all the other competences

Figure 4.1: Traditional robot control system.

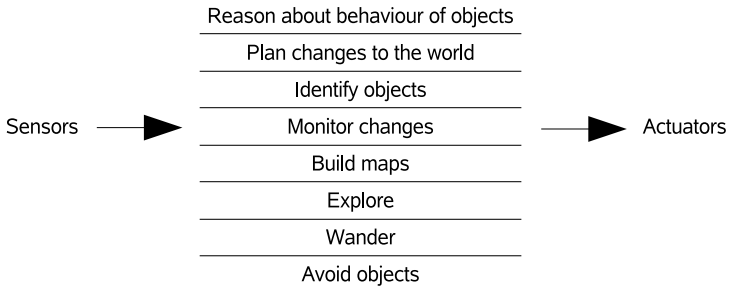


Figure 4.2: Brooks levels of competences.

untouched and remain operational, although with less intelligent behaviour or some abilities impaired. Once the malfunction is fixed, the previously faulty layer can become active again and superimpose its orders to the layers below. The overall system with all the layers working correctly shall behave more intelligently than another similar system with one or more faulty layers. However, the failure of one layer (except for the lowest one), should not compromise all the system, as happens in the pipeline architecture.

The subsumption architecture has obvious advantages over the traditional pipeline approach. Brooks summarises its main characteristics:

- **Multiple goals**—individual layers can pursue individual goals concurrently. In a pipeline architecture, coupling makes that behaviour very difficult. But in a subsumption architecture, it is possible to have different goals for each level of competence. The goals of the higher levels superimpose to those of the lower levels in case of conflict.

- **Multiple sensors**—sensorial readings only need to be fed to the layers where they are relevant, eventually avoiding the need to sensor fusing. For example, collision sensors may only be monitored by the first layer during normal operation.

- **Robustness**—as stated above, in case a higher level layer cannot produce results in time, the lower layers will still be able to keep running and produce correct results, although at a lower level of competence.

- **Additivity**—the layers can be built and debugged incrementally, and even run on different processors. According to the author, bandwidth requirements on communication channels between

layers are relatively low.

4.1.2 A critical view

Brooks' subsumption architecture has already been largely debated and used for several years. It is a quite remarkable and important approach. When the model was proposed, there has been a lot of interest in it. Nonetheless, in practice it has been very difficult to implement such a model, at least exactly according to the original idea. The theoretical principle is very appealing, but from a practical point of view there are some problems which need to be dealt with.

One problem is that the model has too many layers. The division, implementation and coordination of all the competences shows many difficulties that arise from the complexity of the overall system. The consequence is that the architecture, although appealing, is not practical. It is too finely divided to be of use for the most common applications. Another problem is that in case of failure of a top layer, there is no guarantee that the error will not propagate to the lower layers. The architecture is only tolerant to errors that are detected, and errors from the top layers will only not propagate to the bottom (or not cause erroneous behaviour/output) if the failure causes the faulty layer to suppress its output or there is a critic module that detects the problem and neutralises the faulty module's output.

Apart the practical difficulties of implementing the complete architecture, the idea of building a robot layer by layer, in a hierarchical structure, is still appealing. To some degree, the hierarchal model mimics what happens in the human brain, as described in Section 2.2.2. A robot with two major layers would be the closest, in terms of hierarchical structure, to the human brain. The lower structure should model the old brain, which controls basic vital functions such as breathing, equilibrium or instincts. It should, therefore, control the basic behaviours of the robot, such as collision avoidance, battery charge or equilibrium if the robot is dynamically stable. The higher layer would mimic the neocortex, which is responsible for the intelligent behaviour and high-level planning activities of the system. Trajectory planning and communication, for example, must be competences of the top level.

Analysing Brooks' proposal under the light of the theories on the brain and intelligence, discussed in chapters 2 and 3, again it is clear that there are many differences and difficulties in concealing all Brooks, Kanerva and Hawkins' proposals. Hawkins and Kanerva, among many other researchers, agree that an associative memory is the source of intelligence the way humans conceive it. Artificial intelligent behaviour will surely be different from natural intelligent behaviour, but experts seem to agree that it will probably be based on a large memory system with an associative memory at its core. On the other hand, a hierarchical, layered architecture, such as Brooks proposes, would be very difficult to implement based on such a memory. Each layer can have its memory, if needed, but there is too much

partitioning of competences and rigidness of the system for such an architecture[1]. Behaviours such as the abstraction of a concept or pattern association are not naturally implemented. In summary, the subsumption architecture is still an interesting idea. But perhaps, from the practical point of view, it has to be redesigned and simplified for being used in intelligent memory-based robots.

4.2 Mapping and navigation paradigms

The problem of robot navigation is often decomposed in three different, more specific, tasks. The first is the problem of localisation: the robot must be able to identify its location. The second is the problem of mapping: the robot needs to build some representation of the environment. The third is the problem of path planning: the robot must find a path from its current location to the desired position, through the environment, and that path must be feasible and safe. Those three problems are intrinsically related, and the approach to address one strongly influences the options available to address the others.

Some popular approaches to navigate robots are the use of Voronoi Diagrams and Potential Fields methods, both described in the following subsections. As for mapping the environment, it is a common approach to build geometric models, and then plan trajectories based on those models. However, humans do not need exact models. Humans often rely on sequences of remembered images, and usually use maps only for long distances or unknown areas. Navigating through the use of view sequences is not as common as those major approaches, but some research has already been done onto the area.

4.2.1 Occupancy grids

The use of occupancy grids as a mapping strategy is a popular approach. The method was proposed in 1989, by A. Elfes [23] and it is very straightforward. The robot's world is represented by a matrix, such as the one shown in Figure 4.3. The Figure shows a world divided in 100 squares. The robot is in square 4H, and the goal is to reach square 6A. The black squares represent obstacles that the robot cannot pass through. Therefore, the robot is expected to avoid those obstacles, planning a path through blank squares only. An example of a possible path is marked in the figure in gray.

In practical terms, an occupancy grid can be represented by a matrix of floating point numbers. Each element of the matrix stores a probability that the corresponding square in the world is occupied by some obstacle that can prevent the robot from passing through it without some damage to the robot or to the very obstacle. The matrix can start empty, with all the elements at zero. It is assumed, therefore, that the all the represented positions in the world are available and can be visited by the

[1] S. Jockel proposes a multi-SDM model that in some aspects resembles such an architecture [47].

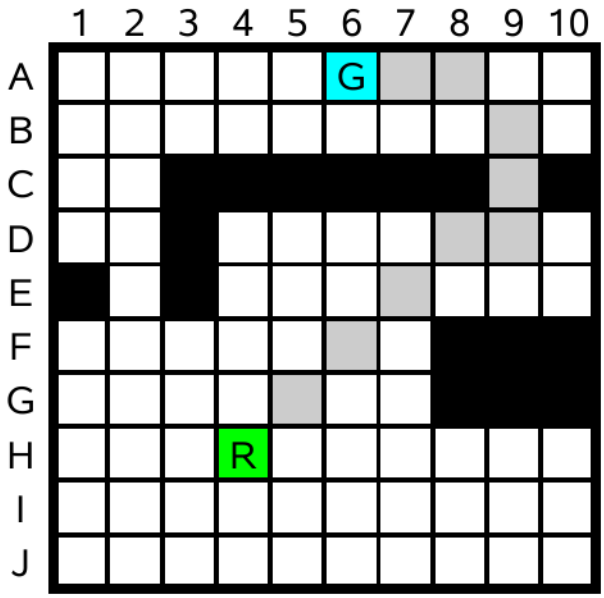


Figure 4.3: Example of an occupancy grid. "R" is the robot, "G" is the goal.

robot. Then the robot explores the world using distance sensors, such as sonars or laser range finders. As those sensors encounter obstacles, the matrix is updated accordingly and a more accurate map is built incrementally.

For easier robot localisation, it is possible to store additional view information in a separate data structure. That additional information can be indexed to the matrix coordinates, so that it is always possible to match it with the correct map location, if the robot's localisation algorithms work correctly.

The occupancy grid technique is very simple to implement and to use, and it is a very robust framework. However, it also has important drawbacks. One important drawback is the way it deals with dimensions. It is necessary to define how much of the real world is represented by a number of the matrix. A fine grid is able to represent more details of the world, but requires large amounts of memory. A coarse grid saves memory, but looses details of the environment. It is, therefore, necessary to find a satisfactory compromise, between grid resolution and memory requirements. Another problem is that of the boundaries of the map. For robots that operate in a small space, the occupancy grid is an appropriate solution. But for large environments, or environments with undefined boundaries, it is a problem to choose the appropriate resolution and size of the matrix. Yet another problem is that of the origin. If a robot has to explore an unknown area, it will have to start placing the information in the occupancy grid somewhere. However, if it has no *a priori* information about its location, it is difficult to choose the right starting point. The problem can be minimised if the size or dimensions of the matrix can be adjusted in runtime. However, that strategy may be very time consumming to do in running time, for large occupancy grids.

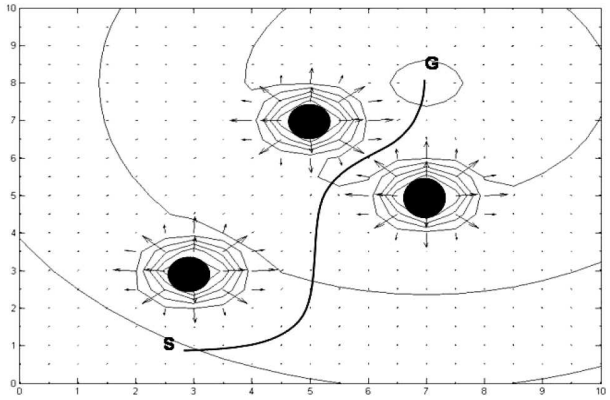


Figure 4.4: Example of a robot being guided by a potential field. "G" is the goal and "S" is the starting point.

4.2.2 Potential fields

The Potential Field (PF) method is a grid-based approach for robot navigation. The theory was proposed for the first time by O. Khatib [55], mostly for the motion of a robotic arm. But since then the same method has been extensively used to navigate mobile robots.

The idea behind the PF method is that the robot moves in a virtual potential field. The goal and the objects in the environment all create potential fields around them. The robot is sensitive to those virtual potential fields, thus being subject to virtual forces caused by their influence. Those virtual forces will push the robot away from obstacles and pull it towards the goal. Figure 4.4 illustrates the idea. In the figure, the robot starts moving at the starting point "S", attracted by the goal "G". However, the obstacles exert a repulsive force on it. That causes the robot to move away from the obstacles and avoid collisions with them. Under that conditions, the robot follows a path that is not the shortest safe path from S to G. However, the path followed should be the safest, in the sense that it maximises the distances between the robot and the obstacles. In theory, it is as if the robot is moving in a grid, driven by a force $\vec{F}$, that is the sum of one attractive and many repulsive forces [102]:

$$\vec{F}\,(x,y) = \vec{F_G}\,(x,y) + \sum_i \vec{F_{O_i}}\,(x,y) \tag{4.1}$$

The force $\vec{F_G}$ is attractive, pulling the robot to the goal. The forces $\vec{F_O}$ are repulsive, pushing the robot away from the obstacles. The equation represents a case where there is only one goal. If there are multiples goals, there will also be multiple forces F_{G_j}, all attractive, pulling the robot in possibly different directions. Those forces are proportional to the gradient of the virtual potential fields caused by the goal and the objects:

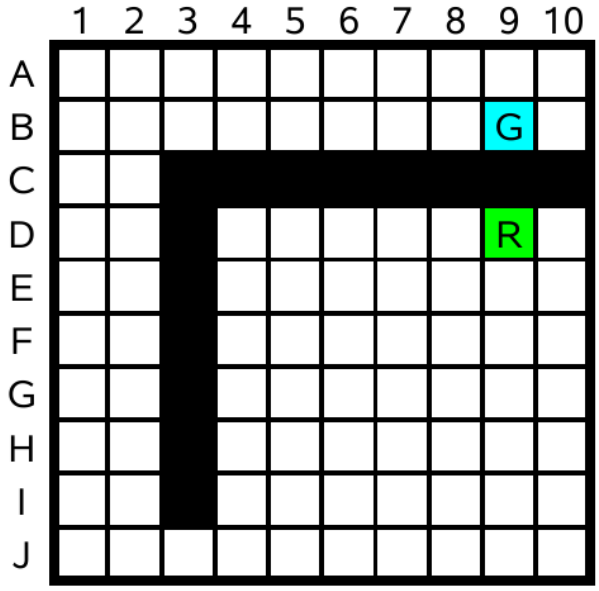


Figure 4.5: Robot trapped in a potential field minimum. "G" is the goal.

$$\vec{F_k}(x,y) = -\nabla U_k(x,y) \tag{4.2}$$

In the equation, U_k is the virtual potential field caused by object k at point (x, y). Since the intensity of the potential field decreases as the distance from the source of origin increases, the influence of the farther objects is smaller than the influence of the closer ones. Since the force applied to the robot is proportional to the intensity of the virtual potential field, the robot will be more attracted to the goals that are closer than to the goals that are farther away.

In practice, the potential field method can be implemented over a data structure similar to the occupancy grid. In the grid, each cell contains information about the intensity of the force that should be applied to the robot. The robot then follows from one cell to another, as if pushed by the virtual forces it is being subject to.

The PF method, albeit simple, is not very robust, as pointed out by Y. Koren [57]. One problem is that the robot gets trapped by local minima very easily. Figure 4.5 shows an example of a situation where the robot is trapped in a local minimum. In the figure, the robot is very close to the goal, so the attractive force is very large. However, to reach the goal it has to follow through cell 2J, where the virtual potential field is much lower than it is in its current position. A similar problem occurs when there are two closely space obstacles, or very narrow passages. However, the PF method is very popular, and may produce good results in simple environments. In complicated environments, such as mazes, however, it is hardly appropriated.

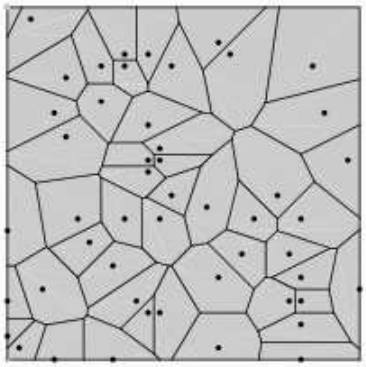


Figure 4.6: Voronoi diagram.[2]

4.2.3 Voronoi diagrams

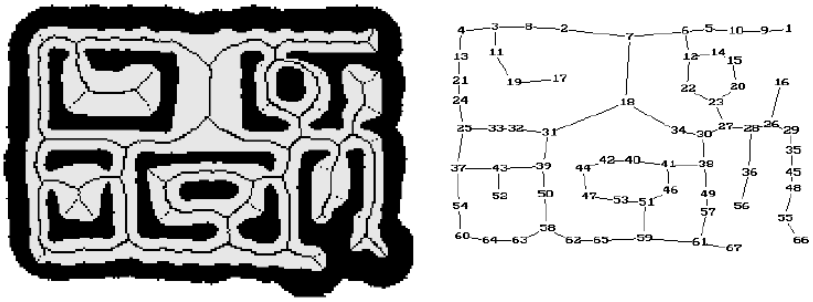
A Voronoi Diagram (VD) is a geometric structure which represents distance information of a set of points or objects. The basic structure was considered as early at 1644 by René Descartes, as a form of representing the solar system [28]. They were later used by Dirichlet, in 1850, in the investigation of positive quadratic forms. That is the reason why VDs are also referred, in the literature, as Dirichlet tessellations. Later, in 1908, Voronoi did further research and extended the original model [104]. Since then, it became more popular. Currently, Voronoi diagrams are extensively used in many different areas of application, such as computer graphics and robot navigation.

In a VD, each point is equidistant to two or more sites. The diagram is, therefore, divided in cells, so that in each cell the points are closer to a given point than to any other point. Figure 4.6 shows an example of a VD, illustrating the concept.

Voronoi diagrams are an interesting model for robot navigation. Each object is assigned its own cell, and the lines between the cells represent robot paths. If the robot follows the lines between the cells, it will choose the safest path to avoid collisions with the objects. While moving, the robot must sense the objects and walls and incrementally create the diagram. Its goal is to always be in the diagram lines.

The Voronoi diagram model solves the problem of local minima that trap the robots in the potential fields. In that aspect, it is an improvement. However, it is still a grid-based approach. Hence, it suffers from all the drawbacks of the grid-based methods, pointed out in Section 4.2.1. Moreover, the paths in a Voronoi diagram are still not the shortest possible. They are the safest, since the robot keeps the maximum possible distance from all the objects in the scene. However, there may be other paths which are shorter and still safe for the robot.

[2]Source: http://mathworld.wolfram.com/VoronoiDiagram.html (last checked 2009.12.12).



(a) Voronoi diagram. (b) Topological map.

Figure 4.7: Voronoi diagram and topological map of a region[3].

4.2.4 Topological maps

Grid-based methods are easy to implement, for they can easily be modelled using a two-dimensional array. However, they require a lot of memory, proportional to the size of the space and the resolution of the representation. Therefore, scalability is a problem for those models. Additionally, path planning might be very time consumming, because every cell of the grid might have to be processed to plan a single path.

To overcome the difficulties of dealing with grid-based methods, another paradigm has been suggested: the use of topological maps. Topological maps overcome many of the drawbacks of the grid-based methods. They consist of more sophisticated representations, which neglect details of the environment. The robot builds a graph of the routes available for the robot to travel. Then, algorithms such as Dijkstra [21] or A* [36] can be used to compute the possible paths and possibly choose the best alternative. Using those models, there is no need to record information of all the path that the robot follows. Instead, the robot only stores information about the points at which decisions are made [58]. Figure 4.7 shows an example of a Voronoi diagram and a topological map of a region. While the Voronoi diagram is represented over a grid, the topological map is just a set of points. Each point is connected to one or more points through a line segment. To represent an environment, it is only necessary to maintain a data structure of the feature points and the connections between them, which requires much less memory and processing power than the storage and use of the whole grid.

The topological paradigm was proposed for the first time in 1988 by Kuipers and Levitt [58]. Actually, the authors propose a complete semantic hierarchy of the descriptions of the interaction with the environment, consisting of four hierarchical levels:

1. **Sensorimotor** — That is the lower level of the hierarchy, represented by the raw sensor input and actuator output data.

[3]Based on [97].

2. **Procedural** — Those are procedures the robot learns to accomplish particular instances of place finding or route following tasks. Those procedures are acquired in terms of sensorimotor primitives, such as control strategies to cross a boundary defined by two landmarks. The basic elements of procedural behaviours are represented in a production-like schema, $< goal, situation, action, result >$. The interpretation of the production is "When attempting to reach goal *goal*, if the current view is *situation*, perform action *action* and expect the result to be *result*. At this level, the robot is able to perform basic navigation tasks and path planning.

3. **Topological** — While at the procedural level the environment is described in terms of the egocentric sensorimotor experience of the robot, at the topological level the world is described in terms of relations between other entities. A description of the environment is possible in terms of fixed entities, such as places, paths and landmarks, linked by topological relations, such as connectivity, containment and order. At that level, it is possible to perform navigation tasks and route planning more efficiently than at the procedural level. However, there is no information on the cost of alternative paths. In topological terms, it is only possible to describe actions such as "turn" or "travel", but there is no associated description of the magnitude required for the actions.

4. **Metric** — That level adds to the topological level metric information, such as distance and angle information. At the metric level, the robot has all the information it needs to plan routes and choose the quickest or shortest alternative paths. It is possible to compute accurately the length of a path, as well as the number and magnitude of the turns that the robot has to perform.

Kuipers proposes three models which are based on the idea of a four-levels semantic hierarchy: the Tour model, the Qualnav simulator and the NX Robot. All those models have been tested in software simulation [58], to validate the four-levels hierarchy described.

The Tour model is a computational model of a robot more suitable for environments that have approximate network-like structures, such as urban environments or the interior of large buildings with corridors and walls. The Tour model robot is expected to explore the world and find places and paths that can be followed, in order to build a topological map with metric information of the environment. At the sensorimotor level, Tour has available a sequence of views and actions. At the procedural level, it has procedures built from the sensorimotor schemas. Therefore, Tour is expected to grab a sequence of views $V_0, ...V_n$, and for each view V_j, it must perform action A_j. At the topological level, Tour builds place-path networks, boundary relations, regions and skeletal networks. At the metric level, Tour keeps information about local geometry of places and paths. That confers on the robot the ability to perform actions where the action can be $Travel(\delta)$ to move forward or backward, and $Turn(\alpha)$ to change direction, for example.

The Qualnav model is more suitable for navigation in open, possibly mountainous terrain. The Qualnav model implements a search strategy in order to spot landmark points that can be used as reference points to build a topological graph of the environment. Robot localisation and navigation is then implemented based on techniques such as stereo vision and triangulation.

The NX Robot is another extension of the Tour model, simulating a real robot with range sensors and an absolute compass.

Kuipers simulated robots obtain the topological models without great difficulty, in the simulation environment. In a virtual environment, sensorimotor readings can be assumed to be correct and as precise as wanted. Kuipers used those readings to feed the topological level of the semantic hierarchy and build the topological representation of the environment. In the real world, however, sensorial readings are subject to a multitude of interferences and imprecisions that make the task of building a topological map much more difficult than in a controlled environment. Variable amounts of noise from the sensors, scenario changes and other phenomena interfere with the performance of the system and make the mapping task a very challenging job. The most difficult task is to detect the nodes in the graph. It requires a match of two points or line segments, when the robot comes from two different directions. That problem is quite complicated, because it requires either very accurate metrical information or very precise pattern matching.

Many authors implemented topological navigation, based on Kuipers' simulated models, although the practical problems have to be addressed to achieve a robust navigation model. Often, topological navigation is combined with other navigation models, as do S. Thrun and A. Bucken [97]. The topological maps are excellent for high level route planning. But for local navigation, a data structure that contains more detailed information is more appropriate. Hence, a combination of two or more map models facilitates the planning and navigation process.

4.3 View-based navigation

Humans rely on vision for immediate navigation, and on topological representations for long trip planning. However, computer vision requires a lot of processing power and memory. Hence, other sensorial inputs have been preferred by researchers, instead of vision, for tasks which require real time processing. Nonetheless, as the computer processors get more powerful and the computer memory gets cheaper, the interest in vision-based approaches increases. This section briefly reviews the paradigm and some selected approaches.

4.3.1 The view-based paradigm

Many approaches for vision-based navigation are teach-and-follow approaches. The robot is first taught a path, guided by a human supervisor. While guided, the robot captures images at regular intervals. The relevant images are stored into the hard disk, along with other navigation information, such as identification of the sequence, number of the image in the sequence and motion that the robot was executing when it captured the image. The stored information can be used later, for autonomous navigation. In the autonomous runs, the robot is able to follow the same path by capturing an image, retrieving the closest image from its memory and executing the same motion that is associated with the stored image (motion required to go to the place where that image was taken). Repeating the procedure iteratively, the robot is able to reproduce the original path.

Compared to techniques that use other sensors to build maps of the environment, view-based approaches have some advantages, but also many disadvantages. Vision is among the most powerful sensing technologies. A single camera is usually very cheap and provides detailed information of a wide region of the scenario, in the form of a matrix of pixels. Other sensors only provide information about a narrow area or a specific point, in the form of a number or a vector. That is the case with distance meters, such as sonars, infrared or laser range finders. However, information from those sensors is also much more straightforward to process. It is much faster to process information from a small array of infrared sensors, which usually comprises just tens of numbers, than one single image, which usually contains hundreds or thousands of pixels.

To take advantage of both the approaches, many authors have built robots that follow hybrid techniques, such as using vision for localisation and distance sensors for collision avoidance and mapping. Additional sensors provide additional information that has to be processed in real time, and the robots usually have to develop internal models of the world based on that sensorial information. Multiple sensors add increased complexity to the system, while vision alone can provide the same or more information, using appropriate computer vision techniques. But the simplification of the hardware adds complexity to the analysis, since a sophisticated and time-consuming processing of the images is required.

"Teach and follow" based on vision is a much simpler and biologically inspired approach for basic navigation purposes than other autonomous navigation techniques. The main disadvantage is that it requires a supervised learning stage, although it is also possible to implement "wander" or "explore" behaviours solely based on vision. Additionally, to implement vision-based navigation images have to be grabbed and stored at regular intervals during learning, and matched against a large database of images during the autonomous run. Good resolution images take a lot of storage space and processing

time. To overcome that problem, some authors choose to extract some features from the images, in order to minimise processing needs in real time.

Another possible approach for navigation based on vision is that of using landmarks. Those landmarks may be artificially placed on the scene (a barcode or data matrix, for example), or selected from the actual images or the real scene in runtime. Selecting a few landmarks which are easy to identify, the navigation task is facilitated. C. Rasmussen and G. Hager [83] follow such an approach. They store panoramic images at regular intervals during learning, and select salient *markers* from them. *Markers* are characteristics considered invariant over time, such as colour, shape or texture. By sticking to those *markers*, the system saves considerable processing power, for there is no need to process the whole image to detect the best match.

One huge disadvantage of the vision-base approach, besides the computational and memory requirements, is that it is a very sensitive technique. Vision is sensitive to illumination, scenario changes and colour degradation, among other problems. Other sensing techniques are usually more immune to those problems. Therefore, a vision-based navigation algorithm must be robust enough to handle those difficulties. Humans perform view-based navigation in a highly efficient manner. The brain filters out illumination and scenario changes, as well as other environmental changes and problems, in a very natural and efficient way. But how the brain does it is still a *mystery*. Hence, it is still very difficult to mimic the behaviour, despite the fairly great advances in the area.

4.3.2 Matsumoto's approach

Y. Matsumoto et al. propose a vision-based approach [64, 65] for robot navigation, based on the concept of a "view-sequence" and a look-up table filled with controlling commands. Their approach requires a learning stage, during which the robot must be guided by a human supervisor. While being guided, the robot memorises a sequence of views automatically. Those views are pictures taken at regular intervals. While autonomously running, the robot performs automatic localisation and obstacle detection, taking action in real-time.

Localisation is calculated based on the similarity of two views: one stored during the learning stage and another one grabbed in real-time. The robot tries to find matching areas between those two images, and calculates the distance between them in order to infer *how far* it is from the correct path. That distance is then used to extract stored motor controls from a table, thus guiding the robot through the same path it has been taught before.

To calculate the drift in each moment, Matsumoto uses a block matching process, as illustrated in Figure 4.8. A search window taken from the current view is matched against an equivalent window

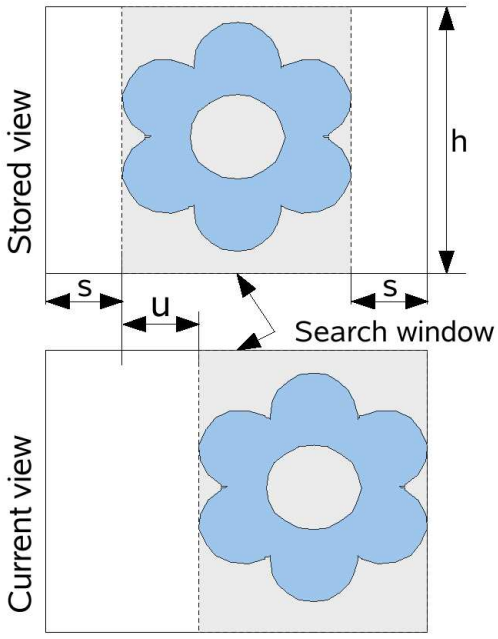


Figure 4.8: Matching of two images, using a search window.

in the memorised image, calculating the horizontal displacement that results in the smallest error e. Considering two images I_1 and I_2, Matsumoto defines the matching error between them as:

$$e(u) = \sum_{x=s}^{w-s} \sum_{y=0}^{h} |I_1(x,y) - I_2(x+u,y)| \tag{4.3}$$

In the formula, w and h are, respectively, the width and height of the image, in pixels. The letter u represents the offset, or horizontal displacement of the search window in the image. The search window has width $w - 2s$ and height h. The value $I_i(x,y)$ is the brightness intensity of pixel (x,y) of image i. And the error that is matters to measure the horizontal drift is the minimum $e(u)$, defined as follows:

$$e = min(e(u)), -s \leq u < s \tag{4.4}$$

If the robot is following a previously learnt image sequence, its view in each moment must match some stored image. If there is a horizontal drift, then e will be minimum for some horizontal displacement $u \neq 0$. Equation 4.3 is just the sum or the errors between the pixels in the image, for some value u of the horizontal search range. Equation 4.4 allows to determine the horizontal displacement u that gives the minimum error e. The approach is not as robust as stereo matching techniques, or other block matching methods that can be found in the literature. However, it is a very straightforward and quick approach for being implemented in real time. It only considers horizontal displacements, but that should not be a problem. The camera is fixed to the robot, so that significant vertical movements are not expected.

Matsumoto uses 32×32 pixels images, a search window 16×32 pixels and a horizontal search range

of 8 pixels, so that $-8 \leq u < 8$. The original images, captured by the robot, are 480×480 pixels, but the author claims that there is no significant loss of essential information when the images are subsampled down to the 32×32 pixels thumbnails that are being used.

In a later stage of the same work [66], Matsumoto equipped the robot with an omnidirectional camera. Omnidirectional images represent all the surroundings of the robot. That is achieved through the use of a hyperbolic mirror, which is placed above the robot and reflects the scenario 360^o wide onto a conventional camera. Omniviews are neither easy to recognise for humans, who have a narrow field of view, nor to process using conventional image processing algorithms. However, they allow the robots to overcome the limitation of the narrow field of view of traditional cameras. One camera can be used to sense all the directions at the same time. Forward and backward motion and collision avoidance may be planned using the same single sensor with the same accuracy in all the directions. To facilitate image processing, the omniview image can then be transformed into a cylindrical image that represents the surroundings of the robot. That cylindrical image can then be processed as if it was a common image, although it is necessary to pay attention to the fact that it represents a field of view of 360^o and there are some perspective distortions. Its left and right borders are actually images of the back of the robot, while the middle of the image represents its front. Additionally, due to the way the projective geometry works, it is difficult to obtain an omniview proportionally correct—a characteristic that introduces an additional difficulty for accurate robot localisation.

Matsumoto extended the navigation algorithms and sequence representations. Junctions, such as corridor intersections, were detected based on the optical flow differences. Assuming the optical flow is just caused by the motion of the robot, closer objects generate larger optical flows in the image. That simple principle makes it possible to detect junctions and construct a topological map of the building. The images were stored with additional information of neighbouring images, so that it was possible to follow a corridor and, at the appropriate junction, skip to another corridor. Route planning was done using the Dijkstra algorithm [21], so that the shortest path was followed. Matsumoto's approach, however, has the disadvantage of being too computationally intensive. Also, it is only suitable for indoors navigation, where the illumination is not expected to suffer large changes under normal conditions. The author does not report the implementation of any strategy to deal with illumination or scenario changes.

4.3.3 Ishiguro's approach

Ishiguro and Tsuji [45] also implemented vision-based robot navigation. However, the approach used is slightly different from that followed by Matsumoto. Ishiguro uses omni-directional views, as Matsumoto did in the later stage of the work. But in order to speed up processing, the images are replaced by their Fourier transforms. The omni-directional views are periodic along the azimuth axis. Therefore,

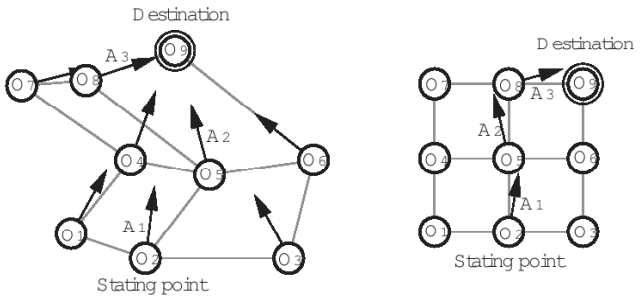


Figure 4.9: Geometry of a recovered and real environment, used by Ishiguro.

Ishiguro applies the Fourier transform to each row of an image, and the image is then replaced by the set of magnitude and phase Fourier coefficients. The technique considerably reduces both the memory and the computational requirements.

In order to model the geometry of the environment, Ishiguro defines a set of observation points. From each observation point, the robot captures a view I_i of the surrounding environment. Then it compares the captured view I_i with all the views I_j that it has learnt before, computing a measure of similarity, making use of m significant frequencies of the Fourier series, using the formula:

$$Sim(I_i, I_j) = \sum_{y=0}^{l-1} \sum_{k=0}^{m-1} |F_{iy}(k) - F_{jy}(k)| \tag{4.5}$$

In the formula, $F_{iy}(k)$ and $F_{jy}(k)$ are the Fourier coefficients of the frequency k of the row y of images I_i and I_j. The similarity is interpreted as a measure of the distance—the larger the similarity measure, the farther apart the images must be. The relation holds up to a threshold value. Above that threshold, the similarity measure is rendered meaningless, the interpretation being that the images are probably unrelated at all.

Based on the described ideas, Ishiguro proposes a navigation strategy. To start the process, the agent explores the unknown environment, capturing and storing images at a number of reference points. Those images are then processed and organised according to the similarities computed, leading to a geometrical representation that must describe the environment accurately in terms of paths that can be followed, as illustrated in Figure 4.9. The robot is then able to navigate from one reference point to another, choosing the shortest path according to the perceived distance between images.

4.3.4 The T-Net

Another navigation approach, in many aspects related to Ishiguro's method described in Section 4.3.3, is the Target Network (T-Net), also implemented by Hiroshi Ishiguro [44]. A T-Net is similar to

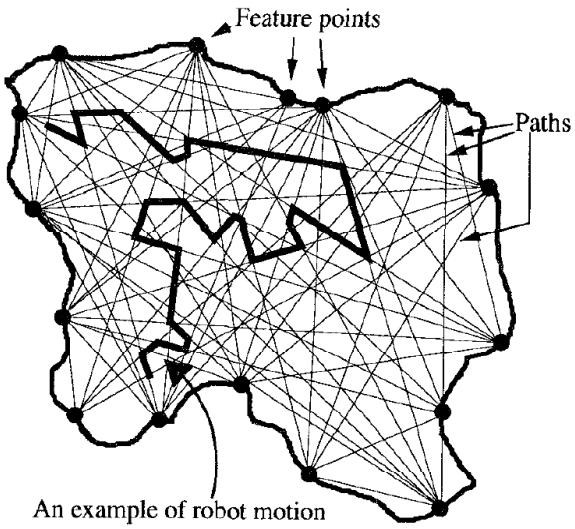
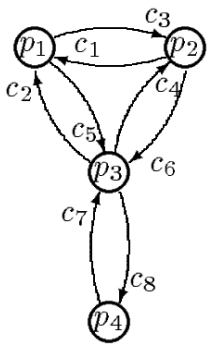


Figure 4.10: Feature points and paths of a T-Net.

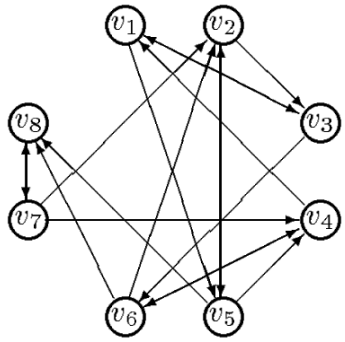
the topological mapping model, but it relies on some key "feature points," paths and intersections between those paths, as illustrated in Figure 4.10. Ishiguro uses a robot with a catadioptric vision system. The robot travels along the paths by tracking the feature points of those paths, which are their starting and ending points. The angular relations between the paths are also calculated, based on the omnidirectional views. As the robot moves, it picks candidates for intersections between paths and explores the environment to verify if the intersections are possible. If the intersections are possible, they are stored, as passage points between different paths. The T-Net is, therefore, constituted by the set of feature points, paths and intersections, thus being a solid basis for quick and efficient route planning and navigation.

Similarity between the images of the feature points is computed by a special algorithm, known as the Sequential Similarity Detection Algorithm (SSDA). However, to make tracking of the feature points possible, it is necessary that the images input to the SSDA algorithm do not present significant changes in size. Ishiguro solves that problem using cameras with zoom. The magnification factor is decreased as the robot approaches the feature point and increased as it moves away from it.

The T-net is, in summary, an interesting and robust approach for vision-based navigation. However, it is clearly messy and expensive to implement. It requires expensive hardware, namely a mobile robot with good processing capabilities and cameras with zoom control. At least two cameras are required, although Ishiguro reports to have used four. A lot of computer power is necessary to process the images, compute the intersections between paths in real time and confirm those intersections. An open problem is that of defining the feature points. There may be a way to do it automatically, but the problem is not addressed by the author. In summary, although the approach may look appealing and robust, it comes at a high cost of implementation and running.



(a) Topological representation.



(b) Corresponsing view graph.

from view

to view	v_1	v_2	v_3	v_4	v_5	v_6	v_7	v_8
v_1	–	–	g_b	g_l	–	–	–	–
v_2	–	–	–	–	g_b	g_r	g_l	–
v_3	g_b	g_r	–	–	–	–	–	–
v_4	–	–	–	–	g_l	g_b	g_r	–
v_5	g_l	g_b	–	–	–	–	–	–
v_6	–	–	g_r	g_b	–	–	–	–
v_7	–	–	–	–	–	–	–	g_b
v_8	–	–	–	–	g_r	g_l	g_b	–

(c) Transition table of robot motions (gb=go back, gf=go forward, gr=go right and gl=go left).

Figure 4.11: Representations of a maze environment, as used by Mallot.

4.3.5 Mallot's approach

Mallot et al. [62] use infrared views of the environment for robot localisation in a maze and implement path planning using a topological representation. The robot explores the environment, by detecting close objects using the infrared sensors. When an object is detected in the moving direction of the robot, the robot stops to avoid colliding with that object and stores the sequence of readings for future reference.

The sequences of views retained are stored into a neural network. The neural network is trained to associate a current view and motion with the next view. That way the system can later retrieve from the neural network which places can be achieved from any given place. The authors use that behaviour to perform path-planning. By presenting the neural network the possible images and suggested motion decisions, it is possible to simulate a complete path without actually following it. To a great extent, that is very alike the human ability to mentally follow any given path.

From the view sequences, Mallot builds a topological representation for higher level planning tasks. Figure 4.11 shows an example of the technique. The first graph is the higher level topological representation of a maze. The points p_i are actual places in the maze. Those places are connected through corridors c_i. Since the corridors can be traversed in either direction, each corridor is represented by two arcs in the graph. Figure 4.11(b) represents the corresponding view graph. The view graph is much more complex, since each place p_i in the topological representation can originate several nodes in the view graph. Those nodes all correspond to the same place, sensed from different points of view. Figure 4.11(c) represents the corresponding transition table. The table summarises the movements which have been learnt to move from one view to another. The transition g_b means *go backward*, g_f means *go forward*, g_l means *go left* and g_r means *go right*.

4.3.6 Other view-based approaches

Stephen Jones et al. [51] also attempted view-based navigation, in an approach very similar to Matsumoto's idea described in Section 4.3.3. The images used in Jones' system are not panoramic, but low resolution wide angle images. Correlation between the images is computed using a method of Zero mean Normalised Cross Correlation (ZNCC). To correct small drifts when following a path, the trajectory of the robot is adjusted in order to maximise the correlation between the images grabbed by the robot and the images stored in the database. Odometric information is also used to correct the estimated position of the robot and make the system more robust.

A similar approach is that of Mao-Hai Li et al. [61]. Li partitions the images and generates a k-dimensional tree of features during the learning stage. Estimation is improved using particle-filters. During the autonomous run, those features are matched using a nearest-neighbour technique.

Niall Winters and Santos-Victor [108] use yet another different approach. The authors also propose a view-based navigation method, using omni-directional images. The images are compressed using Principal Component Analysis (PCA), thus producing a low-dimensional eigenspace. The eigenvalues of the images are projected into the eigenspace and connected by a linear interpolation model. That forms a manifold in the eigenspace, which represents the path that the robot must follow. The manifold is a kind of topological map of the environment. The robot is then guided with the goal of minimising the angle between the path followed and the desired path.

Winters and Santos-Victor's approach is in many aspects similar to the approach followed by Matthias O. Franz et al. [27]. However, the latter authors do not use compressed images. The manifold is constituted by the images themselves. To save computational requirements, only images that are relevant to the navigation task are stored, which means that only "snapshot images" taken at decision points are retained. The authors also consider the problem of similar views, which is solved using the past history of the robot, assuming that the robot moves continuously in the navigation space.

4.3.7 Associative memory using a look-up table

R. Nelson [77] uses an associative memory implemented through a look-up table to home a robot. Homing is the process of returning a robot to a key position, known as the home of the robot, or its default location. It is useful to home a robot for many reasons. Those reasons include, among others, to recharge batteries or simply to await further commands.

Nelson's robot also performs vision-based navigation. A camera mounted on top of the robot captures images in real time, and it uses visual information to localise itself. During a learning stage, each visual

pattern (image) is associated with a set of motor commands, so that for each known position the robot can later, autonomously, perform the necessary steps to return to its home position. The associations are learnt through the use of a look-up table. The similarity between images is computed using a nearest-neighbour technique.

Look-up tables are a very straightforward method to implement an associative memory. They are attractive for their simplicity. However, they exhibit several drawbacks which make them unsuitable to implement general-purpose associative memories. Some of those drawbacks are:

- State vectors of mobile robots, in real applications, tend to be of very high dimensionality. Therefore, it is not practical to use state vectors as address references or indexes for look-up tables, because the complete table easily becomes huge and requires unbearable quantities of physical memory. That problem can be addressed if the look-up table contains two entries, one for address and other for data. In that case, it can work like a kind of linked list, where items can be added or removed at will. Nonetheless the memory requirements can easily get quite large if there is no additional management of the entries, and each state vector is required to be stored into the memory;

- Despite its simplicity, a look-up table may easily become computationally very expensive, if the address space is large enough. Training or searching a look-up table may require more time than what is acceptable in many practical circumstances;

- A simple look-up strategy is hardly a general solution for normal robot navigation tasks. In the best case, it has to be complemented with other techniques to be of practical use. For example, noise filters, to filter out noise from sensorial information, or more powerful data structures to help perform higher level planning tasks.

4.4 Previous work using SDMs

The SDM model is now decades old, and there is already some work done on equipping robots with SDMs. This section briefly reviews some important work done in the area.

4.4.1 Rao and Fuentes Goal-Directed navigation

Rao and Fuentes [82] were probably the first to use a SDM in the robotics domain. They used the SDM to store perception (sensorial) information, which is later used to navigate the robot.

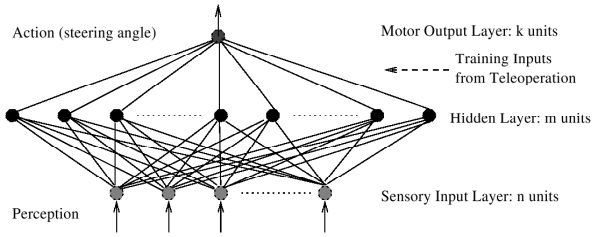


Figure 4.12: Rao and Fuentes implementation of a SDM. Source: [82].

Rao and Fuentes' model is a hierarchical system, inspired by Brooks' subsumption architecture, in which the lower levels are responsible for lower level tasks, such as collision detection and obstacle avoidance. At that level the behaviour of the system is essentially reactive, and a simple hill-climbing learning algorithm was implemented.

The SDM used is implemented as a neural network, as shown in Figure 4.12. That neural network is used at the upper level of the hierarchy, to provide goal-directed navigation. It is trained by the user, who manually guides the robot through the desired path, during a learning stage. While in the learning mode, the sensorial information from optical sensors (perception) is recorded and stored into the SDM, along with the motor control vectors. During the autonomous run, similar perceptions (readings from the optical sensors), are expected to retrieve the correct actions (motor commands), making the system follow the same path by using that information, and thus repeating the same trajectory step by step. The authors report simulation results only, and they mention nothing about drift corrections or other error control strategies.

The SDM used in Rao and Fuentes' system is also a variant of the original Kanerva's proposal. The initial addresses are picked at random, as Kanerva suggests, and represent the first layer of the neural network. Hence, before operation, the weights w_j of the neural connections are assigned random values. But, contrary to the original idea, in Rao and Fuentes' implementation the values of those neurons can be changed, adjusting the addresses towards the most active regions of the address space. Rao and Fuentes call that implementation a "Self-Organising SDM".

Learning in a self-organising SDM, both for the address and the data spaces, occurs using a soft competitive learning rule. For each perception (input) vector p, at time t, the Euclidian distance between p and the corresponding neural weight w_j is computed: $d_j = |w_j^t - p|$. Then, the weight vector is adjusted as follows:

$$w_j^{t+1} \leftarrow w_j^t + g_j(t) \cdot P_t(d_j) \cdot (p - w_j^t) \tag{4.6}$$

In the equation, P is the equivalent of the probability of the prototype vector w_j winning the current

competition for perception p, which, for m neurons in a layer, is computed as follows:

$$P_t(d_j) = \frac{e^{\frac{-d_j^2}{\lambda_j(t)}}}{\sum_{i=1}^{m} e^{\frac{-d_j^2}{\lambda_j(t)}}} \tag{4.7}$$

$\lambda_j(t)$ is the equivalent of the "temperature" of the system, thus representing its susceptibility to change. The temperature is higher at the beginning and lowers as the system stabilises over time. The parameter $g_j(t)$ is also a stabiliser variable, given by:

$$g_j(t) = \frac{1}{n_j(t)}, where \quad n_j(t+1) = n_j(t) + P_t(d_j) \tag{4.8}$$

4.4.2 Watanabe et al. AGV

Watanabe et al. [105] also applied a SDM in robotics, for the task of scene recognition in a factory environment. Their goal was to build intelligent vehicles that could move around autonomously—Autonomously Guided Vehicles (AGV)—in industrial environments. Those vehicles are usually required to run the same routes in a repetitive manner, transporting tools or other materials, from one point to another of the production line. Under normal circumstances, it is required that the vehicles must be able to identify small changes in the environment, such as the presense of other AGVs, people, tools or other entities that may get into the route of the AGV. Even in the presence of those obstacles, intelligent AGVs must avoid collisions and successfully complete their goals.

Watanabe et al. use a Q-learning technique to teach the robots the paths they must perform, and also to learn the behaviour to negotiate with other AGVs they can encounter in the environment. Scene recognition is performed based on the use of a SDM. The SDM is, therefore, used as a pattern recognition tool.

The factory plan is input to the AGV in the form of an occupancy grid. When moving, for each grid position the AGV is in, the system computes in real time the correct motion to execute, in function of the designated goal. If at any given point its sensorial readings differ from the expected sensorial readings, which are stored into the SDM, then an obstacle is probably in the way of the AGV. In that case, the vehicle has to replan its trajectory and find an alternative way to bypass the obstacle and still make its way to the target point.

To the best of our knowledge, Watanabe et al. report simulation results only, for a virtual AGV that can sense obstacles around itself and capture the scenes to store into the SDM. That AGV always has access to its current location, as well as its goal location. The SDM used was implemented as a

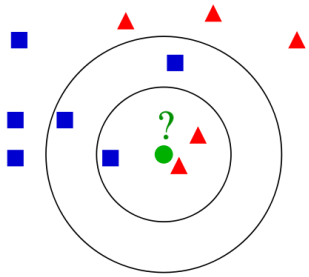


Figure 4.13: Example of k-nearest neighbour classification.[4]

neural network.

4.4.3 Similar techniques

There are other models which are similar in purpose to the SDM, in the sense that they are suitable to work with sequences of patterns and similarity-based techniques. Those include the K-Nearest Neighbour (KNN) approach, Hidden Markov Models (HMM) and Support Vector Machines (SVM).

Nearest-neighbour

The k-nearest neighbour approach is very popular, for it is among the simplest techniques used in machine learning and data classification. The strategy is as follows: for a given datum that needs to be classified into a given set of categories, search the k nearest neighbours, according to some metric, and assign the new datum to the category that is the most frequent amongst the k nearest neighbours. Figure 4.13 illustrates the idea. In the figure, the green circle needs to be assigned to a pre-existing class. There are two classes: the red triangles and the blue squares. If k is set to 3, then the circle is assigned to the class of the red triangles, because there are two red triangles and only one blue square in a radius of 3 elements. If the radius is extended to 5, the circle is assigned to the class of the blue squares, because those outnumber the red triangles that exist in a radius of 5 elements.

R. Nelson used a nearest-neighbour approach, as described in Section 4.3.7, for robot navigation. To a great extent, the SDM is also a nearest-neighbour model. However, the concept of activation radius is used in a different way in the SDM, and the selection process is also different. Using a KNN it is required that the k nearest elements are retrieved. Then the selection is made through a mechanism of popularity, and the output is an element of the dataset. In the SDM, all the elements that are *similar up to a given degree* are selected, and the output must/can account for all of them. The result is that a pure KNN technique is more vulnerable to the effect of outliers. It also does not implement mechanisms which are natural for the SDM and biologically inspired. Those include behaviours such as "forgetting"

[4]Image source: http://en.wikipedia.org/wiki/K-nearest_neighbor_algorithm (last checked 2009.12.25).

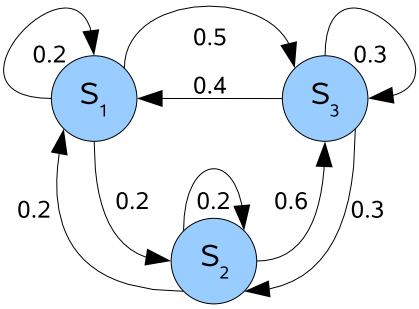


Figure 4.14: Example of a system governed by probabilities.

and associating related data. In that aspects, the SDM is a more complete and sophisticated model, when compared to the KNN approach.

Hidden Markov Model

The HMM is a statistical model which describes the transitions of rigid system states in terms of probabilities. The model can be applied to systems that have a finite number of states, if the transitions between the states are governed by a set of probabilities. Knowing the state x_t, which represents system x at time t, the model predicts the state x_{t+1} with some probability. At each moment, the current state determines the probability distribution for the next state. Figure 4.14 shows an example of a system with three states, in which the transitions between states are governed by probabilities. For example, if at a given instant the system is in state S_1 ($x_t = S_1$), then it is known that in the next observation there is a probability of 0.5 that the system is in state S_3 [i.e., $P(x_{t+1} = S_3 | x_t = S_1) = 0.5$].

HMMs have been successfully used for pattern recognition, in several areas. They are quite popular in speech recognition, where the sounds are the input patterns and the written words are the hidden states that need to be predicted. They have also been used in gesture recognition, handwritting recognition, musical score following and similar tasks.

Some research has also been done on using HMMs to predict the navigation states of mobile robots, based on the use of a topological map. Trentin and Cattoni [99], Koenig and Simmons [56] and Shatkay and Kaelbling [94] are examples of such approaches.

The HMMs are an interesting and powerful approach for state estimation. However, they have some drawbacks that render them very difficult to use in practice as a general purpose technique. They are not a flexible approach. Instead, they require some *a priori* knowledge or training to the specific environments they are going to model. There are two sets of parameters that have to be learnt: structure of the environment and the probability distributions. It is necessary to know the structure of the environment to create the model with the correct number of states. While the SDM can start empty

and grow until it is very large, the HMM model does not have that versatility. As for the probability distributions, they also have to be set up, or the system has to be trained to learn the most accurate values before it can make useful predictions. In summary, the HMMs are a very robust predictor, but they are a very rigid model. Their application is limited to environments which are known in advance. And even so, they are more suitable to work with topological representations, which lead to less states than non-topological approaches[5].

Support Vector Machines

Support Vector Machines are a popular model which is used for data classification. Figure 4.15 shows a 2-dimensional example of how the model works. In the example, there are two sets of geometric figures: blue squares and green circles. Those figures are set apart, and it is possible to find the plane that best separates them through a process of minimising the distance errors between the points and the plane. That is the same as maximising the distance between the plane (continuous line) and the margin planes (dashed lines). Once the best plane is found, new examples can be easily classified by determining which side of the plane they belong to.

SVMs have been successfully used for data classification in many areas, such as text and image recognition. They are better suited for data classification in just two classes, but the method has been extended to accommodate more classes. One strategy to deal with multiple classes is to create a set of binary classifiers. There are two major approaches for doing that: (i) one-against-all, and (ii) one-against-one. In the first approach, one-against-all, the binary classifiers are trained to recognise only one class against all the other classes. For a given test datum, the classifier with the larger output defines the class. In the second approach, one-against-one, one binary classifier is created for each pair of classes. For a given test datum, it is classified using a voting strategy, in which it is assigned to the class that receives more votes. Another popular approach is to build a tree of binary classifiers. Using the binary tree of classifiers, it is possible to start at the root of the tree and move down in the hierarchy step by step after each classification. After each binary classification, it is found which part of the tree the datum belongs to. Therefore, the uninteresting branches are pruned. That way, it is possible to avoid processing the datum through many of the classifiers.

SVMs have already been used for robot navigation. Schwenker et al. used SVMs for object recognition from a single view during robot navigation [92]. For quicker processing, Schwenker uses a hierarchy of binary classifiers, as described above. Wurm et al. use SVMs and laser scans to detect vegetation in the terrain, so that the robot can avoid areas covered with grass or grass-like vegetation [109]. Aviña-

[5]Recently, Dizan Vasquez [103] reports on an extension of the model to accommodate an increasing number of states, so that new states can be added to the model when needed.

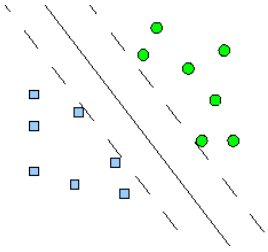


Figure 4.15: Illustration of the SVM operation mode.

Table 4.1: Characteristics of approaches to store sequences of images.

	SDM	KNN	NN	HMM	SVM
Learning	natural	-	difficult	difficult	difficult
Forgetting	natural	-	difficult	-	-
Associativity	natural	natural	natural	-	-
Incomplete data	natural	robust	robust	robust	robust
Noise	robust	robust	robust	robust	robust
Dimensions	high	high	high	limited	high
Hardware requirements	high	medium	high	very high	very high

Cervantes uses SVMs to classify human-made roads, based on colour images [75]. Lei [60] presents a very complete survey of important applications as of 2005.

One great advantage of the SVMs is that they produce only one global and unique solution. There are no local minima traps. The solution is always unique, and the SVMs work very well with high-dimensional spaces. Two important drawbacks are the speed of learning and the computational requirements. The SVMs are slower than typical neural networks for a similar generalisation performance. Also, the choice of the SVM parameters strongly influences the performance of the system, but there are no rules that can guide the choice. SVM models are suitable for specific classifications, such as tell vegetation or a road from the remainder object scenarios. However, for general-purpose vision-guided navigation, they are hardly appropriate.

Comparison

Table 4.1 summarises some important characteristics of the SDM and similar approaches. The first two rows compare the ability to learn and to forget. The third row concerns the ability to associate two different data patterns. The fourth and fifth rows compare the ability to deal with incomplete and noisy data. The sixth row concerns the ability to deal with high-dimensional vectors. The last row refers to the computational requirements of the model. The columns refer to the models compared: the sparse distributed memory, k-nearest neighbours, neural networks, hidden Markov models and support vector machines.

As the table shows, the SDM is superior in terms of learning and forgetting abilities. It can implement one-shot learning and natural forgetting. SDM, KNN and NN work as associative memories, but HMM and SVM are hardly suitable for that purpose. In terms of ability to deal with incomplete and noisy data, as well as suitability to work with high-dimensional spaces, all the approaches can be considered appropriate. In terms of computational requirements, it seems fair to assume the KNN approach is the less demanding, while the SVM and HMM should be the most demanding. The SDM simulated in software is also very computer intensive, but it is suitable to be implemented in dedicated hardware. In that case, the performance could be considerably improved.

4.5 Summary

This chapter presented a short review of the important concepts and works in robot navigation. It starts with a brief presentation of the pipeline and hierarchical robot architecture models. The pipeline architecture is more straightforward, but less robust and more difficult to scale for complex systems.

Some important mapping models are also presented. Table 4.2 compares those models in terms of sensorial requirements, memory requirements, processing power needed and scalability of the systems. Occupancy grids are a very simple approach, and can be built using different sensors. However, they require large amounts of memory for large spaces or resolutions. The processing requirements are also high, and the models are difficult to scale. Voronoi diagrams are a tool that facilitates the navigation process, by providing a means for safe localisation of the robot, possibly building on occupancy grids. They are easy to implement and scale. Topological maps are more sophisticated representations of the environment, but also more succint and requiring a higher level of abstraction. They provide succinct and powerful descriptions of the world, in terms of paths and connection points. They ignore details, thus being ideal for high-level path planning in complement to other methods that implement local navigation strategies. View sequence-based navigation has the advantage of being biologically motivated. A single sensor provides huge amounts of information. However, extracting information from images is difficult, and storing them requires large amounts of disk space. That approach is often used with a supervised learning stage. The use of landmarks greatly facilitates the mapping process. The landmarks are key points that provide relevant information of the environment. However, they usually have to be manually placed in the environment or selected, thus not being a general solution.

Table 4.3 summarises the view-based navigation approaches that were described in this chapter. Some of them use more than one camera, or omniview images. Those use more information, and the complexity of the system and the hardware requirements increase. The omnidirectional images are a useful tool, but difficult to process and not biologically inspired—at least in what concerns human vision,

Table 4.2: Summary of the analysed mapping approaches.

Method	Sensorial requirements	Memory requirements	Processing	Scalability
Occupancy Grids	various	huge	high	low
Voronoi Diagrams	various	large	large	medium
Topological Maps	various	low	low	good
View Sequence	camera	huge	large	good
Landmark-based	landmarks	low	low	low

Table 4.3: Summary of the analysed view-based navigation approaches.

Approach	Images	Landmarks	Compression	Similarity
Matsumoto	omniview	intersections	–	block match
Iconic memory	omniview	defined points	Fourier	Fourier coef.
T-net	4 cameras	path ends	–	SSDA
S. Jones	wide angle	–	–	ZNCC
Winters	omniview	–	PCA	eigenvalues
Franz	omniview	intersections	–	neural net
Mallot	infrared	–	–	neural net
Nelson	1 camera	–	–	KNN
Li	1 camera	features	–	KNN

because animals such as flies actually use similar vision. Some of the approaches also use landmarks or special points. Those landmarks are intersections, user defined points, the beginning and end of paths, or detected feature points. In order to speed up processing in real time or reduce memory requirements, some authors also use compression of the images, based on Fourier transforms or Principal Component Analysis. The last column of the table refers to the method used to compute the similarity between images.

Section 4.4 reviews important previous works that also used SDMs for robot navigation. Rao and Fuentes used a SDM to store sequences of sensorial readings and motor commands that were later used by the robot to follow the same path. Watanabe at al. used a SDM as a pattern recognition tool, to recognise environment scenes.

This chapter also compares the SDM to similar techniques, such as KNN, HMM and SVM. The SDM presents the most interesting characteristics of all the methods reviewed. It is highly robust to noise, works with incomplete information, is suitable to work with high-dimensional vectors and learns and forgets in a natural way.

Chapter 5

Experimental platform

The most exciting phrase to hear in science, the one that heralds new discoveries, is not 'Eureka!' but 'That's funny...'
(Isaac Asimov, American author and biochemist)

Contents

Chapter 3 describes the Sparse Distributed Memory model. Chapter 4 reviews important notions of robot architecture and navigation. This chapter intends to describe the hardware and software architectures of the experimental platform used to apply a Sparse Distributed Memory to navigate a robot based on visual information. First, the characteristics of the hardware platform are reviewed. Then the architecture of the software developed and used is described, as well as the techniques applied to improve the quality of the images.

5.1 Hardware and software

The main hardware used to perform the experiments includes a small robot that has a built-in video camera and a radio communication module. There is also a laptop computer which runs the software and controls the robot in real time via a wireless link.

5.1.1 The robot

The robot used is a Surveyor[1] SRV-1, as shown in Figure 5.1. Although that Surveyor is not very precise, it is very robust and provides enough sensorial information and versatility to perform the desired tests. It is a small open-source robot, with tank-style treads. It was named "Lizard," after the Portuguese word *lagarta*, which in Portuguese can mean both "lizard," the animal, and "tread".

The chassis of Lizard is made of machined aluminium. The total weight of the robot is approximately 300 grams, and the dimensions are approximately $120 \times 100 \times 80$ milimeters. It is powered by a 7.2 V Lithium-ion polymer battery with a capacity of 2 Ah. That allows a maximum operation time of about four hours per charge. The robot has differential drive via two precision DC gearmotors. The speed range is 20 $\text{cm}\cdot\text{s}^{-1}$ to 40 $\text{cm}\cdot\text{s}^{-1}$. The board includes a 60 mips 32-bit processor, 64 Kb SRAM memory and JTAG interface.

Lizard has a built-in digital video camera. The camera is placed in the front of the platform, and can provide JPEG images with resolutions of 80×64, 160×128 and 640×480 pixels. The field of view was experimentally determined to be approximately 40^{o}.

The robot also has four infrared sensors. Those sensors are placed one on each side of the robot (front, rear, left and right), in sets of emitter-and-receiver, and can be used as short-range proximity detectors.

Lizard also has a built-in communication module, which operates in the ISM (Industrial, Scientific & Medical) frequency band of 2.4 GHz and provides IEEE 802.15.4 standard communication. The transmitter power is 100 mW and the receiver sensitivity is up to -100 dB. Those characteristics should provide an efficient communication at distances of up to 100 m indoors and up to 1500 m in line of sight outdoors. Data rate is up to 250 000 bit/s.

[1] http://www.surveyor.com (last checked 2009.12.27).

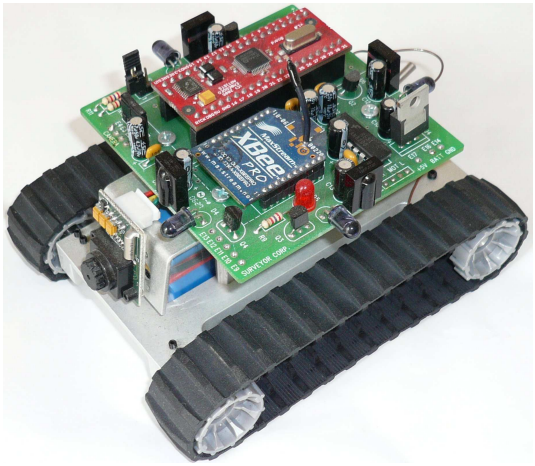


Figure 5.1: Robot used. There is a camera on the front (left-hand side).

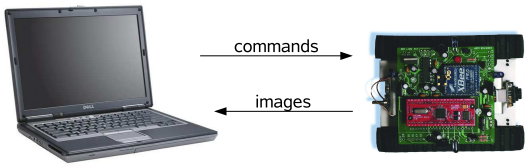


Figure 5.2: Computer and robot used.

5.1.2 Computer control system

Although the robot can operate autonomously, the memory and processor available on board are not powerful enough to store and run a SDM in real time. Therefore, the control software was designed and implemented to run in a laptop. The laptop communicates with the robot via a wireless link, as represented in Figure 5.2. That architecture also facilitates the user interface and control, as well as the test and debugging processes.

The laptop used has a 1.8 GHz processor and 1 Gb RAM. Real time wireless communication is established between the radio module available in the robot, and another module connected to the laptop via a USB cable. That wireless link is fast enough to provide satisfactory speed of operation in real time. During normal operation, the robot captures its current front views using the camera and sends them to the computer, compressed in JPEG format. Then the computer processes the images and sends motion commands to the robot.

The laptop is running OpenSuSE Linux 10.2, and the software was written in C++, using Qt libraries for the graphical interface and some image processing functions.

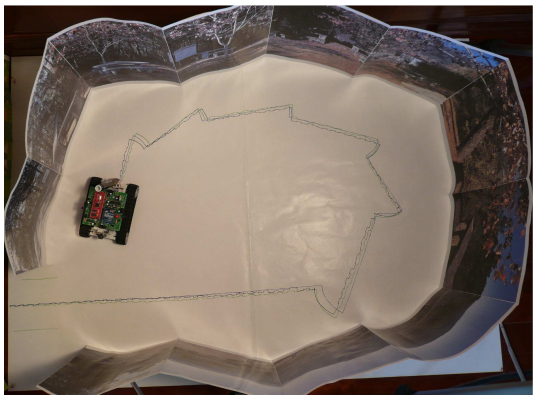


Figure 5.3: Top view of the robot testbed, surrounded by one of the mountain scenarios.

5.1.3 Testbed

To facilitate the process of repeating experiments in controlled conditions, the navigation experiments were performed in an arena as shown in Figure 5.3. The ground surface of the arena is covered with a sheet of white paper that can be easily replaced between experiments. In the first version of the arena, the sheet used was 1 meter long and 0.7 meters wide. That is the version shown in Figure 5.3. Nonetheless, soon the length of the sheet was doubled, in order to get an arena 2 meters long 0.7 meters wide, which is an appropriate dimension for the size of the robot. To control the scenario that is captured by the camera, the robot is surrounded by a paper wall 30 cm high. The wall can be covered with sets of pictures of different scenarios. Most of the tests were performed with a scenario in which the wall is covered with pictures of different mountain environments. Those mountain environments include images of trees and vegetation, villages and other countryside views. Some tests were also performed with other pictures, such urban environments showing streets, buildings and cars, and single objects such as cars and trees.

Lizard was equipped with an apparatus to maintain a marker pen attached to its rear. That pen moves behind the robot, causing only a minimum interference while being dragged. Using that technique it is possible to make the robot draw a line that marks the path on the paper as it moves, so that the different paths followed by the robot can be easily tracked and compared.

5.1.4 Software

The architecture of the software can be represented, in a simplified graphical form, by the block diagram that is shown in Figure 5.4. It contains three important modules:

1. The SDM. That is the module where the sequences of images and additional data are to be stored and searched.

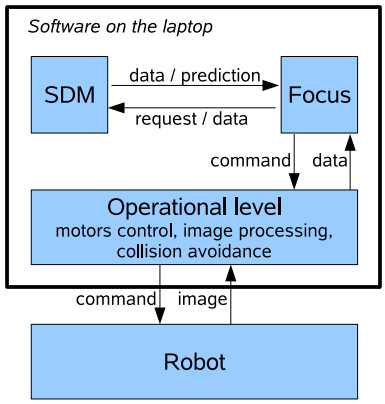


Figure 5.4: Architecture of the implemented software.

2. The Focus. That module is named according to Kanerva's terminology. It is where the navigation and control algorithms are run.

3. Operational level. That layer is responsible for interfacing the hardware and some low-level tasks, such as controlling the motors, avoiding collisions, and grabbing the images and processing them to improve their quality. That level is described in more detail in Section 5.2.

5.2 Operational level

The operational level is where the most basic functions are performed. In historical terms that level can be considered as inspired by the first level of the Brooks' subsumption architecture, described in Section 4.1. In biological terms, it can be considered as inspired by the "old brain," which controls basic body functions such as the breathing and feeding instincts. Some more details about the brain structure are given in Section 2.2.2.

5.2.1 Behaviours and states

The operational level is responsible for interfacing the hardware and implementing very simple behaviours that simplify the tasks of the levels above. It takes care of dealing with the sensorial inputs. That includes receiving the readings from the infrared sensors, as well as images from the robot's camera. It implements some basic image processing to filter out some noise and improve the quality of the images to the levels above, as described in Section 5.2.2 in more detail. The readings from the infrared sensors are not as noisy as the images. But they also come with noise and occasional outliers that must be dealt with for better accuracy. One common method of dealing with outliers and noise in that kind of situations is to sample the sensors *a few times* and use the median, or the average, of the readings.

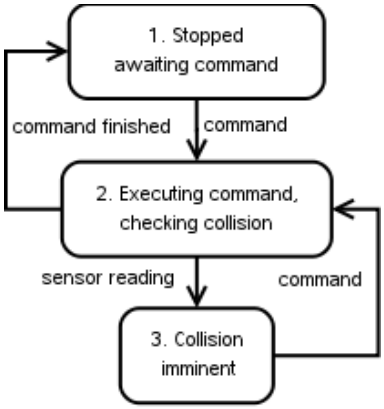


Figure 5.5: Possible states of the operational level.

In the present work, considering the characteristics of the sensors, it was deemed appropriate to use the average. The median is more efficient in filtering out outliers, but those are very uncommon for the IR sensors used. The operational level also processes the infraread readings, by sampling the sensors three times and computing the average for each value that is passed on to the Focus.

The operational level operates as a finite automaton, with three possible states. Figure 5.5 represents the model. The possible states are:

1. Stopped. That is the default state. The robot is idle and awaits commands from the upper levels to skip to state two.

2. Executing order. In that state, it is processing an order from the upper levels. The robot attempts to execute all the orders that are received. While in the second state, the images collected from the camera are analysed to possibly detect when focus is lost. The proximity to any object is also monitored by reading the IR sensors. An image out of focus or an IR reading above a given threshold are considered an indication that a collision is imminent. If that happens, the system jumps to state 3, in order to avoid the collision.

3. Collision imminent. When images taken from the camera are out of focus, or the IR sensor in the direction of the motion returns a reading above a given threshold, the system jumps to state 3. State 3 is a kind of emergency state. The motors are all stopped immediately, and a message is generated to warn the robot user. Any command from the above level will make the system return to state 2. In state 2 the robot will try to execute the command, and if successful it will return to state 1 after the command is finished, resuming normal operation.

5.2.2 Image processing

In order to facilitate user interface and improve the performance of the system, the images may be processed before being used for navigation purposes. The most important processing method is equalisation, which makes the system much more tolerant to illumination changes.

Image capture

The robot captures the images and converts them in JPEG. The JPEG format is very convenient for data transmission, since it is very compact. Once the computer receives the images, they are then converted into 8-bit Portable Gray Map images (PGM). The colour information is disregarded, to reduce processing and storage requirements. All the processing, from that point onwards, is therefore done with 256 gray levels images. As for the resolution, two operation modes were implemented. In the lower resolution mode, the images are 80×64 pixels. In the higher resolution, the images are twice that dimensions (160×128). As will be emphasised later, the smaller images are are much faster to process, since they have only one quarter of the number of bytes of the larger images. There is no noticeable impact on other aspects of the performance, under the circumstances in which the experiments were carried[2].

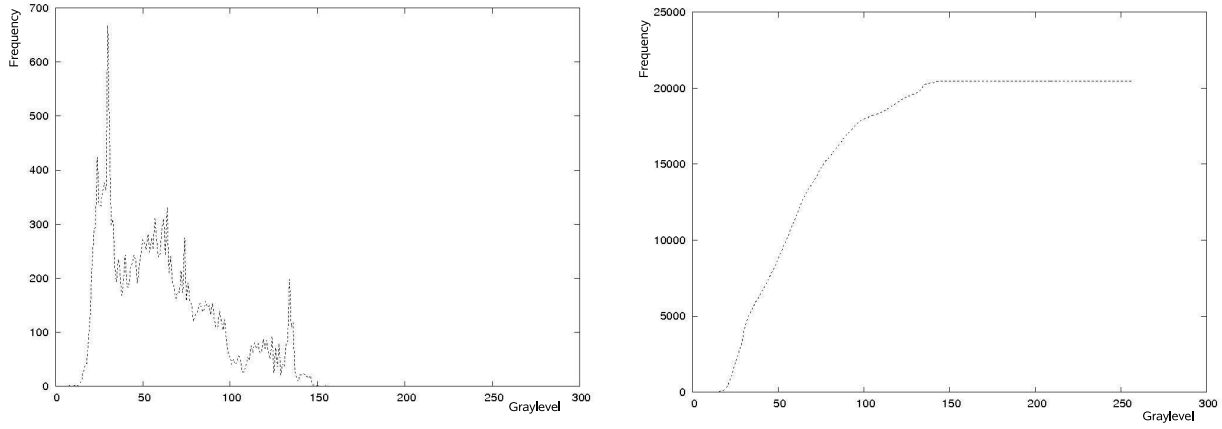
Image enhancement

The quality of images as grabbed directly from the camera depends a lot on ambient illumination. Dim light produces dark images with little contrast, while good illumination provides lighter images with better contrast. The practical consequence of those dynamic environmental conditions is that a robot which has been in one place and captured images of it, will not recognise the same place if it goes there again and the illumination is significantly different. In the present work, images are just arrays of integers in the interval [0, 255]. Under dim light the numbers will tend towards one end of the interval (towards 0, the black pixel). Under clear light the values will move towards the other end of the interval (towards 255, the white pixel). The difference between the images is an error that may eventually lead the system to consider them as two different pictures.

The illumination problem is very common in vision-based systems, and it is very difficult to solve. So far, there is no universal solution to it. But using some computer vision techniques, it can be minimised. Contrast stretching, also called contrast normalisation, is one of such techniques. Histogram equalisation is another technique, possibly more commonly used, since it is not computationally expensive and

[2]That is in line with the claims of other authors, such as Matsumoto, who uses images as small as 32×32, as explained in Section 4.3.2.

Figure 5.6: Image captured under dim light.



(a) Histogram.

(b) Cumulative histogram.

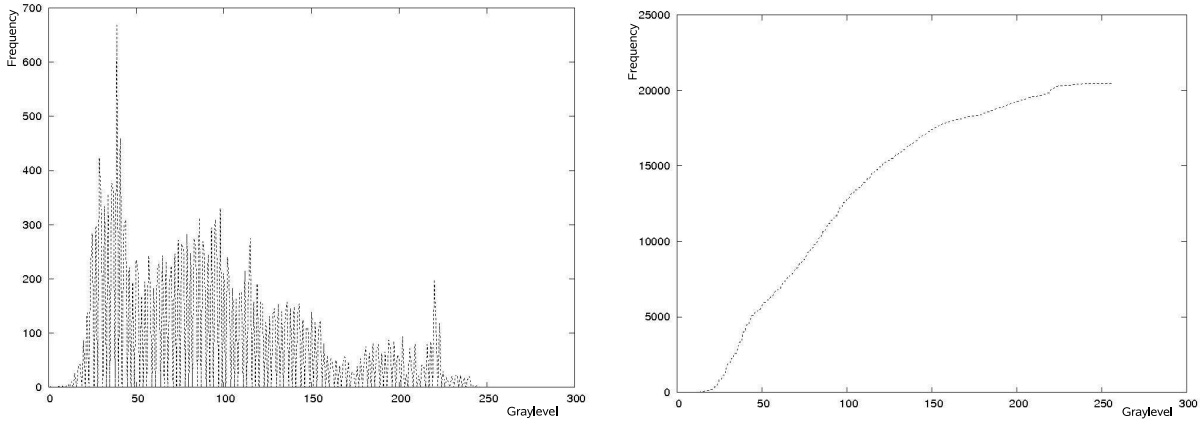
Figure 5.7: Frequency and cumulative histograms of the original image.

produces better results [24].

Both contrast stretching and histogram equalisation were tested in the present work, with an image of a gray car captured under dim light. The image is shown in Figure 5.6. The frequency and cumulative histograms are as shown in Figure 5.7. As the figures show, the image is very dark. It is very rich in brightness intensities below 150, and contains almost no values above that level. The graylevels with higher frequencies are actually very dark, below 50. The cumulative histogram is a curve which rises very fast until 100, and a horizontal line above 150. Besides the visual discomfort of working with such images (which is only relevant during the learning stage), dealing with them in the robot memory can also be difficult, as stated above. Computing the error between two images of the same place, one with the histogram tending to the left and the other with the histogram tending to the right, can possibly result in a huge difference, meaning that the images may not be considered the same by the robot. Contrast stretching and histogram equalisation techniques may help overcoming that problem.

Contrast stretching

Contrast stretching is a method that consists in applying a linear function to the image, in order to improve its contrast. It is also called contrast normalisation. The intensities of an image with poor



(a) Histogram. (b) Cumulative histogram.

Figure 5.8: Frequency and cumulative histograms of the normalised image.

contrast, such as the test image shown in Figure 5.6, do not extend through the full range of available pixel values. In the case of the test image they all actually fall in the range [7, 178]. Applying a linear transform to the image it is possible to stretch the intensity levels so that they occupy all the range [0, 255], resulting in an image with improved contrast. Mathematically, the technique is simply a function that maps one interval into another interval:

$$f : [a, b] \longrightarrow [c, d] \tag{5.1}$$

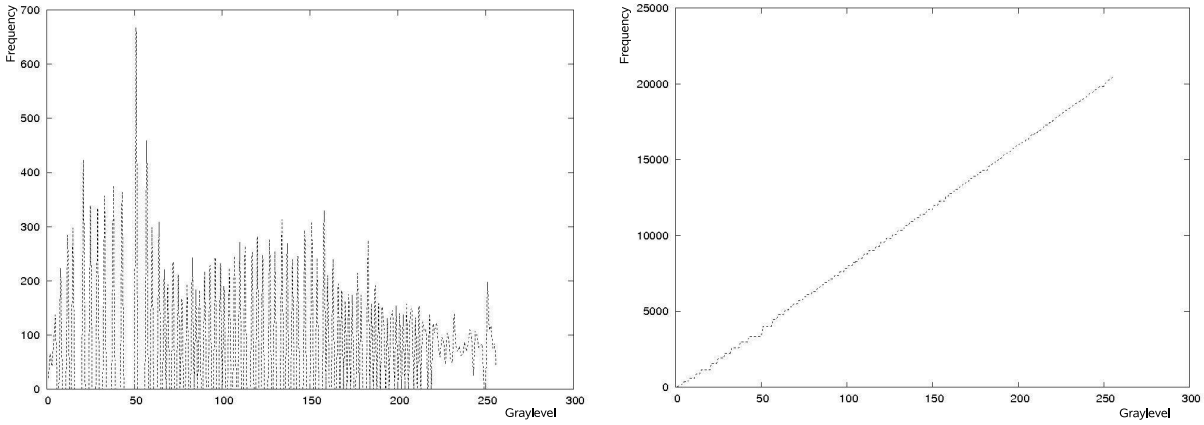
In Equation 5.1, a and b are the minimum and maximum intensity levels found in the image, and c and d are the minimum and maximum intensity levels that can be used. The value of each pixel P_{in} is then reassigned into P_{out}, according to Equation 5.2:

$$P_{out} = c + (P_{in} - a)\left(\frac{d - c}{b - a}\right) \tag{5.2}$$

Figure 5.8 shows the result of applying the contrast-stretching transform to the image. As shown, both ends of the the histogram were stretched, so that it occupies all the range [0, 255]. There is at least one pixel with intensity 0, and there is also at least one pixel with intensity 255. The cumulative histogram was turned into a curve without long horizontal segments. Therefore, the distribution of the pixel values over all the available range was significantly improved by the mapping function.

Histogram equalisation

Histogram equalisation is a technique to manipulate images based on the analysis of the corresponding histograms. The method consists in taking the original cumulative histogram of the original image, normalise it towards 255 (or the maximum intensity value that can be found in the image), and then use it as a mapping function to the original image, thus achieving a uniform histogram after the transform.



(a) Histogram.

(b) Cumulative histogram.

Figure 5.9: Frequency and cumulative histograms of the equalised image.

The uniform histogram obtained will correspond to a brightness distribution where all the values should be equally probable. Due to the discrete nature of the digital images, the result is usually just an approximation. Nonetheless, the technique results in a significant improvement of the original images, as shown in Figure 5.9.

In mathematical terms, the procedure is formalised as follows. First, let's consider an image x, of L different gray levels and n pixels. Let n_i be the number of occurrences of graylevel i in the image. The probability of occurrence of a pixel of level i in the image is given by equation:

$$p_x(i) = p(x = i) = \frac{n_i}{n}, i \in 0, ..., L \tag{5.3}$$

The probability $p_x(i)$ is, indeed, the same as the graylevel's frequency value in the histogram, once it is normalised into the interval [0, 1].

The cumulative probability distribution function of p also coincides with the image's normalised cumulative histogram:

$$c(i) = \sum_{j=0}^{i} p_x(j), i \in 0, ..., L \tag{5.4}$$

The value $c(i)$ is a transformation that for each pixel value x will produce a corresponding value y, so that the cumulative probability function of y will be linearised across the value range. That is, if it is used the transformation of Equation 5.5 to map every pixel value in the image from x to y, the output image will exhibit a linear histogram in the codomain [0, 1].

$$y_i = c(i) \cdot x_i, i \in 0, ..., L \tag{5.5}$$

To get the image back into the original [0, L] interval, it is necessary to apply a linear transform to

(a) Original. (b) Normalised. (c) Equalised.

Figure 5.10: Comparison of the original, normalised and equalised images.

stretch the codomain:

$$y'_i = y_i \cdot L \tag{5.6}$$

For real time processing, it is possible to merge the equations 5.4, 5.5 and 5.6 together. That way it is possible to skip the prior normalisation of the cumulative histogram into the interval [0, 1] and perform all the operations with a single loop to process all the image (providing the cumulative histogram has already been computed) [67], as shown in Algorithm 1.

Algorithm 1: Histogram equalisation

```
begin
    alpha ⟵ L/numPixels;
    for each pixel do
        y ⟵ C(x) * alpha;
    end
end
```

In Algorithm 1, $numPixels$ is the total number of pixels in the image, and $C(x)$ the cumulative frequency of graylevel x.

Figure 5.10 shows the original image, and compares the results of normalisation through contrast stretching and histogram equalisation. As can be seen, contrast stretching improves the contrast a little bit. However, the equalised image exhibits the best contrast and quality.

5.2.3 Focus detection by contrast measurement

To detect obstacles, Lizard can make use its infrared sensors. However, to implement an additional level of safety, it is possible to detect obstacles from the visual information captured through the front camera. Besides the advantage of additional safety, that approach has the advantage of confering on the robot the ability to navigate without the IR sensors, maintaining equivalent safety standards. One

Figure 5.11: Artificial image, in (left) and out (right) of focus.

simple method to detect the presence objects close to the robot is to check if the camera is in focus. If the robot is very close to an object, then it will get out of focus and the images captured by the front camera and sent to the computer will be blurred.

There are essentially two classes of methods to determine if a camera is in focus: active and passive methods. Active methods are based on the use of measurement systems, such as laser range finders or ultrasonic devices, to determine the distance from the camera to the target. But that requires additional hardware, so it is the same as using the infrared sensors, which is contrary to the present goal. Passive methods depend solely on the analysis of the images captured. They do not require the use of any beam to direct towards the object to be focused. Passive methods usually work based on the principle that an image in focus will exhibit a significant contrast, while an image out of focus will be blurred and with little contrast between adjacent pixels, as shown in Figure 5.11. Active methods have the advantage of not depending on illumination conditions. They may also be more robust in special situations, such as measuring distances through glass windows or other transparent materials. However, they rely on the reflective characteristics of the obstacles. Objects with special reflective characteristics, such as glass or pure black objects, may not be sensed. Passive methods have the advantage of not depending on reflection conditions, but their accuracy depends a lot on ambient illumination and image contrast in general. A passive method may fail to detect focus in a too dark or too clear image, such as a white wall, even while the camera is correctly adjusted and the sensed object is still very far.

In present work, the goal is to use an algorithm that depends solely on image information. Therefore, a contrast measurement technique is used. Another point to consider is that a thorough analysis of all the image in real time requires some computational resources which are very scarce, considering that all the processing must be done in real time. Considering all that factors, a very simple and straightforward focus detection procedure was implemented. It samples only one or two lines of the image, instead of processing all the pixels. The algorithm starts by processing a single horizontal line in the middle of the image. If the image is considered in focus, no further processing is required. If it is considered out of focus, then a vertical line is also scanned, to confirm the conclusion. Algorithm 2 describes the procedure. The first step is to process an horizontal line and count how many times the difference of the intensities between pixels n and *n-1* is above the given threshold. $I(n)$ is the graylevel value of pixel n

and T is the threshold. After completing the count, if the obtained value is above the minimum, then the image is considered as in focus. It has at least *minCount* edges, therefore it must not be completely blurred. If not enough contrast is detected in the horizontal line, then a vertical line is also scanned, using the same process. If not enough edges are detected also in the vertical line, then it is assumed that the image is out of focus. Obviously, it may happen that the robot is in some position where the camera is capturing same-colour horizontal and vertical scanned lines, without being out of focus due to the presence of a close obstacle. A complete scan of the whole image would also spot objects that may be in one of the currently unscanned quadrants of the image. However, that situations are very unlikely to happen. Thus, it was deemed appropriate to continue scanning just two lines of each image, and assuming the presence of a close obstacle if not enough contrast is detected in those lines.

Algorithm 2: Focus detection

```
Data: image I, width w, height h, threshold T, minCount
Result: 0 if image is out of focus, count otherwise
begin
    count ⟵ 0 ;
    for every n pixel of horizontal image line h/2, from the second to the last do
        if abs(I(n) − I(n − 1)) > T then
            count ⟵ count + 1;
        end
    end
    if count > minCount then
        return count
    end
    count ⟵ 0 ;
    for every n pixel of vertical image line w/2, from the second to the last do
        if abs(I(n) − I(n − 1)) > T then
            count ⟵ count + 1;
        end
    end
    if count > minCount then
        return count
    end
    return 0
end
```

The focus detection algorithm described is very simple and fast. For a small 80×64 image, there are

at most 144 comparisons to decide if it is in or out of focus (142 pixel intensity comparisons, plus two comparisons of the counter). Therefore, it is appropriate to execute in runtime, for every single image. When the robot grabs a new image it processes that image through an implementation of Algorithm 2, in order to decide if it is out of focus or not. In the experiments, a $minCount$ of 3 edges and an intensity difference threshold of 65 produced good results. During the autonomous run of the robot, an image that is assumed to be out of focus, causes the system to skip to state 3 (Figure 5.5) and stop all the motors. A message is sent to the user, which may then decide the next action.

It was experimentally determined that Lizard's infrared sensors detect obstacles at distances in the range 40–90 cm. It is not possible to get precise distances in the range 0–40 cm. On the other hand, the camera only looses focus at distances of 3–5 cm from the object. Therefore, the IR sensors may be used to probe the environment, but, considering the small size of the robot and the proportional experimental arena, the emergency stop is being triggered only when an image is detected out of focus.

5.3 Summary

This chapter described the hardware, testbed and software architecture of the experimental platform. The hardware is a small Surveyor SRV-1 robot, given the name "Lizard". It is controlled in real time from a laptop, via a wireless link. The testbed is a 2×0.7 m^2 arena, surrounded by a wall that can be covered with pictures of different scenarios. The software has three main modules: the memory, the focus and the operational level. The memory is where the data are stored. The focus is where the control algorithms are run. The operational level implements basic behaviours. It is where the hardware is interfaced and the images are equalised and/or contrast stretched to improve quality. It also runs a collision avoidance algorithm, based on image focus detection. The camera looses focus at 3–5 cm from the object.

Chapter 6

SDM-based robot navigation

Not all those that wander are lost.
(J. Tolkien, in *The lord of the rings*)

Contents

Chapter 5 describes the experimental platform—the hardware and the main software modules. This chapter describes the architectures of the SDMs that were implemented and tested, and how the memory parameters were calculated. The first robot navigation algorithm is also described, as well as the preliminary results. Part of this has been published by the author, for the first time, in 2007 [69] and

2008 [71]. When the first tests were performed, it became clear that the version of the memory that was implemented had some problems with information encoding. Those problems are presented in this chapter, and they will be studied in more detail in Chapter 7.

6.1 SDM implementation

Since the goal of the present work is to perform vision-based navigation, it was deemed appropriate to store other sequence and navigation information into the memory, along with the images (namely odometric and image identification data). That additional information will facilitate the process of computing motor commands in real time, and will also be used later to help disambiguation in cases of very similar images. This section describes the structure of the address and data vectors, as well as the structure of the SDMs actually implemented. The characteristics of those models are also demonstrated, based on the characteristics of the equivalent boolean space.

6.1.1 First design

As described in Chapter 5, the SDM will be used to store sequences of images that are necessary for robot localisation and navigation. Those images are to be stored in 256 gray levels PGM format, where each pixel is represented by an integer in the range [0, 255]. For images 80×64 pixels, it is necessary to store 5120 bytes, or 40960 bits. That is about 2^{3000} times the length of the example space 2^{1000} described in Chapter 3. It is, therefore, immensely larger, compared to the example studied by Kanerva. Nonetheless, Kanerva studied the space 2^{1000} just as an example. There should be no problem in using a space much larger. The relevant properties of the space arise from its large size, and apparently the larger the better, as explained in Section 3.2.

One possible solution to implement robot navigation is to associate each image with the next. In the SDM, that could be implemented storing image im_i at address im_{i-1}. When navigating, the robot would capture an image im_{j-1} and query the memory for the closest match. Supposing that im_{j-1} is similar enough to im_{i-1}, the SDM would make an accurate prediction of the image im_j from the input im_{j-1}. Using computer vision techniques, the system could then compute the necessary motor commands to move towards seeing image im_j. Such an approach is very straightforward from the point of view of the memory, and it is a purely vision-based model. However, the algorithms to compute motor commands from the images alone are not trivial, specially in the case of forward and backward motions. That would be difficult to run in real time. Therefore, it was deemed appropriate to store

Table 6.1: Summary of the total dimensions of the input vector.

Image resolution	Image bytes	Overhead	Total bytes	Total bits
80×64	5120	13	5133	41064
160×128	20480	13	20493	163944

additional information associated with the data vector, so that address im_{i-1} will store:

$$x_i =< im_i, seq\_id, i, timestamp, motion > \tag{6.1}$$

Thus, address im_{i-1} will store a vector x_i which comprises image im_i, a sequence number *seq_id*, an image number i, a timestamp and a character that describes the motion that the robot performed to move from im_{i-1} to im_i. Since the image im_i comes after im_{i-1}, sequence storage begins only when the second image is grabbed, because there is no address vector to store im_0. The sequence number *seq_id* is an auto-incremented, 4-byte integer, unique for each sequence. It is used to identify which sequence the vector belongs to. It is used mostly for disambiguation purposes and debugging. The image number i is an auto-incremented, 4-byte integer, unique for every data vector in the sequence. It is used to quickly identify every image in the sequence. It is used also for disambiguation and debugging purposes. The *timestamp* is a 4-byte integer, storing Unix timestamp. It is read from the operating system, and is also used as supplemental information for disambiguation and debugging purposes. The *motion* is a single character, identifying the type of movement the robot has performed to move from image im_{i-1} to image im_i. It can be 'f' for forward, 'l' if the robot was turning left, 'r' if the robot was turning right, and 'b' if it was moving backwards.

Since every pixel is stored as an 8-bit integer, an 80×64 image needs 5120 bytes, and the additional information comprises 13 additional bytes. Therefore, each data vector contains a total of 41064 bits. For comparison purposes, the software was also implemented to support images of resolution 160×128. In that case, each image uses 20480 bytes. Table 6.1 summarises the size of the data vectors in both situations.

Access circle

As described above, the address vectors used in those implementations of the SDM are images. The smallest image has 5120 pixels and is 40960 bits long. Thus, the addressable space is at least 2^{40960}— much larger than Kanerva's example of 2^{1000}. The total number of addressable locations is a number of several hundred digits[1]. That is no problem, for the SDM is suitable to work with very high dimensional

[1]Numbers such as 2^{40960} are not computable with ordinary computer software. The large numbers presented were computed using Aribas, which implements algorithms described in [26]. The software is available at http://www.mathematik.uni-muenchen.de/~forster/sw/aribas.html (last checked 2010.01.02).

spaces. The only problem may be of practical concern. Such a memory, during normal operation, takes several milliseconds to produce results, since it will be simulated in software using an ordinary laptop computer, as described in Section 5.1.2.

In the case of the input being an image 40960 bits long, the statistical distribution of N can be approximated by a normal function with parameters $\mu = \frac{40960}{2} = 20480$ and $\sigma = \frac{\sqrt{40960}}{2} = 101$, as explained in Chapter 3. Due to the tendency to orthogonality, most of the space lies in the interval $[\mu - \sigma, \mu + \sigma]$, and 99.9999% of the space lies in the interval $[\mu - 5\sigma, \mu + 5\sigma] = [19975, 20985]$. Therefore, a circle with radius less than 19975 bits will encircle only a very small, marginal portion of the binary space. As an example, the area of a circle with radius 19500 bits is:

$$A_c = \binom{40960}{19500} + \cdots + \binom{40960}{0} \approx 2.79 \times 10^{12308} \tag{6.2}$$

Therefore, A_c is approximately 1.8×10^{-20} per cent of the space.

The control software developed supports a variable radius for the access circle, and also another algorithm to compute similarity between two addresses using an arithmetic distance. The latter approach is inspired by Ratitch et al.'s method, described in Section 3.6.6. Since all the elements in the input vector can be read as 8-bit integers, the absolute sum of the arithmetic differences of those integers is used as a similarity measure. That implementation will be explained in more detail in Section 6.1.3.

Memory capacity

According to what was explained in Section 3.3.1, to prevent saturation the memory must not grow beyond its limits. Each hard location must represent about the same number of locations as the number of locations in the access circle. In the case of an access radius of 19500 bits, it is necessary to ensure:

$$2^x = A_c \Rightarrow x = log_2 A_c \approx 40887 \tag{6.3}$$

Therefore, each hard location represents about 2^{40887} points of the space 2^{40960}, or about 1.06×10^{-20} per cent of it. To enforce this condition, the number of hard locations, y, should be about 1.05×10^{-22} of the total number of locations:

$$\frac{y}{2^{40960}} = 2^{-40887} \Rightarrow y = 2^{73} \tag{6.4}$$

As Equation 6.4 shows, under normal conditions the memory can hold up to $y = 2^{73}$ hard locations, which can account for several gigabytes of information (see Section 3.3.1 for more details on these calculations). That number can even grow, if a smaller radius is used for the access circle. It has to shrink, if the access radius is increased. However, the addressable space seems already rich enough for normal operation under a variety of scenarios.

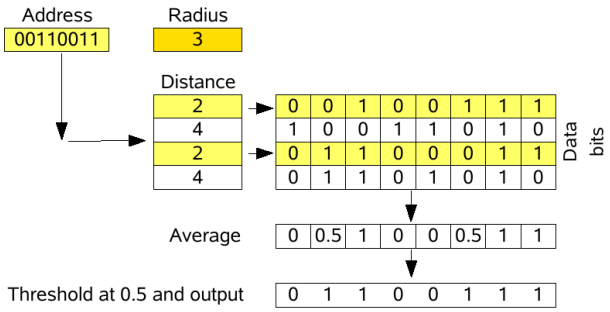


Figure 6.1: Bitwise autoassociative SDM.

6.1.2 Bitwise implementation

One of the models implemented for the tests with the SDM is called "bitwise model", since it does not use bit counters, but a single bit for each bit of the data vector. Additionally, the memories implemented are autoassociative: the same vector is used simultaneously as datum and address, in order to save computer memory in the software simulation. Figure 6.1 shows the architecture of the bitwise memory model, and the working details are described in this subsection.

Using bits instead of bit counters

Kanerva's model has a small handicap: the arrays of counters are hard to implement in practice and require a lot of processing when the memory has to be simulated in traditional computers, as is the case of the present implementation. They also increase the processing time, which is a considerable disadvantage for real time applications.

Furber et al. [29] claim their results show that the memory's performance is not significantly affected if a single bit is used to store one bit, instead of a bit counter. For real time operation, that simplification greatly reduces the need for processing power. It is described in more detail in Section 3.6.7. Writing in such a model is much simpler: it is done by simply replacing the old datum with the new datum. Furber uses an OR operator, so that the memory never forgets once it has learnt a 1. However, that approach does not seem appropriate in the long run. Actually, *natural forgetting* is one key characteristic of the SDM. Therefore, in the present implementation, the write operation erases the old data that may exist in the affected hard locations. An additional characteristic of this model is that, since it does not use bit counters, each data location consists only of zeros and ones. Hence, for the read operation, when summing the values columnwise, the average value for each bit can only be a floating point number in the interval [0, 1]. Therefore, the threshold for bitwise operation can be any floating point value in the interval]0, 1[. The value 0.5 should be an appropriate choice under normal conditions, and it is the value used.

Autoassociative memory

To store a sequence of images in the memory, using the original design described in Section 6.1.1, image im_i should be stored at address im_{i-1}. According to that non-autoassociative version of the SDM, each image has to be stored twice: the first time as datum, the second time as address. The only exceptions are the first and the last images, which are only stored as address and datum, respectively.

In order to save computer memory, it was deemed appropriate to use an autoassociative version of the memory, so that the same vector is used simultaneously as datum and address. Image im_i is stored at address im_i, along with the corresponding overhead information as shown in Equation 6.1: sequence number, image number in the sequence, timestamp and motion. To speed up the retrieval process, the motion stored with the image im_i is not the motion that lead to it, but the motion that the robot performed after grabbing it. Thus, all but the last one image can be stored in the SDM, working simultaneously as address and data and cutting the memory requirements to about 50%. During the autonomous run, the system has to grab an image, query the memory for the closest match and retrieve the associated motion command. By executing the same motion command again, it should be able to follow the same path it has learnt.

As described in Section 6.1.1, the data vectors contain the image and an additional overhead of 13 bytes. Those overhead bytes are unknown at the time of the autonomous run, except perhaps for the sequence number. Therefore, they are a kind of noise to the address. They could be set at random, as Kanerva suggests. According to the theory, 20% of the bits of the vector should suffice for correct retrieval, assuming non-coincident bits are random noise with a probability 0.5 of being correct. For the input $im_j + overhead$, with the overhead bits set randomly, the image's noise level could be, at most, slightly below 20% of the bits, which is perfectly acceptable for normal operation of the robot. However, it was considered preferable to implement yet another change to the model, instead of filling the overhead bits with random information. The software was set up to calculate the similarity between just the image part of the vectors, ignoring the remainder bits. The image can be regarded as the address, and the overhead the data. That saves computational power in the simulated implementation, and at the same time contributes to reduce the influence of random data in the result. The tolerance to noise in the images should increase a little bit.

Randomised reallocation

Another difference in the present implementation, relative to Kanerva's proposal, is that the virtual space is not filled placing hard locations randomly in the addressing space in the beginning of the operation. Instead, Ratitch et al.'s Randomised Reallocation algorithm is used. As described in Section

3.6.6, RRA works as follows: start with an empty memory, and allocate new hard locations when there is a new datum which cannot be stored in *enough* existing locations. The new locations are allocated *randomly* in the neighbourhood of the new datum address.

To use the random reallocation algorithm, it is necessary to decide when to introduce new hard locations, and when the existing hard locations are enough. The introduction should occur when a new datum to be learnt activates only a small number of hard locations, below a given threshold (which can be zero). In that situation, the difference between the number of active hard locations and the number of required hard locations is the number of new hard locations to be added. That threshold is a variable parameter of the present implementation. Values from 1 to 20 were tested. Obviously, the higher the threshold, the slower the memory operation and the faster it reaches saturation. On the other hand, a large threshold is also a way to enforce distribution of the data, thus making the memory more robust. The next chapters describe some tests that involve different values for the threshold and discuss its influence in the results.

6.1.3 Arithmetic implementation

The arithmetic version of the SDM, inspired by Ratitch et al.'s work, is described in more detail in Section 3.6.6. In that variation of the model, the bits are grouped as integers, as shown in Figure 6.2. The similarity between vectors is computed using an arithmetic distance, instead of the Hamming distance. Namely, the distance between address vectors x and y is computed as the sum of the absolute differences between all the k bytes[2]:

$$d(x, y) = \sum_{k=0}^{n} |x_k - y_k| \tag{6.5}$$

Learning is achieved using a kind of reinforcement learning algorithm, through a gradient descent approach. When writing to the memory, the following equation is applied to update every byte value of the selected locations:

$$h_t^k = h_{t-1}^k + \alpha \cdot (x^k - h_{t-1}^k), \quad \alpha \in \mathbb{R} \wedge 0 \leq \alpha \leq 1 \tag{6.6}$$

In the equation, h_t^k is the k^{th} 8-bit integer of the hard location, at time t. The value x^k is the corresponding integer in the input vector x and α is the learning rate. The speed of learning can be controlled changing the value of α. If $\alpha = 0$, that means to keep the previous values unchanged, so that the system never erases a datum it has learnt. If $\alpha = 1$, that implements one shot learning, and also rapid forgetting of a datum when another datum is to be written into the same hard location. The previous value of the hard location is overwritten.

[2]Note that the approach followed in the present work is intentionally different from that described by Ratitch et al.

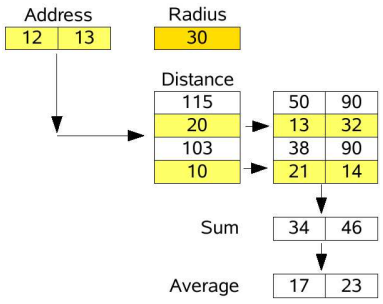


Figure 6.2: Representation of the arithmetic SDM model.

As Figure 6.2 shows, the version of the arithmetic memory implemented is also autoassociative. There is a single array of bytes that are used simultaneously as addresses and data.

6.2 Dealing with noise

One of the most difficult tasks when working in computer vision is that of dealing with noise in the images. In the present implementation, the typical noise levels were estimated, and the access radius is set in function of those noise levels. That is an elegant way to minimise the problem, taking advantage of a natural feature of the SDM.

The problem of noise

Noise is an important problem to deal with in vision-based systems. Even images taken with high-quality cameras, always present a huge level of noise. That noise can be caused by many different sources. Electronic noise in the camera's sensor is one important source. Small illumination changes, for example, also make shadows and reflexes have a different fingerprint in the images. Even communication errors leave a trace in the image by changing pixel values. The human brain and optical structure can filter out large amounts of noise in a natural way. Thus, "normal" amounts of noise pose no visual problem for humans. But for robots it is a serious computer vision problem. Since Lizard navigates based on vision, the problem of noise in the images may significantly affect its performance. Figure 6.3 shows 3 different images captured by the robot while following a path. As the figure shows, the images are not free from noise. Hence, some experiments and studies were done in that particular aspect.

The impact of noise in navigation will be noticed when the system has to compute the similarity between two images. If there was no noise, two images taken consecutively without moving the robot or any environmental change would be exactly equal, pixel by pixel. The error in similarity, as defined in Equation 3.13, would be zero. However, it is hardly so under normal circumstances. The distance between any two different pictures is rarely zero, even if the difference is just normal noise. Equalisation

(a) Image 13. (b) Image 32. (c) Image 55.

Figure 6.3: Images captured by the robot while following a path.

and contrast stretching, as described in Section 5.2.2, contribute to minimise the problem, but do not eliminate it completely. J. Vanhala et al. [101], as described in Section 3.6.9, used a very straightforward approach. They use only the most significant bit of each pixel. In natural binary code, the significance of the bits decreases by powers of two as one moves from the most significant to the least significant bits. Therefore, the least significant bits are the ones that carry most of the noise, and less of the relevant information. Hence, under normal circumstances, they can safely be ignored without compromising the performance of the system. Anyway, Vanhala's approach is very radical. It is not a wise choice, specially if to use with the Hamming distance, as in the bitwise operation mode. In the bitwise mode the distance between memory items is computed based on the number of bits in which the vectors differ. Cutting out all but the most significant bit, the addressing space is considerably reduced. That problem is discussed in more detail in Section 7.1.1.

Estimating noise values

According to the theory, the SDM should be very tolerant to noisy data. However, it was deemed appropriate to have an idea of the magnitude of the problem, and the impact of image processing in aiding solving it. Therefore, different tests were carried to measure the noise levels. Noise is defined as the error in similarity between two consecutive images, taken without moving the robot and maintaining the scenario and environmental conditions unchanged. The numeric value of the error depends on the operation mode of the SDM. In the arithmetic mode it is computed using Equation 6.5, while in the bitwise mode it is computed using the Hamming distance, as defined in Equation 3.13.

The results of the tests are summarised in Table 6.2. Since the noise values suffer so large variations, the table summarises the results obtained over a series of 31 images. A total of 30 noise levels were computed. The table shows the largest noise level, the smallest, the average and also the standard deviation measured for the 30 runs of each experiment. The results are presented for the bitwise and for the arithmetic operation modes. The table compares the results obtained without any image processing, with equalisation, with contrast stretching, and finally with equalisation and then contrast stretching.

As the table shows, in the bitwise mode, without image processing the average noise value is close

Table 6.2: Noise values measured. The results are the average of 30 runs.

		No processing	Equalisation	Contrast Stretch	E + CS
Bitwise	MAX	5881.00	6522.00	8080.00	6475.00
	AVG	5471.43	6232.90	6865.33	6273.37
	MIN	5201.00	6031.00	5741.00	6082.00
	STD	175.44	120.57	578.14	104.62
Arithmetic	MAX	9795.00	16448.00	28568.00	16297.00
	AVG	8701.20	15114.20	15508.80	14804.30
	MIN	7264.00	13883.00	9590.00	12993.00
	STD	588.92	649.7	4317.65	841.46

to 5500 bits. Equalisation and contrast stretching push that value to more than 6000 bits. That is an expected result—both techniques enhance image contrast, so that dark images become clearer, and clear images become darker. That also amplifies the noise values, since the range of pixel values is wider in the processed image. The same effect can be noticed in the arithmetic mode, where the average noise level jumps from about 8700 to about 15000. It can also be noticed that specially contrast stretching has a very large impact on the standard deviation. Equalisation has a much smoother impact.

Comparing the processing techniques, equalisation and contrast stretching, it is noticed that contrast stretching has a larger impact in the noise. The noise is amplified in both cases, but contrast stretching amplifies it more. Additionally, from the results obtained in Section 5.2.2, it can be noticed that equalisation produces visually more appealing images. It is definitely a better algorithm to use for the intended goal, compared to the contrast stretching alternative.

Still from Table 6.2, it can be seen that, in the bitwise mode, the noise can be up to almost 6600 bits, in the equalised image. That value is clearly acceptable, for normal memory operation. Even 7000 bits is only about 17% of the whole address vector, which contains 40960 bits. That number also means that for robot navigation using a view sequence one must not store images which differ less than 7000 bits from their predecessor, since they will actually be processed as if they were the very same image. In the arithmetic mode, the numbers are very different, but the conclusions are the same. Both image processing techniques amplified the noise levels, and contrast stretching amplified the noise a little more than equalisation alone. In that case, the maximum error in similarity obtained, with equalised images, was close to 16500.

Use of a dynamic access radius

All the noise levels are, in theory, in the *acceptable* range for normal operation of the memory. However, they exhibit a huge standard deviation and are very dependent on the illumination, diversity of environmental patterns and other variables that always influence vision-based robots. Noise levels can be

very low if the scenario is monotonous and dimly illuminated, and very large if the scenario is rich and well illuminated. Although equalisation minimises those variations, it cannot solve them completely, as Table 6.2 shows.

In all the tests performed, it was determined that the maximum noise level was always within 2.5 standard deviations of the average value, in a diversity of scenarios and illumination conditions. It was also found out that it was less than 15% above the average, for bitwise operation, and less than 30% above average for arithmetic operation, in all the cases analysed, except for the contrast-stretched images. The latter approach is the one being used, for it is mathematically simpler to compute in real time a percentage than a standard deviation.

To make navigation a little more robust, it was deemed appropriate to use a method of calibrating the system, estimating noise levels before the robot starts operation. A dynamic algorithm was implemented, to automatically adjust the access radius to the noise level when the system is turned on. Once the system is turned on, Lizard captures 3 consecutive images and computes the difference between the first and the second, and between the second and the third. The average of the two values is taken as a good approximation to the actual average noise level. To make sure the images are retrieved from the memory when following the sequence, the access radius was set as a function of that noise level. It was set to the average value of the noise increased 15% for bitwise operation, and 30% for arithmetic operation. Therefore, images which differ more than those values are different enough and are stored into the memory. That approach works well for unprocessed images, or for equalised images. Contrast-stretched images, if not equalised, may not be recognised correctly due to the presence of noise.

6.3 Robot navigation

Robot navigation, in the present work, involves three major features: supervised learning, autonomous navigation and ability to correct small drifts. During supervised learning, the robot is commanded by a user and records navigation information. During the autonomous navigation step, Lizard follows learnt paths using information stored into the SDM. Small lateral drifts are detected using image matching techniques.

Path learning

Lizard learns paths during a supervised learning stage. During that stage, the user navigates the robot through the graphical interface. In that interface, the user clicks the buttons to drive the robot forward, backwards, right or left. During learning, for each step forward or backwards the robot moves

approximately 2.25 cm. For each step left or right, it turns about 2.4°.

Any new path is automatically assigned a name and sequence ID, both unique for each sequence. After each move, the robot captures its current view and sends it to the computer. The computer calculates the distance (error in similarity) between that view and the last stored view. If the error is smaller than the SDM's access radius, then that image is discarded, because it is considered too close to the last stored image to be necessary, or even useful, for navigation. Otherwise, it is assigned an image ID and stored into the SDM and in the computer disk for later use in localisation and navigation tasks.

When the user signals the end of the path, the robot resets the sequence and image numbers and returns to the waiting state, in which it can accept any new task.

Autonomous navigation

The robot enters the autonomous navigation mode when requested by the user. After entering that mode, Lizard captures an image of the environment and queries the memory for the closest matching image. The data of the best matching image is used for localisation. The robot assumes to be at a point very close to that where it captured the best matching image. Therefore, it suggests the user to follow that path until the end. Otherwise it goes back to a waiting state.

The first version of the implemented navigation algorithms is called "Basic Algorithm" (BA). It is described in Algorithm 3. For better localisation precision, during the autonomous run the robot moves in steps 1/16th the length of those steps used during the learning mode.

After each step, the algorithm verifies if the robot has been kidnapped. The "kidnapped robot" problem happens when the robot is suddenly transposed from one place to another. That may happen because someone has grabbed it and moved it to another place, because it has slipped for a considerable distance, or because of some other unexpected occurrence. When the robot is kidnapped, it is expected to figure out that it has been moved to a new position and resume normal operation under the new environment. The implemented Basic Algorithm detects that the robot has been kidnapped by comparing the ID of the image predicted in the last iteration with the ID of the image predicted in the new iteration. If the absolute value of the difference is above a given threshold (usually set to 3), then it is assumed that the robot has been kidnapped. It may also happen that the robot has been kidnapped from one sequence to another. That is detected by comparing the sequence ID of the predicted image with the ID of the sequence that was being followed. If they are different, there is a high probability that it has been kidnapped (the alternative being that there was a prediction error, or some other unexpected error). In a kidnapping situation, the algorithm may be configured to continue following the sequence, warn the user or just stop. The behaviour depends on what has been decided by the user.

Algorithm 3: Basic navigation

begin

$grabbedImg \longleftarrow$ current view;

$newImg \longleftarrow$ SDM output based on $grabbedImg$;

$sID \longleftarrow$ sequence number of $newImg$;

$imgID \longleftarrow$ image number of $newImg$ in sequence sID;

$lastInSequence \longleftarrow$ length of sequence sID;

repeat [follow while getting closer to the target]

Verify if kidnapped;

Warn user if $newImg$ predicted for the 5^{th} time or (error in similarity increases more than 35% and the robot has not been kidnapped);

$command \longleftarrow$ retrieve command from $newImg$;

Execute $command$;

if $command \neq MOVE\_RIGHT$ *and* $command \neq MOVE\_LEFT$ **then**

Correct lateral drift;

end

$grabbedImg \longleftarrow$ current view;

$newImg \longleftarrow$ SDM output based on $grabbedImg$;

$imgID \longleftarrow$ sequence number of $newImg$;

until $imgID \geq lastInSequence$;

end

Correcting lateral drifts

In Algorithm 3, MOVE_LEFT and MOVE_RIGHT are the commands to turn the robot to the left and to the right, respectively. It was experimentally determined that most of the times correcting lateral drifts after a turn (yaw) movement could result in long drift correction loops, which in some cases might be infinite. Therefore, it was deemed appropriate not to correct drifts in those cases. If, after turning, there is a small error in the orientation of the robot, it will most likely be corrected during its forward or backward motion. But after each step forward or backwards, the Lizard tries to adjust its heading. That is done running Algorithm 4, to determine if there is a significant lateral drift, and correct it if necessary.

Algorithm 4: Correct lateral drift

```
begin
    D ⟵ Compute lateral drift;
    count ⟵ 0;
    lastD ⟵ 9999;
    while (|D| ≥ MIN) and (|D| < lastD) and (count < MAX) do
        Turn in the direction that corrects D;
        newImg ⟵ Capture new image;
        Compute lateral drift D using newImg;
        lastD ⟵ |D|;
        count ⟵ count + 1;
    end
end
```

In Algorithm 4, the lateral drift D is computed using the approach described in Section 4.3.2. If D is above the threshold value MIN, then it is corrected by turning the robot in the opposite direction, 1/16 of the normal 2.4° step. In the present implementation, MIN was set to 5 pixels. Drifts less than 5 pixels are ignored. It was noticed that in almost all situations, errors below 5 pixels are extremely difficult to correct: the robot would turn right and left many times, but the error would not decrease. In another improvement to avoid long loops that result in no meaningful improvement, if the drift increases at some point of the correction, the algorithm stops.

The number MAX is the maximum number of iterations, and it is also used to prevent the robot from performing long correction loops. Those loops are not very likely to happen, because the correction loop will also end as soon as the drift increases relatively to the last iteration. Anyway, just for safety, the MAX value is set to 5, thus limiting the number of correcting movements to 5 before getting

back to the main loop (Algorithm 4). It should be noted that the MAX number does not impair the robot's performance in any way. If the correction requires more than MAX steps, the main algorithm (Algorithm 3) will call the correction method again in the next movement and more correcting turns will be performed, again up to a maximum of MAX steps. That makes the robot correct heading as it moves in a smooth way, along many short and elegant correction loops, instead of long *unpleasant* turning sessions without visible progress.

6.4 Tests and preliminary results

In order to have a better insight into the performance of the navigation algorithm, its performance was assessed following some simple paths in the testbed. The results show the different performances of the robot with and without image equalisation, as well as the impact of using the arithmetic or bitwise implementations.

6.4.1 Processing time

To measure the processing time of each algorithm cycle, the memory was loaded with 100 vectors of the format represented in Equation 6.1. Distribution of the data was enforced, so that each vector was stored into at least three hard locations (using the Randomised Reallocation Algorithm, described in Section 3.6.6). Therefore, the memory had a total of $100 \times 3 = 300$ hard locations.

The search of the closest matching image (normal size, 160×128 pixels) takes about 230 ms in bitwise mode and about 240 ms in arithmetic mode. It should be mentioned that the bitwise operations take longer because they are simulated in software. A hardware implementation should provide much faster results for the bitwise operation. Nonetheless, those measures confirm that real time operation of such a memory requires the use of a fast processing unit, or appropriate hardware.

The system as a whole also has other time delays, when operating in real time. Communications, for example, are a bottleneck to the speed of the system. It usually takes about 210 ms to grab a normal image, and 100 ms to grab a small image of 80×64 pixels.

Table 6.3 shows examples of processing times necessary to run one full iteration of the navigation algorithm, including all steps: capture an image of the current view; find the best match in the SDM; decide the next command; and send the command to the robot. The images were neither equalised nor contrast stretched when those times were recorded. The results shown are the average of 20 iterations. It can be seen that, using small images, the system is able to process about one image every 0.8 seconds, or about 1.25 frames per second. Using the larger images, it takes about 1.1–1.2 seconds per image, which

Table 6.3: Processing time (in milliseconds), when the memory was loaded with 3 copies of 100 vectors (total of 300 hard locations).

	Small image	Normal image
Arithmetic mode search	50	230
Bitwise mode search	51	240
Grab, transmit & receive image	100	210
Other tasks	647	786
Arithmetic mode total	797	1226
Bitwise mode total	798	1236

means that the maximum frame rate should be less than 0.9 frames per second, for that hardware under those conditions. It should be noted that most of the time is wasted in communication with the robot and other tasks. Those other tasks include deciding and executing the next command, converting the image from JPEG into PGM, verification of imminence of collision and other precautions and routine tasks.

6.4.2 Illumination changes

The effect of image processing was also tested, in order to analyse the advantages of using algorithms to improve the quality of the images grabbed from the camera. Although simple, the processing also adds a few microseconds to the processing loop, which is an additional cost to account for in real time operation. Hence, it is only justified if it is useful for improving the performance of the system, making it more robust to illumination changes.

In order to assess the performance of the system under different illumination conditions, the robot was taught one path under a strong light source, and then it was tested under two different illumination conditions. All the experiments were done in a $4 \times 7\ m^2$ room. The room was closed in a way that there was no significant penetration of light from the outside. The relative position of the bulb, projector and large objects was maintained during the experiments. Granted those conditions, the experiments were carried on:

1. Under strong light. That included ambient light and fill light. The ambient light is provided by an 11 W compact fluorescent bulb, suspended from the ceiling of the room. The experimental setup was located at a corner. The fill light comes from a 300 W projector, using a halogen bulb. That projector was directed towards the scenario at the height of 1.2 m, from one side of the arena.

2. With only ambient light.

The results were obtained using a sequence of 80 images. Those images were captured under a strong light and stored into the computer's hard disk without processing. The robot was then placed at the beginning of the path, where it was expected to capture image 1, and the various tests were run. Table 6.4 compares the performance of the memory with and without application of image processing algorithms. CS means Contrast stretching. EQ means Equalisation. CE means that both Contrast stretching and Equalisation were used. The third column specifies the operation mode: BW for bitwise, AR for arithmetic. The fourth column shows the distance (error in similarity) between the robot's captured image and the closest matching image retrieved from the memory. The fifth column shows the distance to the second best matching image. The sixth column indicates how larger the distance to the second best match is, compared to the first best match. The seventh column shows the average distance to all the hard locations. The eighth is a calculation of how larger that distance is, compared to the best matching image. Finally, the last column shows which image was predicted by the SDM output.

It can be seen that there were some prediction errors. The first image was the expected (correct) result in all the cases. But, as the table shows, under dim light, without image processing, image 10 was predicted in the arithmetic mode, and image 53 was predicted in the bitwise mode. Also, in the bitwise mode with contrast stretching, image 2 was predicted instead of image 1. Under normal circumstances that could be an acceptable result, but it is an error anyway.

As the table shows, image processing in general improves the results and the performance of the robot under dim light. First, there are prediction errors in non-processed images under dim light. Those errors do not occur with equalised images. There is still an error in bitwise mode with contrast stretched images, but that can be considered a *minor* error, since the image predicted is the next to the right one—it is actually as close to the right prediction as it could be.

Another conclusion that can be drawn from the results is that image processing, in general, in the bitwise mode decreases the absolute values of the errors: the images become closer to each other in the binary space. In the arithmetic mode, the effect is the contrary: the images become more distant after processing. However, the most important result is that with processed images, the average distance between the current view and the pool of images in the SDM increases proportionally more than the distance between the current view and the first prediction. Therefore, for processed images, the correct prediction is easier to retrieve from the others. That effect is clearly visible in the results obtained under dim light. The average distance from the input image to the pool of images in the SDM is just 27.08% larger than the distance to the first prediction in the arithmetic mode, and 4.72% in the bitwise mode. Equalisation and/or contrast stretching more than double that difference in the arithmetic mode, and greatly increase it in the bitwise mode.

The results summarised in Table 6.4 clearly show that image processing, specially through equali-

Table 6.4: Matching errors for different image processing algorithms.

	Proc.	Mode	Dist. to 1^{st}	Dist. to 2^{nd}	inc. (%)	Dist. to Average	inc. (%)	$imID$
Strong light	–	AR	240342	266458	10.87	440792.10	83.40	1
	CS	AR	271550	300208	10.55	539367.31	98.63	1
	EQ	AR	401707	419718	4.48	709934.27	76.73	1
	CE	AR	400324	418955	4.65	709363.06	77.20	1
	–	BW	36912	37469	1.51	39758.20	7.71	1
	CS	BW	35890	36178	0.80	39168.33	9.13	1
	EQ	BW	35369	35732	1.03	39352.21	11.26	1
	CE	BW	35311	35723	1.17	39367.54	11.49	1
Dim light	–	AR	662573	665912	0.50	842004.94	27.08	10
	CS	AR	311795	326919	4.85	577167.11	85.11	1
	EQ	AR	454176	461733	1.66	712281.14	56.83	1
	CE	AR	452801	460476	1.70	711719.65	57.18	1
	–	BW	39579	40224	1.63	41448.90	4.72	53
	CS	BW	36436	36551	0.32	39529.25	8.49	2
	EQ	BW	37358	37580	0.59	39559.84	5.89	1
	CE	BW	37269	37276	0.02	39584.21	6.21	1

sation, succeeds in increasing the matching error between the correct prediction and the other images in the memory. That will make the system more robust and reduce the probability of prediction errors, specially in the case of changing illumination conditions.

6.4.3 Full sequence tests

After the tests described in the previous sections, Lizard was made to follow a complete path, guided by a sequence of images previously learnt. Figure 6.4 shows an overall view of the paths followed during a test run[3]. The lines were drawn using a pen attached to the rear of the robot. Therefore, they mark the path described by its rear, not its centre. That explains why there are small arcs when the robot changes direction. The black line shows the path that was taught during the learning stage. All the data grabbed from the robot, while learning the path, was recorded into the computer hard disk, so that the same exact information of the path could be loaded again into the system memory quickly, whenever necessary. That procedure made it possible to test autonomous runs both with and without preprocessing of the images, as well as test different operating parameters. The whole path is described by 150 small-size images. Unless stated otherwise, all the results presented from this point onwards are

[3] At that point of the work, the arena was the smaller-size one. That is why the robot looks so large compared to the white floor.

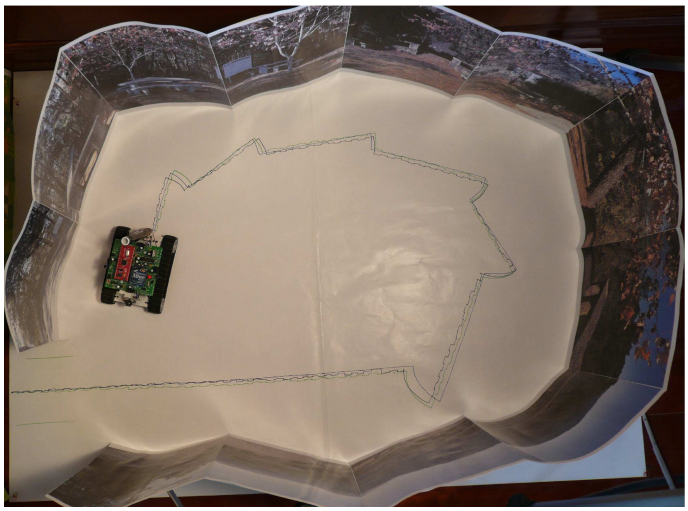


Figure 6.4: Overview of the experimental platform and preliminary results.

for small-size images.

In Figure 6.4, the green line shows the path followed during an autonomous run, in which the images were equalised and compared using the arithmetic distance. The blue line shows another path followed during an autonomous run, but that time the images were not preprocessed. The similarity measure was still the arithmetic distance. Although there are small differences in the paths, those are hardly enough to draw any conclusions. As for the bitwise similarity measure, the results are very similar—the robot is able to accomplish its task successfully. It can be affirmed that all the approaches are robust enough to make the robot follow the same path, under those conditions.

6.5 Arithmetic and Hamming distances

As shown in the previous sections, it is possible to navigate a robot using a sequence of images stored into a SDM. Image processing, namely equalisation, helps improving image quality and makes the system more robust. Additionally, two methods can be used to compute the similarity between images: an arithmetic distance, and a Hamming distance. The method used can also influence the results, making the system more or less robust. This section compares the two methods, and assesses their impact on the performance of the system.

6.5.1 Comparing distances

The Hamming distance is the one more consistent with the theory. It is the method proposed by Kanerva for the original SDM model. The arithmetic distance, as proposed by Ratitch, though, is more successful in separating the images, so it may produce better results by making the system more robust

Table 6.5: Comparison between arithmetic and Hamming (bitwise) distance.

	$imID$	Distance to 1^{st}	Distance to 2^{nd}	Inc_2 (%)	Distance to average	Inc_A (%)
Arithmetic	1	13610	134019	884.71	178417.75	1210.93
	6	133469	143573	7.57	171896.58	28.79
	40	117683	147990	25.75	171524.35	45.75
	59	151764	153925	1.42	182483.96	20.24
	65	130505	164335	25.92	185011.20	41.77
Average		109406.20	148768.40	35.98	177866.77	62.57
STD		54917.62	11336.36		6093.73	
Bitwise	1	6170	9421	52.69	9835.55	59.41
	6	9388	9446	0.62	9800.45	4.39
	40	9338	9556	2.33	9779.13	4.72
	59	9601	9668	0.7	9880.83	2.91
	65	9094	9601	5.58	9905.68	8.93
Average		8718.20	9538.40	9.41	9840.33	12.87
STD		1435.85	104.1		53.08	

to noise and interferences.

Table 6.5 shows a comparison of the differences between various images using the two measures. In the experiment, the memory was loaded with a sequence of 69 small (80×64) images, which were equalised before being stored into the memory. Lizard was then put in five randomly chosen points of the path, and the difference between its current view and all the other images stored into the memory was calculated: first the difference to the best matching image (third column), then the difference to the second best matching image (fourth column), and then the average distance to all the images stored into the SDM. As the table shows, the arithmetic mode exhibits the best results. The difference from the right image to the other images is larger—always above 20% more, while in the bitwise mode it is only larger by a narrow margin. Figure 6.5 illustrates that characteristic. The bars plotted for the bitwise operation mode are just about the same height, while in the arithmetic mode there is a much larger difference.

It is noticeable that the distance to the best matching image of image 1 is well below average using both methods. It is 13610 for the arithmetic mode, and 6170 for the bitwise mode, while the average of the distances to the best predictions are, on average, 109406 and 8718, respectively, for the arithmetic and bitwise modes. The most probable reason for that behaviour is that the robot is always placed in the same position at the beginning of the sequence. For the points where it captured images similar to images 6, 40, 59 and 65, it was not placed so precisely. At the home position (beginning of the sequence), the robot was placed tight against small boxes stuck to the floor, while for the other positions there were only marks drawn in the paper. Therefore, the distances to the closest matching images are much

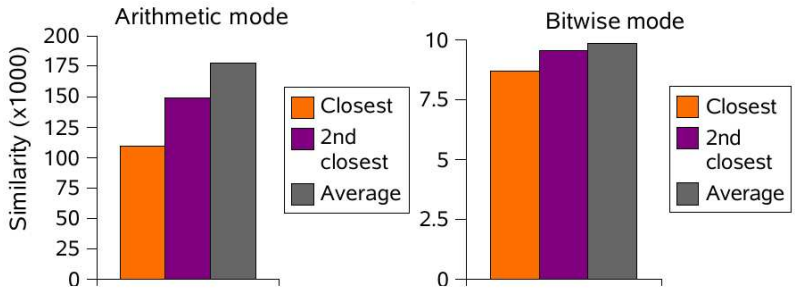


Figure 6.5: Summary of arithmetic and Hamming distance measurements.

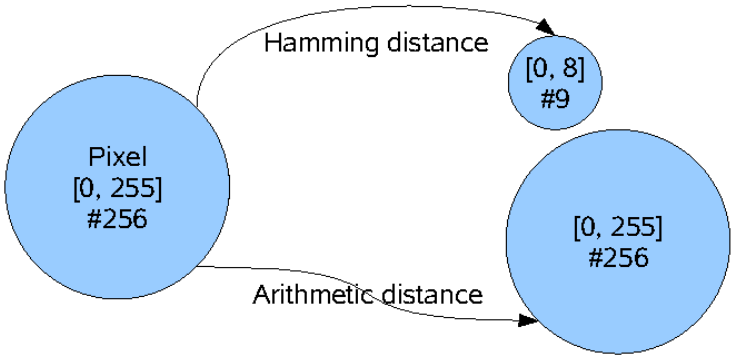


Figure 6.6: Domain and codomain of the arithmetic and hamming distance functions.

larger in those cases.

6.5.2 Comparing mapping functions

Tables 6.4 and 6.5 clearly show one fact: the average arithmetic distance between an image and the pool of images in the SDM is about 18 times larger than the equivalent Hamming distance. That fact is not surprising, considering that the arithmetic distance function maps distances onto a much larger codomain. For every single image pixel, which is represented by a byte number, the Hamming distance is a number in the interval [0, 8]. The arithmetic distance is a number in the interval [0, 255]. The cardinality of the codomain of the Hamming distances is 9, while the cardinality of the codomain of arithmetic distances is 256. Figure 6.6 illustrates the idea.

Let's consider the case of the small images, with 80×64 pixels resolution. They have exactly 40960 bits. The difference between a black image (all zeros) and a white image (all ones) is exactly 40960 bits. That is actually the maximum value that the Hamming distance can have. The Hamming distance is, therefore, a function with codomain [0, 40960] for the whole image.

As for the arithmetic distance, the codomain has the same cardinality as the domain. In that case, it is computed based on the sum of the arithmetic differences between image bytes. An image with

Table 6.6: Comparison between Arithmetic Distances (AD) and Hamming Distances (HD).

	No processing		Contrast Stretch		Equalisation	
	Distance to 1^{st}	Distance to Average	Distance to 1^{st}	Distance to Average	Distance to 1^{st}	Distance to Average
AD	240342	440792.1	271550	539367.3	401707	709934.3
HD	36912	39758.2	35890	39168.3	35369	39353.2
AD/HD	6.51	11.09	7.57	13.77	11.36	18.04

resolution 80×64 has 5120 bytes. Each byte contains an integer in the interval [0, 255]. Therefore, in that case the difference between a black image (all the pixels are zero) and a white image (all the pixels are 255) is $255 \times 5120 = 1305600$. Hence, the arithmetic distance is a function with codomain [0, 1305560].

In summary, the codomain of the arithmetic distances is much larger than the codomain of the Hamming distances. Indeed, it is 31.88 times larger. Thus, it could be expected that the arithmetic distances be, on average, about 32 times larger than the equivalent Hamming distances. However, they are much smaller, just 6–18 times larger than the Hamming distances.

Table 6.6 compares the values of selected arithmetic distances and Hamming distances. Those values were taken from Table 6.4. The second and third columns show values without image processing. The fourth and fifth show values obtained with images that were contrast stretched. The last two columns show values obtained with images that were equalised. In all the cases, "Distance to 1^{st}" is the error in similarity between the input image and the closest image in the SDM, and "Distance to Average" is the average distance between the input image and all the images in the SDM. As the table shows, without image processing, the Hamming distance to the closest image is 6.51 times smaller than the equivalent arithmetic distance. As for the average distance, the difference is even larger: 11.09 times smaller. Contrast stretch enhances the differences. Equalisation enlarges them even more. The table also shows that the average distance grows faster than the distance to the first. Therefore, image processing improves the results obtained, and it is expected that the arithmetic distance takes more advantage of it.

6.5.3 Momentary localisation errors

The distance between images, computed using the Hamming distance, is much smaller than the distance computed using the arithmetic distance. The gap between distances computed using the HD is also narrow (e.g., using the equalised images, the average distance is less than 40 000 bits, while the distance to the closest image is more than 35 000 bits). A consequence of that fact may be that in the bitwise mode the system has a higher probability of making a wrong prediction, retrieving the wrong image and

Table 6.7: Momentary localisation errors during autonomous runs following the same path.

	Arithmetic mode	Bitwise mode
Prediction errors	3	13

command to follow. To confirm that hypothesis, a new test was performed, that consisted in obtaining an estimative of the number of prediction errors during a sequence run. Due to the nature of the problem, it is very difficult to obtain a precise count of the number of prediction errors. However, there is a situation in which it is clear that one prediction error has occurred. That situation is defined as a "Momentary Localisation Error" (MLE).

A MLE occurs when the robot retrieves image im_{i-j} after having retrieved im_i, for $i, j > 0$. Under normal operation, when autonomously following a sequence, the robot will always go forward in the sequence. After seeing image im_i, it should never again see image im_{i-j}. Therefore, the $imID$ of the image predicted at time t should always be greater than or equal to the $imID$ of the image predicted at time $t-j$. If it is inferior, then one of the images must have been incorrectly predicted. Therefore, every time that happens, one prediction error can be counted. The only situations where the method fails are the cases of robot slippage, or if it is kidnapped while travelling. In those circumstances, the robot may retrieve im_{i-j} after im_i, and both predictions may be correct. But under normal circumstances the MLE measure, as defined above, should correctly count almost all the prediction errors.

It should be noted that the MLE captures the prediction errors both when the system goes back and when it goes further in the sequence. If at im_i the system predicts im_{i+j}, then it is expected to retrieve im_{i-j} soon, to get back to the correct point in the sequence, and the error is still counted. The MLE may, however, miss some wrong predictions in cases where there are two or more consecutive errors. For example, in the unlikely event that the predictions are ..., im_i, im_{i+j}, im_{i+j+k}, im_i, ..., only one MLE is counted, although both im_{i+j} and im_{i+j+k} were wrongly predicted.

In order to have an insight into the number of prediction errors that occur during normal navigation, the robot was made to follow a path described by 69 small images. The number of MLEs was recorded, and it is shown in Table 6.7. As the table shows, the number of errors is larger during bitwise operation. During the arithmetic run, the system went back to a previous image only three times during all the path. In the bitwise run, the same happened on thirteen different occasions. Although those errors do not actually seem to compromise the performance of the system under controlled operating conditions, they are, nonetheless, an undesirable vulnerability of it. Since the path is described by 69 images, during the autonomous run the robot is expected to perform at least $69 \times 16 = 1104$ steps. Therefore, the error rate is less than 1.18% in the bitwise mode, and less than 0.27% in the arithmetic mode.

6.6 Summary

This chapter describes the architecture and implementation details of two Sparse Distributed Memory models that were implemented to use in robot navigation. One of the models uses the Hamming distance to compute the error in similarity between two images. However, in that model the bit counters are replaced by single bits, to reduce memory and processing requirements in the simulation. The other model uses an arithmetic distance. The access radius used in both models is dynamic. It is computed before operation, in function of the amount of noise measured in the images. That makes the system adaptable to various illumination conditions. Tests were also performed, in order to have an insight into how image processing, namely through equalisation and contrast stretching, impact on the performance of the system. The results show that both methods improve the tolerance of the system to some illumination changes, and equalisation is superior to contrast stretching. In order to speed up processing in real time, it was decided to use only small images, and equalise them before processing. Unless stated otherwise, that is the procedure followed in further experiments.

After the initial tuning process, the basic navigation algorithm was designed. Then the robot was made to follow complete sequences, in which it succeeded. It is able to process about 1 image per second, due to the processing and communication times required. Comparing the performance of the robot in the bitwise and arithmetic modes, they seem very close. However, the bitwise mode causes much more Momentary Localisation Errors, which are defined as the situations in which the system retrieves an image of the sequence that is older than expected. That may be due to the fact that the Hamming distance function has a much smaller codomain than the arithmetic distance function.

The system presented so far is able to follow paths described by sequences of images and recognise places based on image similarity. It is immune to the kidnapped robot problem, due to the fact that localisation is only based on visual recognition of the place.

Chapter 7

Searching for a better code

Confidence in nonsense is a requirement for the creative process.
(M. C. Escher, Dutch graphic artist)

Contents

The previous chapter presented results of running the robot using two different memory implementations: the bitwise and the arithmetic. It was also noticed that the bitwise mode produces more momentary localisation errors. That may happen because the natural binary code (NBC) is not appropriate for processing that kind of information the way it is necessary to process it to make the SDM work, using the Hamming distance.

According to the theory, the properties of the SDM should be effective if the information is purely random. Images, however, are not truly random information, and the navigation errors are highly dependent on the methods used to compute the distance between two images, or the way the visual information is encoded, stored into and retrieved from the SDM.

This chapter analyses in more detail the problem of encoding the information to store into the SDM. Different methods were tested, and the results are presented and discussed. Part of this work has been published, by the author, for the first time in 2009 [72, 73].

7.1 Binary codes and the Hamming distance

Sensorial information is encoded by the robot using the natural binary code. In natural binary code, the value of each bit depends on its position in the binary number. But the Hamming distance does not account for that factor. Therefore, when using the Hamming distance to compute the similarity between memory items, it is actually a criterion different from the criterion used to encode the information. That problem is addressed in this section.

7.1.1 The problem of using the Hamming distance

In natural binary code the value of each bit depends on its position. The numbers 01_2 and 10_2 are different from each other, although the number of ones and zeros in each number is exactly the same. Additionally, the number of ones can increase and decrease as the represented value of the number grows. For example, 1000_2 contains less ones than 0011_2, but its binary value is more than twice the value of 0011_2. Those characteristics of the natural binary code make it inappropriate for using with the Hamming distance, because the Hamming distance does not account for the the binary values of the numbers involved. For example, the Hamming distances $hd(1000_2, 0100_2) = hd(1000_2, 0001_2) = 2$. However, in the first case, 1000_2 and 0100_2 are much closer to each other than 1000_2 and 0001_2, if they represent pixel values.

Table 7.1 shows the Hamming distances between all the 3-bit natural binary code numbers. As the table shows, the Hamming distance is not proportional to the arithmetic distance. The Hamming distance sometimes remains the same, other times even decreases, while the arithmetic distance increases monotonically as the numbers grow. One example is the case of 001_2 to 010_2, where the arithmetic distance is 1 and the Hamming distance is 2. Comparing with the distance from 001_2 to 011_2, the arithmetic distance increases to 2, but the Hamming distance decreases to just 1 bit. In total, there are 9 similar situations in the table. Those are *undesirable* situations, because the Hamming distance decreases while it should increase or, at least, maintain its previous value, to be proportional to the arithmetic distance.

The main problem in the present work arises from the fact that different criteria are being used to encode the information and to process it. The PGM images are encoded using the natural binary

Table 7.1: Hamming distances for 3-bit numbers.

	000	001	010	011	100	101	110	111
000	0	1	1	2	1	2	2	3
001		0	2	1	2	1	3	2
010			0	1	2	3	1	2
011				0	3	2	2	1
100					0	1	1	2
101						0	2	1
110							0	1
111								0

code, which takes into account the position of the bits to represent different values. But the Hamming distance does not consider positional values. The performance of the SDM, therefore, might be affected because of those different coding schemes being used to encode the information and latter to process it. In summary:

- The arithmetic distance produces good navigation results. The domain of arithmetic distances is actually much larger than the domain of Hamming distances, as described in Section 6.5. But the arithmetic distances are computed using groups of bits. It is not a bitwise operation. Thus, it is not according to the characteristics of the SDM, which were studied for a boolean space.

- The Hamming distance is not proportional to the arithmetic distance. However, the processes of storing and retrieving data from the memory rely strongly on it. Hence, different criteria are being used to encode sensorial information and to process it inside the SDM.

It may happen that those characteristics of the binary code and the Hamming distance may be neglectable when operating with random data, as studied by Kanerva. If the data are random, the impact of using different criteria to handle them may have little impact. However, in the specific problem of storing and retrieving graylevel images, there may be situations where problems may arise. For instance, let's consider two consecutive images im_1 and im_2. Let's also assume a given pixel P has graylevel 127 (01111111_2) in im_1. However, due to noise, the same pixel has graylevel 128 (10000000_1) in im_2. The value of P is *almost* the same, in natural binary code. The arithmetic distance between P_{im_1} and P_{im_2} is just 1. Nonetheless, the Hamming distance between the value of P in each image is the maximum it can be for 8-bit pixels: $hd(01111111, 10000000) = 8$ bits. That *small* amount of noise would almost certainly cause a large number of prediction errors if the images were not large enough. However, under normal circumstances, the large dimensionality of the vectors used may contribute to minimise the tolerance of the system to the problem.

As described in Section 6.2, Vanhala et al [101] use an approach that still relies on the use of the

Table 7.2: Distance between binary numbers.

Binary numbers		Distance method		
A	B	Arithmetic	Hamming	Vanhala
00000000	11111111	255	8	1
00000000	01111111	127	7	0
01111111	10000000	1	8	1

natural binary code and is more robust to noise. By using only the most significant "one" of each byte, the number of errors is greatly reduced. However, that approach works as a very rough filter, which maps the interval [0, 255] onto a smaller binary codomain where only two graylevels can be represented: $D \rightarrow D', D = [00000000, 11111111], D' = \{0xxxxxxx, 1xxxxxxx\}$. While effective reducing noise, which Vanhala reports to be the primary goal, that mapping is not the most effective solution to the distance problem being discussed. It also has, however, the merit of reducing the length of the address vectors to one eighth of their original size.

Table 7.2 shows examples of distances between various binary numbers. Those distances were computed using the arithmetic distance, the Hamming distance and the Hamming distance after Vanhala's method. For example, the distance from a black pixel (00000000_2) to a white pixel (11111111_2 in binary code, which should be converted to 10000000 considering only the most significant bit), using Vanhala's code, is just 1. The arithmetic distance and the Hamming distance are both the maximum they can be, 255 and 8, respectively. The distance between the same black pixel (00000000_2) and a mid-range gray pixel (01111111_2) is also very different from method to method. It is 127 using the arithmetic mode, 7 bits using the Hamming mode, and zero using Vanhala's method. The distance between two mid-range gray pixels, 01111111_2 and 10000000_2, is just 1 using the arithmetic mode, while it is 8 (the maximum it can be) using the Hamming mode, and again 1 using Vanhala's method. It is clear that to effectively store graylevel images into the SDM, and still use the Hamming distance or another bitwise method, data has to be encoded using a different method. Possibly, data has to be encoded using a method where the number of ones is proportional to the graylevel value of the pixel, since the Hamming distance operates based on the number of ones existing in the binary vectors.

7.1.2 Gray code

One possible candidate to be the method of encoding the information to input to the SDM is the Gray code [33]. The Gray code was proposed in 1947, as a way to reduce the number of errors in electromechanical switches. It is also known as "reflected binary code", because it can be derived from the natural binary code by a sort of reflection process. In Gray code, only 1 bit differs between any two adjacent numbers. That property ensures that transitions such as the one from 127_{10} to 128_{10} occur

Table 7.3: 3-bit natural binary and Gray code.

Decimal	Natural BC	Gray code
0	000	000
1	001	001
2	010	011
3	011	010
4	100	110
5	101	111
6	110	101
7	111	100

smoothly. While in natural binary code all the 8 bits change, in Gray code only one bit differs. The Gray code has been widely used for information transmission, because in data transmitted using the Gray Code a single bit error is not as catastrophic as it can be using the natural binary code. It is still used to facilitate error correction in digital transmissions, such as digital terrestrial television. Table 7.3 compares the Gray code and the natural binary code for 3-bit numbers.

The Gray code, however, exhibits two undesirable characteristics for the problem being studied: i) the Hamming distances between numbers are not proportional to the arithmetic distances; and ii) it is circular, so that the last number only differs one single bit from the first number. In the case of image processing, in the worst case that means that a black image is *almost* a white image, according to the Hamming distance applied to images encoded using the Gray code.

Table 7.4 shows the Hamming distances between all the 3-bit numbers represented using a Gray code. As happens in the natural binary code, there are also many *undesirable transitions* in the table. There are some symmetries in the matrix, but there is no clear advantage in that fact. It should be noted, however, that sudden transitions such as the transition from 127_{10} to 128_{10}, never occur. In the same row of the matrix, two adjacent numbers are always consecutive, although they can increase and decrease from the left to the right. In summary, while the Gray code solves the very specific problem of transition of all bits between consecutive numbers, it is not a general solution to the general problem of encoding the information to compute the distances between memory vectors.

It is necessary to note, however, that under normal circumstances, most of the errors can be expected to occur between very close numbers. In that aspect the Gray code can be expected to produce better results than the natural binary code. However, due to its symmetry, the probability of confusion between very dark and very clear images poses an additional problem. Therefore, the Gray code is not a general solution to the encoding problem being discussed.

Table 7.4: Hamming distances for 3-bit Gray Code.

	000	001	011	010	110	111	101	100
000	0	1	2	1	2	3	2	1
001		0	1	2	3	2	2	2
011			0	1	2	1	2	3
010				0	1	2	3	2
110					0	1	2	1
111						0	1	2
101							0	1
100								0

7.1.3 Sorting the bytes

As explained in Sections 7.1.1 and 7.1.2, neither the natural binary code nor the Gray code exhibit Hamming distances between numbers proportional to the arithmetic distances. Actually, there probably is no such code. But one tentative solution to find such a code might be sorting the bytes of the natural binary code in a more convenient way, so that the Hamming distances become proportional to the arithmetic distances.

Generating different sortings

Using n bits, it is possible to generate 2^n different numbers. However, the codomain of Hamming distances contains only $n - 1$ different distances. The codomain of arithmetic distances is much larger than the domain of Hamming distances, as explained in Section 6.5. The consequence of that fact is that it is not possible to find a perfect code, because there are not enough elements in the codomain of the function. The goal is, therefore, to find a code in which the top-right rows of the matrix of Hamming distances increase monotonically from left to right. At least, it should not exhibit so many *undesirable transitions*, the situations in which the Hamming distance decreases while the arithmetic distance increases. Such a code should be found by trying different sortings of the numbers and computing the matrix of Hamming distances for each sorting. Those sortings can be generated by the different permutations of the numbers. For 3-bit numbers, there are 8 different numbers and $8! = 40320$ permutations. 40320 permutations can be easily computed using a modern personal computer, in a reasonable amount of time. After an exhaustive search, different sortings are found, but none of them ideal. Table 7.5 shows one of the best of those sortings, better than the natural binary code shown in Table 7.1. The new code found contains only 7 undesirable transitions, while the natural binary code exhibits 9 undesirable transitions. Therefore, the sorted code should perform better if used in a SDM with the Hamming distance.

Table 7.5: Hamming distances for specially sorted 3-bit numbers.

	000	001	010	100	101	111	011	110
000	0	1	1	1	2	3	2	2
001		0	2	2	1	2	1	3
010			0	2	3	2	1	1
100				0	1	2	3	1
101					0	1	2	2
111						0	1	1
011							0	2
110								0

It should also be mentioned that there are several sortings with similar performance. In the permutations of 3-bit numbers, there are 2783 other sortings that also have 7 undesirable transitions. The code shown in Table 7.5 is the first that was found by the permuting algorithm.

Computing permutations

While permutations of eight numbers are quick to compute, permutations of 256 numbers generate 256! sortings, which is computationally very expensive. However, as happens in the case of 3-bit numbers, there are numerous sortings that should exhibit similar performance. In the case of 8 numbers, the sorting shown in Table 7.5 was found after 35 iterations. In the case of 16 numbers (4 bits), a sorting with just 36 undesirable transitions (the minimum possible) is found after only 12 iterations of the permuting algorithm. Therefore, it seems reasonable to assume that after just a few iterations it is possible to find one sorting with minimum undesirable transitions, or very close in performance to it. The assumption also makes some sense considering that the number of ones in one binary word is strongly correlated with its numerical value. Big numbers tend to have more bits "one", and small numbers tend to have less bits "one". Therefore, the natural binary code is already partially sorted, although not optimised.

The matrix of 8-bit numbers in natural binary code shows 18244 undesirable transitions. Picking 5000 random sortings, the average number of undesirable transitions was found to be 25295.25. And even after several days of computation, the software was not able to randomly find a better sorting. As described above, that makes some sense, considering that the NBC is partially sorted, and also considering the dimension of the search space. However, that also shows that sortings better than the natural binary code must be rare. Additionally, they are most likely to be found by exchanging the biggest numbers of the set, since those big numbers contain, on average, more ones than the smallest numbers, which contain more zeros.

After the random search and initial results, a different approach was tried. It consisted in testing

different permutations, starting from the most significant numbers. After 57,253,888 iterations the software found one sorting with just 18050 undesirable transitions[1]. It is an improvement of less than 2%, compared to the 18244 transitions of the natural binary code. But it is an improvement nonetheless. It is not clear if there is a better sorting, but up to 681,400,000 permutations the software was not able to found a better result. The numbers 244 to 255 have been tested for their *best* position in the vector of sorted bytes, but the other numbers have not been permuted[2].

Which sorting to choose

As written before, there are several sortings which exhibit the same number of undesirable transitions. Therefore, another question arises: shall the performance of all those *equivalent* sortings be similar? The answer might depend on the particular domain. If the data is uniformely distributed, then all the sortings must exhibit a similar performance. But if the occurrence of data is more probable at a particular subrange of the sorted vector, then the sorting with less undesirable transitions in that subrange is expected to exhibit the best performance. In the present work, the goal is to find the best code to compute the similarity between two PGM images. If those images are equalised, then the distribution of all the brightness values is such that all the values are approximately equally probable. As a consequence, it means that it is irrelevant which sorting is chosen. All the sortings with the same number of undesirable transitions should produce similar results. Hence, the results shown further were produced with the first of the best sortings that was found, which contains 18050 undesirable transitions. It is called "optimised code" in further sections.

7.1.4 Reducing graylevels

As written previously, using 256 graylevels in a one-byte long word, it is not possible to find a suitable binary code with few undesirable transitions. Therefore, it is not possible to take advantage of the representativity of the natural binary code and still take the maximum out of the properties of the SDM. The only way to avoid undesirable transitions at all is to reduce the number of different graylevels to the number of bits + 1. Using such a code, with 8 bits it is only possible to represent 9 different graylevels, as shown in Table 7.6. Using 16 bits, it is possible to represent 17 graylevels and so on. That is the only way to work with a Hamming distance that is proportional to the arithmetic distance. The decimal value of each byte is equal to the sum of the bits. In mathematical terms, number y is represented within an interval $[a, b] = \{y \in \mathbb{Z} | a \leq y \leq b\}$ with the number of 1-bits according to Equation 7.1. Such

[1] The sorting found is: 0, 1, ... 243, 245, 249, 253, 252, 255, 254, 246, 250, 248, 244, 247, 251.

[2] The use of evolutive approaches, namely genetic algorithms, has been considered. But weighing the amounts of effort and time required against the expected gain, it was not considered worth a try for the present work.

Table 7.6: Sum code to represent 9 graylevels.

Decimal	Sum-code
0	00000000
1	00000001
2	00000011
3	00000111
4	00001111
5	00011111
6	00111111
7	01111111
8	11111111

representation is popularly known as the *sum code*, a derivate of *1-of-n* code:

$$sc(y) = y - a, \quad \text{for} \quad y \in \mathbb{Z} | a \leq y \leq b \tag{7.1}$$

The sum code may lead to a significant extension of the dimensionality of the input vector. Since a SDM is particularly designed to cope with high-dimensional input vectors, it should not be a problem. It should be noted that in an unsorted sum code there may be many different possibilities to represent the same number, by shifting the bits. For example, the number 3 in a 4-bit code may have four different representations: `0111;1011;1101;1110`. However, in the present work it is guaranteed that only one form of the sum code is used, and the bits are sorted in each binary word.

The disadvantage of using a sum code, however, is obvious: either the quality of the image is greatly impaired, or the dimension of the stored vectors has to be increased to store additional bits. In the present implementation, however, there is a compromise solution option: the images are still stored as 256 graylevel images, so that quality is not lost for the other operation modes. But they are converted either to 9 or to 17 graylevels to compute similarity between two address locations in runtime. Using this approach, the quality of the images is maintained. That guarantees the comfort of the users that may monitor the robot's operation, and also facilitates the process of testing different modalities with the same sequences of images. The only drawback of reducing the graylevels in real time is that it requires additional processing of each image.

The sum code approach is similar in many aspects to Vanhala's [101] idea of considering only the most significant bit, as described in Section 7.1. However, while Vanhala was more concerned about reducing noise, the goal here is to find a method of computing the similarity between memory items using the Hamming distance, with results similar or better than those obtained using an arithmetic distance. Also, while Vanhala simply truncated the bytes to get only 1 bit, here the bytes are mapped using an indexed table into a smaller domain where the number of graylevels equals the number of bits + 1. The procedure is very simple, as described in Algorithm 5. Notice that after step 2 of the algorithm, the

distance between two vectors computed using the Hamming is proportional to the distance computed using the Arithmetic distance.

Algorithm 5: Compute similarity using a sum code

Data: *number_of_graylevels*

begin

1) Get images im_1 and im_2, encoded using NBC;

2) Map each pixel value of im_1 and im_2 from the domain [0, 255] into the codomain [0, *number_of_graylevels*];

3) Compute similarity between im_1 and im_2 using the Hamming distance.

end

7.2 Experiments and results

Different tests were performed in order to assess the behaviour of the system using each of the approaches described before. The results presented in this section were obtained using a navigation sequence comprising 55 small (80×64) images. The images were always equalised. The path was first taught to the robot and saved to the computer disk. Later, it was reloaded from the disk in different arithmetic and bitwise modes. That way, different parameters of the memory could be tested separately. Each of the presented results is the average of 30 operations, except for the Momentary Localisation Errors. The number of MLEs, for each operation mode, is the average of 5 autonomous runs, because to measure each single result it was necessary to make the robot follow the whole path, which is a very time consuming task. The tests were performed first without enforcing distribution of the data through different hard locations, and later repeated enforcing distribution of each datum through a minimum set of hard locations.

7.2.1 Experiments without data distribution

First, the system was tested ensuring there was only one copy of each image in the memory. It is guaranteed that there is no distribution of the data. The SDM contains one and just one exact instance of each image. That is enforced by setting the activation radius $\beta = 0$ during the learning stage and loading of the sequences from the computer disk. Table 7.7 summarises the results obtained. The columns of the table are as follows:

Operation mode — The operation mode determines how the distance (error in similarity) between memory items is computed. "Arithmetic" stands for "arithmetic mode", as detailed in Section

6.1.3. "NBC" stands for "natural binary code" implementation, as detailed in Section 6.1.2. "OPC" stands for the results obtained using an optimised binary code, as explained in Section 7.1.3. The remainder methods refer to the sum code, where the numbers 2 to 33 stand for the number of graylevels used. For example, "B2" means the use of two graylevels and a 1-bit sum code, as explained in Section 7.1.4.

MLE — The average number of momentary localisation errors, computed for 5 runs, as defined in Section 6.5.3.

Distance to 1^{st} — The distance from the actual input image to the closest hard location in the memory. I.e., the smallest error in similarity that was measured, after testing the input against all the hard locations.

Distance to 2^{nd} — The second smaller similarity error measured. It is a measure of the distance from the input image to the second best matching image.

Inc_2 — The ratio, in percentage, between the previous two columns. Inc_2 shows how successful the memory was in distinguishing the best prediction from the second prediction.

Distance to average — The average error in similarity. It is obtained summing all the similarity errors and dividing by the number of hard locations in the memory.

Inc_{AV} — Inc_{AV} is a ratio in percentage, between the average distance and the distance to the closest hard location: $Distance\_to\_average/Distance\_to\_1^{st}$. It is a measure of how successful the system was in separating the best prediction from the pool of information in the memory.

Time — The prediction time, in milliseconds. It refers to the memory operation only. The clock was started once the image was input to the memory, and stopped once there was a prediction output from the memory. All the remainder tasks, such as image transmission and processing, were not accounted for, as they were presumably the same for all the methods. The value shown is the average time of all the memory predictions measured during the sequence.

The more important variables are plotted in Figures 7.1 to 7.3. Figure 7.1 shows the increase in image distances. Figure 7.2 compares the number of momentary localisation errors . Figure 7.3 shows the evolution of the processing time.

Distance between images

Figure 7.1 plots columns 5 and 7 of Table 7.7. The lighter bars are the ratios between the distance from the best prediction to the second best (Inc_2). The darker bars are the average distance from the best

Table 7.7: Test results obtained without distribution.

Operation mode	MLE	Distance to 1^{st}	Distance to 2^{nd}	Inc_2 (%)	Distance to average	Inc_{AV} (%)	Time (ms)
Arithmetic	29.0	35240	56511	60.36	172236.53	388.75	7.23
NBC	31.4	7575	8195	8.18	9738.01	28.56	6.90
OPC	31.2	7583	8187	7.97	9736.95	28.41	7.11
B02	29.6	161	265	65.12	1038.70	546.22	28.20
B03	31.4	328	521	58.83	1452.23	342.53	28.29
B04	27.6	469	747	59.37	2120.97	352.30	28.54
B05	28.6	561	970	72.97	2814.22	401.91	28.18
B06	27.4	735	1151	56.63	3429.04	366.54	28.16
B07	29.6	869	1381	58.98	4104.85	372.55	28.33
B08	27.8	985	1520	54.26	4765.66	383.59	28.19
B09	29.2	1115	1772	58.97	5438.20	387.89	28.16
B10	27.4	1235	1976	60.03	6101.71	394.17	28.62
B11	27.6	1358	2202	62.12	6749.47	396.95	28.20
B12	29.6	1536	2443	58.98	7428.49	383.48	28.20
B13	26.6	1645	2644	60.73	8092.81	391.97	28.54
B14	26.4	1797	2914	62.15	8763.90	387.71	28.19
B15	29.0	1980	3155	59.34	9460.60	377.75	28.32
B16	29.2	2090	3331	59.40	10096.41	383.20	28.20
B17	26.4	2234	3530	58.01	10802.41	383.56	28.20
B18	24.6	2357	3753	59.19	11439.53	385.29	28.57
B19	25.0	2498	4037	61.57	12149.61	386.30	28.12
B20	24.8	2631	4216	60.22	12812.02	386.95	28.22
B21	26.2	2768	4457	61.00	13487.65	387.24	28.55
B22	26.8	2934	4648	58.41	14166.28	382.84	28.15
B23	27.8	3029	4881	61.16	14798.22	388.57	28.16
B24	27.2	3192	5102	59.83	15493.66	385.37	28.60
B25	28.6	3323	5316	59.97	16150.32	386.00	28.21
B26	27.4	3469	5563	60.37	16852.26	385.85	28.32
B27	26.2	3601	5755	59.80	17495.39	385.79	28.18
B28	28.8	3751	5991	59.71	18214.14	385.57	28.22
B29	26.8	3866	6207	60.53	18843.54	387.35	28.32
B30	26.2	4006	6446	60.92	19546.59	387.99	28.16
B31	27.6	4148	6652	60.35	20188.30	386.70	28.15
B32	26.4	4315	6891	59.72	20873.04	383.77	28.14
B33	26.4	4427	7099	60.35	21553.67	386.85	28.50

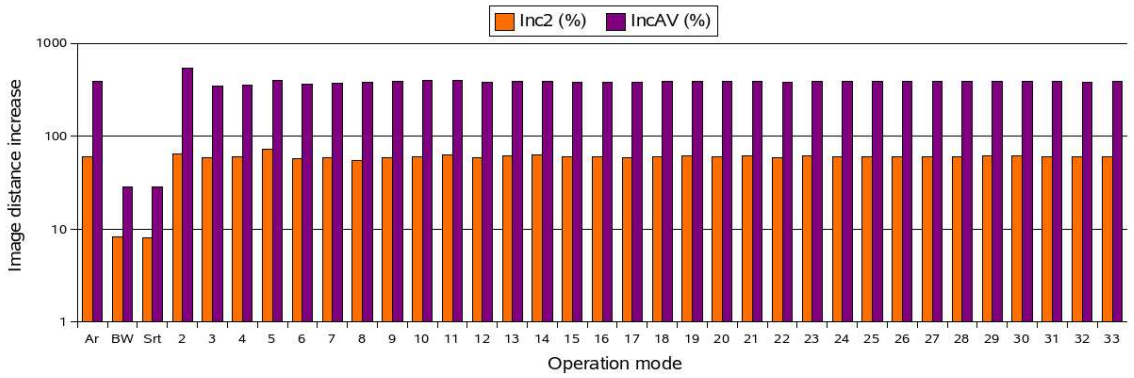


Figure 7.1: Image distance increases, to the second best image and to the average, obtained without data distribution in the SDM.

prediction to the pool of images in the SDM (Inc_{AV}). The scale used in the graphic is logarithmic, so that it is possible to accommodate in the same chart very disparate values for the different metrics. The range of values varies from 7.97 (Inc_2, in the bitwise mode using the optimised code) to 546.22 (Inc_{AV}, using a sum code with 2 graylevels).

As explained in Section 3.2.3, the SDM works by measuring the similarity between the input pattern and all the vectors it stores. Successful operation requires that the SDM succeeds in finding the right pattern and "separating" it from the other vectors. Failure in accomplishing that task results in storage of unnecessary data or output of corrupt data and incorrect predictions. Therefore, the best metric is the one which can provide the largest distance between the correct vector and all the other vectors.

From the experimental results, it is possible to see that the bitwise modes using both the natural binary code and the optimised code provide the smallest increase of the average distance, relatively to the closest image, which is according to the expectations. The sum code shows increases very similar to the arithmetic mode, thus proving to be a bitwise mode of characteristics similar to the arithmetic mode. There are small variations in the graphic as the number of bits increases, but there is no clear trend that can point out a better performance with more or less bits. In Table 7.7 it is also possible to see that the distances increase as more bits are used. But that is just a normal consequence of using more bits: there are more elements on the codomain of the distance functions, and the range of representation is enlarged. The results do not show a clear practical advantage of using more bits for images of this size.

Momentary localisation errors

As Figure 7.2 and Table 7.7 show, there are small variations in the number of momentary localisation errors. The amplitude of the variations is small, and there is no clear trend that can be interpreted as

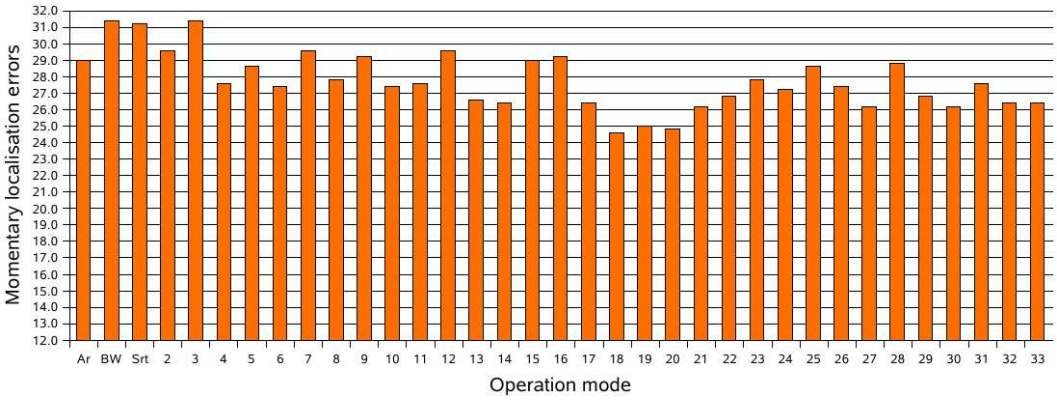


Figure 7.2: Number of MLEs obtained without distribution of the data.

a definitive conclusion. However, a closer look gives some interesting clues that can confirm the theory.

The number of MLEs is lower in the arithmetic mode, compared to the bitwise modes using the natural binary code and the optimised code. That is according to the logical expectations: a larger probability of image confusion in the bitwise modes leads to more memory prediction errors and, therefore, to more momentary localisation errors.

The number of MLEs is smaller using the optimised code than it is if the natural binary code is used. However, the difference is very small. It is hardly solid evidence of the superiority of the optimised code, although it is a good trend indicator.

As for the sum codes, it can be seen that there are large fluctuations of the number of MLEs as the number of bits used varies. There is no clear explanation for those fluctuations. They are probably a result of the random characteristics of the problem. However, it can be seen that in general the trend is that less errors occur as more bits are used. It also seems that the sum code in general causes less MLEs than the other methods. There is apparently no explanation for that behaviour. However, it should be noted that using the sum code the robot moves slower, because converting the image pixels to the sum code requires some processing power and poses a non-neglectable delay in real time. Since the robot moves slower, there may be less slippage and drift errors, and that may be an explanation for the better performance in terms of momentary localisation errors. The sum code using 3 graylevels also shows a high number of average localisation errors (31.4), although that is most likely a spurious result obtained in the experiment.

Processing time

The processing times exhibit a large variation, for the tests were run on a computer using Linux, as described in Section 5.1.2. Linux is a *best effort* operating system, not a real time operating system.

Also, the software was not designed with the purpose of guaranteeing real time performance—it was just optimised, as much as possible, to be versatile and run fast. However, the times shown in the table are the average of 30 runs of 55 images. Therefore, they are quite smooth, as the graphic shows. For better precision in measuring processing times, and real time operation, a real time operating system would be recommended.

Comparing the processing times shown in Section 6.4.1 with those shown in Table 7.7, it can be seen that the processing times in the latter case are much smaller than they are in the former. For the arithmetic mode, while in Section 6.4.1 the processing time was as large as 50 ms, in the latter case it is just about 7 ms . That is mostly due to two different reasons:

1. A much smaller sequence is used here: 55 images, against 100 images in Section 6.4.1. That cuts the processing time down to $\frac{1}{2}$;

2. A smaller number of copies of each image is used here: one copy against 3 copies. That cuts the processing time down by approximately $\frac{1}{3}$.

The two reasons pointed out explain why the average time in these experiments is about $\frac{1}{6}$ of the time presented in Section 6.4.1. Both times are correct, it just happens that the length of the sequence and the distribution are much more favourable in the new experiment.

Table 7.7 and the chart shown in Fig. 7.3 show that the arithmetic mode is a little slower than the bitwise modes using either the natural binary code or the optimised code. The optimised code, where some numbers were sorted, takes a little longer than the natural binary code, because some numbers are swapped in real time to map the natural binary code into the optimised code, as explained in Section 7.1.3. The sum codes take a lot longer, because the bit conversion is also implemented in software and performed in real time for each image byte. That conversion increases the processing time by a factor of 4. In hardware or software specifically designed for the intended purpose, the conversion could be done automatically as the image was grabbed, and the delay, if any, would be minimum.

7.2.2 Experiments with data distribution

In order to get an insight into the impact of using distribution of the data in the SDM, the same previous tests were performed again, loading the memory with multiple copies of each image. Distribution of the data was enforced, setting the minimum number of copies to 5. One of the copies was exact. The other four copies had some noise added before being stored, so that each instance was stored into a different hard location, still within the access circle, as explained in Section 6.1.1.

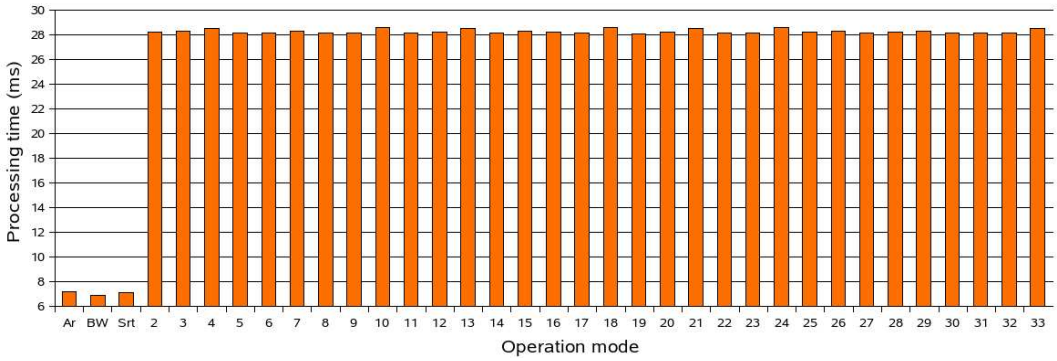


Figure 7.3: Processing times, obtained without data distribution.

Table 7.8 shows the results obtained. The columns of the table have the same meaning as those of Table 7.7. The more important results are also illustrated in charts 7.4, 7.5 and 7.6.

Distance between images

Figure 7.4 plots the distances between images, computed with data distribution. The Y axis scale is again logarithmic, because the distances to the second best prediction and to the average are still of different orders of magnitude. For the average distance, the pattern found in the chart is the same that can be found in Figure 7.1, with no data distribution. The bitwise mode using the natural binary code or the optimised code is not able to differentiate the best prediction as much as the other methods. However, there is a dramatic difference in the distance to the second closest image: it is drastically smaller than the one observed without distribution, and there is no clear pattern in this case. The obvious consequence is that the probability of the second closest image being mistaken by the closest image is much higher with distribution. However, that is an expected result. The memory contains 5 copies of each image: one copy is exact, the others have some noise added, so that they are stored into different hard locations within the activation radius. Therefore, the distance to all of those 5 copies has to be very small, less than the activation radius. Nonetheless, any of those 5 copies is actually an acceptable prediction of the right image. Hence, the difference noticed does not show an undesirable result obtained with distribution. It is just a characteristic of the SDM.

Momentary localisation errors

As for momentary localisation errors, again there is no clear pattern in the data. Nonetheless, comparing the results obtained with distribution to the results obtained without data distribution, some important facts can be pointed out.

First, it important to note that there are, on average, much less momentary localisation errors.

Table 7.8: Experimental results obtained with distribution of the data.

Operation mode	MLE	Distance to 1^{st}	Distance to 2^{nd}	Inc_2 (%)	Distance to average	Inc_{AV} (%)	Time (ms)
Arithmetic	23.2	35528	36467	2.64	172259.43	384.85	33.38
NBC	26.0	7547	7831	3.77	9740.06	29.06	31.63
OPC	24.2	7544	7873	4.37	9742.55	29.15	32.89
B02	27.4	165	169	2.43	1038.38	530.34	137.86
B03	25.0	330	336	1.94	1453.93	340.45	138.90
B04	24.0	464	469	1.11	2123.09	357.89	138.69
B05	27.2	567	582	2.71	2814.12	396.52	107.47
B06	24.2	731	742	1.41	3426.69	368.53	138.92
B07	24.8	867	888	2.37	4106.51	373.61	137.94
B08	25.6	988	1053	6.58	4763.40	382.29	100.03
B09	23.8	1101	1136	3.20	5430.48	393.41	101.56
B10	26.2	1230	1265	2.86	6097.06	395.68	105.37
B11	24.2	1354	1397	3.18	6754.74	399.04	92.09
B12	25.4	1524	1547	1.50	7434.76	387.95	108.53
B13	26.4	1627	1668	2.51	8091.90	397.34	110.93
B14	24.2	1779	1836	3.19	8760.66	392.44	116.25
B15	24.4	1913	1968	2.90	9446.52	393.83	117.21
B16	25.8	2069	2118	2.38	10103.21	388.39	138.24
B17	25.2	2212	2263	2.28	10802.24	388.26	139.22
B18	24.8	2330	2389	2.53	11430.94	390.68	138.90
B19	25.4	2458	2512	2.20	12145.08	394.02	92.62
B20	24.6	2588	2628	1.57	12817.88	395.36	112.67
B21	25.2	2728	2796	2.49	13490.72	394.54	95.60
B22	20.8	2874	2944	2.44	14173.24	393.18	112.50
B23	24.2	2993	3070	2.60	14807.01	394.79	129.57
B24	25.2	3151	3223	2.28	15486.35	391.42	111.59
B25	24.4	3263	3352	2.73	16159.87	395.28	94.35
B26	25.0	3397	3486	2.61	16840.29	395.75	121.92
B27	25.4	3550	3633	2.32	17493.15	392.75	92.18
B28	26.4	3678	3779	2.77	18201.37	394.93	92.00
B29	22.0	3808	3921	2.98	18829.25	394.49	131.98
B30	24.6	3937	4025	2.23	19533.66	396.13	138.20
B31	25.4	4058	4174	2.86	20185.63	397.46	138.72
B32	25.2	4237	4336	2.32	20874.98	392.64	124.05
B33	26.2	4346	4472	2.89	21557.04	395.98	109.33

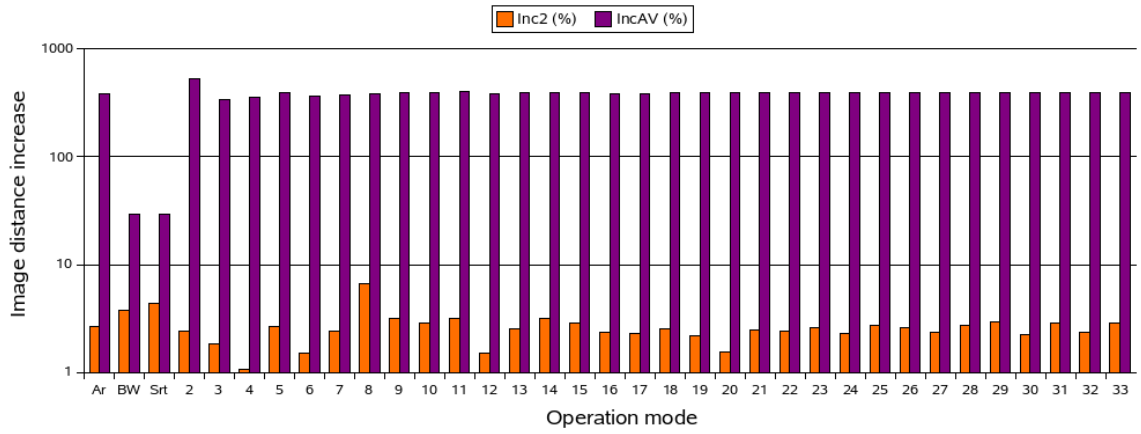


Figure 7.4: Image distance increases, to the second best image and to the average, obtained with data distribution in the SDM.

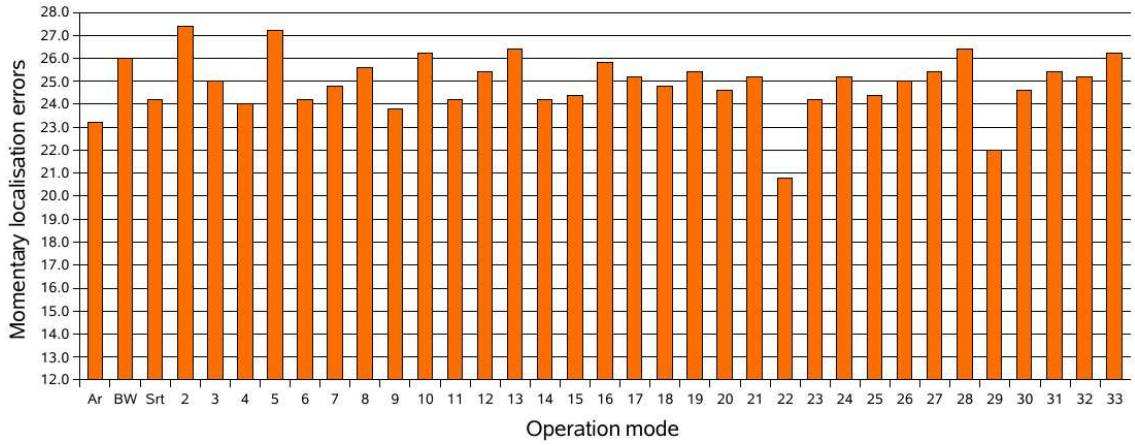


Figure 7.5: MLEs obtained with 5 copies of each image.

The average number of errors without distribution is 27.8. Distribution brings the average down to 24.9. That corresponds to a 9.0% improvement. Also, the number of errors with distribution varies between 20.8 and 27.4. Without distribution, the minimum was 24.6 and the maximum was 31.4. The explanation resides in the fact that distribution enforces 5 similar but different copies of each image in the SDM. Each of those copies is a correct prediction of the original image. Thus, the probability of retrieving at least one copy of the right image is higher than it would be if there was just one copy. The system becomes more robust and immune to noise, at the cost of using both more memory and processing power. The SDM does it *naturally*, as part of its architecture—distribution of the data strengthens the system, making it more tolerant to noise in the images and so reducing the number of prediction errors that were counted during the experiments.

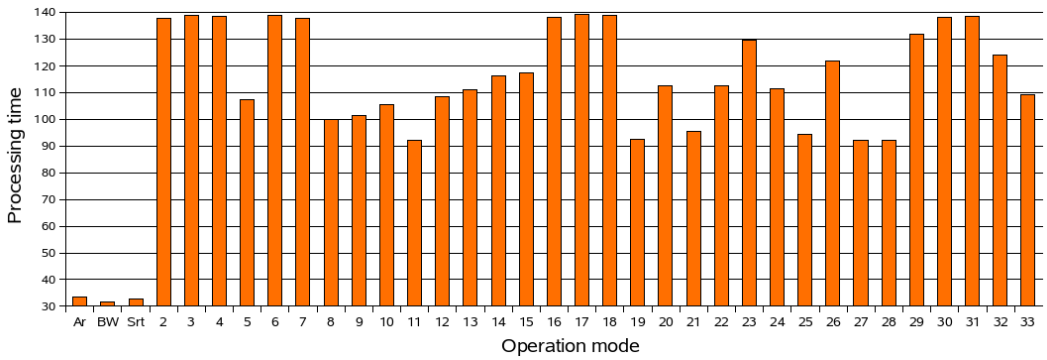


Figure 7.6: Processing times, obtained with distribution of the data.

Processing time

As could be expected, the processing times measured with distribution of the data to at least 5 hard locations are about 5 times larger than those measured in the previous experiment. The processing time is proportional to the memory size, because the memory is simulated in software. The software is being run in a computer with a single processing core, so there is no parallel processing. As the memory grows, there are more hard locations to be searched in order to find out which ones are within the access circle. The processing time is linearly proportional to the number of hard locations that have to be searched. Therefore, a 5-fold increase in the number of hard locations present causes a 5-fold increase in the processing time required. However, as stated previously, the architecture of the SDM is suitable to implement both in hardware and in a distributed architecture. In those cases, processing time would not have to be directly proportional to the memory size or the number of copies stored. Distribution of the data could have just a small, or even null, impact on the search time.

An additional point to be noted is that there are now small variations in the processing times which could not be spotted when storing just one copy of each datum. Those are most probably manifestations of the lack of accuracy of the system in terms of measuring time intervals. That lack of accuracy may be more visible as the measured time interval increases and/or more intervals are summed. There is no other obvious reason to explain the differences measured.

7.3 Summary

In the original SDM model Kanerva proposes that the Hamming distance be used to compute the similarity between two memory items. However, that method may exhibit a poor performance if the data is not random, as is the case of graylevel images in which the pixel values are represented by a binary number using the natural binary code. That happens because in NBC the value of a bit depends on its position in the byte, and the Hamming distance does not account for that correlation—the Hamming

distance is computed as if all the bits had the same weight.

This chapter described the results of various tests performed to assess the performance of the SDM with different methods of encoding the information and computing the distance between two memory items. The NBC with the Hamming distance shows the worst performance, although it is still able to successfully navigate the robot. By sorting some bytes of the code the performance is slightly improved, but not in a significant way. If the bits are grouped as bytes and an arithmetic metric is used, the memory shows an excellent performance. If the number of graylevels is reduced and a sum code is used, the performance is similar to that of the arithmetic mode and the Hamming distance can still be used. Different numbers of graylevels were tested, but for the image size used the performance was similar. Compared to the arithmetic mode, the sum code only has the disadvantage of being more time-consuming, due to being simulated in runtime. Therefore, in future experiments the arithmetic mode is preferred by default, unless stated otherwise. An additional conclusion is that distribution makes the system more robust, by reducing the number of prediction errors, at the cost of requiring additional processing time.

Chapter 8

Tests and improvements to the system

An ambitious horse will never return to its old stable.
(Chinese proverb)

Contents

In previous chapters the SDM model was described in detail, as well as the properties it exhibits. It was also described the architecture of the system developed to navigate a robot using sequences of views stored into the SDM. This chapter describes further experiments that were performed, under different conditions, in order to get a better insight into the behaviour of the model and confirm or refute the proposal. The experiments include the tests of robustness to noise, reinforcement of an idea, suitability to work with different sequences and memory saturation. Those are the characteristics of the SDM that are more relevant for navigation based on view sequences.

The results presented in this chapter for the tests using the sum code are only shown for nine graylevels (i.e., 8 bits per pixel). Additionally, when using multiple copies, it was decided to guarantee the minimum of three copies of each image, for the sake of saving processing time during the experiments and still enforce data distribution.

8.1 Kidnapped robot problem

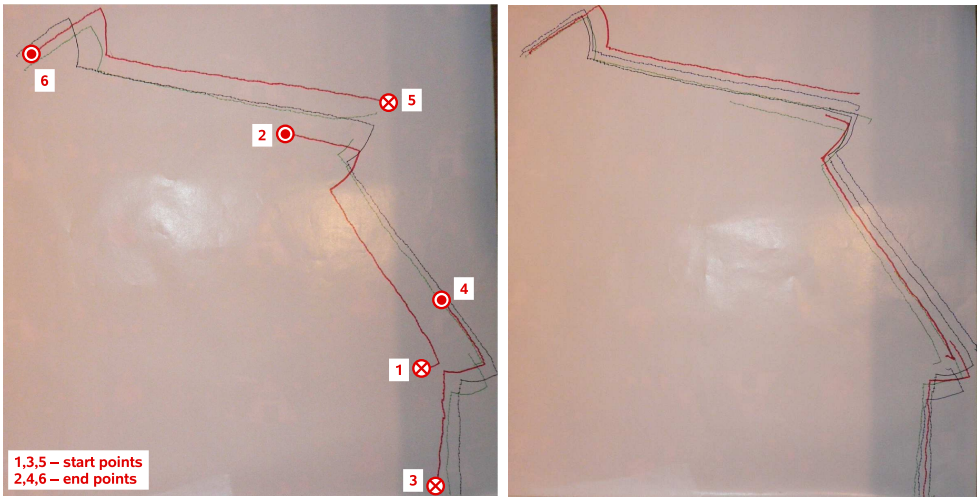
The "kidnapped robot" problem is a common challenge that many robots have to face. The problem happens when a robot is navigating in some environment, but then suddenly kidnapped to another place, either known or unknown. In that situation, the robot is expected to rapidly identify the new environment and adjust its behaviour according to it. Otherwise, it will be lost or wrongly trying to navigate based on out of date information. Obviously, it is acceptable that the identification of the new environment requires some time. Even humans and other animals need some time to localise themselves in situations of discontinuities, such as falls, transition between sleep and waking up and so on.

In the case of the present work, navigation is performed using Algorithm 3, described in Section 6.3. As the algorithm shows, it is based solely on present visual information. Hence, the robot must be immune to being kidnapped. It has no memory of the past to rely on. Therefore, it is irrelevant where it comes from, since the robot cannot "remember" that information. The robot localises itself based uniquely on its latest captured view. A sudden change from one point to another point, as long as it is placed in a known situation, causes the present view to change and the sequence and image numbers will be updated on the fly.

Figure 8.1 illustrates the robot's behaviour in a situation where it was kidnapped during autonomous navigation. The black line is the path drawn by the robot without being kidnapped. It starts at the point where the robot sees image 1 of the sequence, and goes up to the end point, where the robot sees image 54. The red and green lines in Figure 8.1(a) were drawn in three segments, because the robot was kidnapped twice during the autonomous runs. The segments were drawn as follows:

Segment 1–2 — The robot was manually placed at a random point in the middle of the path: point (1) in the image. At point (1), the robot captured an image close to image 16 of the sequence. As the line shows, Lizard rapidly converged to the correct path. It adjusted to the correct heading and proceeded following a trajectory close to the desired path. At point (2) of the path, (close to the point identified with image 23 of the sequence), the robot was kidnapped.

Segment 3–4 — While being transported, the robot captured several different views, which caused



(a) Arithmetic mode.

(b) Bitwise mode using NBC (green), Optimised code (blue) and Sum-code (red).

Figure 8.1: Paths drawn during the tests where the robot was kidnapped (with 3 copies of each image). The robot started at (1), proceeded to (2), where it was kidnapped to (3) and then followed to (4).

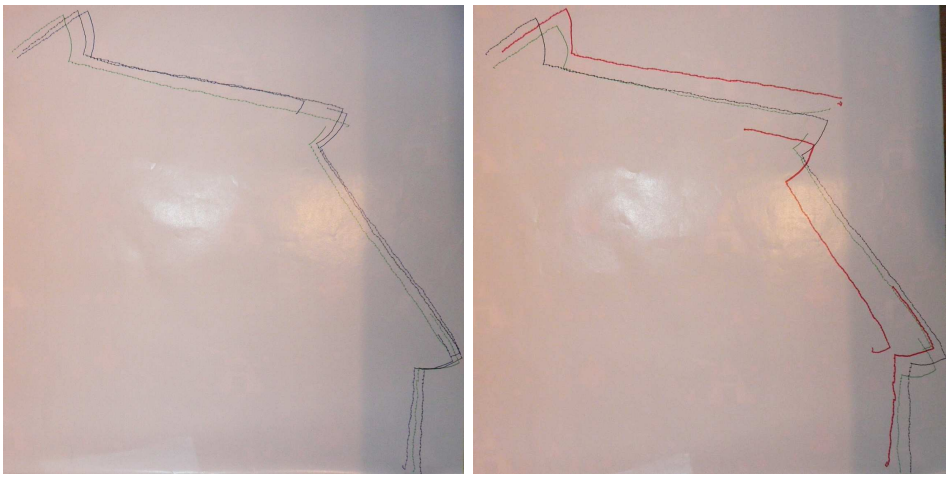
it to get temporarily "lost" in the path[1]. However, as soon as it was placed at a point close to the beginning of the path, it was able to localise itself again. From there it spotted image 1, and slowly converged to the correct path, up to image 16 again (point 4). At point (4), the robot was kidnapped again, to point (5), close to image 30.

Segment 5–6 — From point (5), the robot drew segment (5)–(6), converging towards the correct path until image 54, where it stopped (point 6).

Figure 8.1(a) shows the results of tests performed using the arithmetic mode. Figure 8.1(b) shows the results obtained using other operation modes. As the figures show, there is no noticeable difference on the performance of the system, regardless of the operation mode used.

The results shown in Figure 8.1 were obtained enforcing three copies of each image in the memory. But in order to infer if data distribution had any effect on the performance of the system, the tests were also repeated enforcing just one copy of each image. Figure 8.2 shows the results obtained in various

[1]The numbers of the images retrieved by the robot while drawing the red line are: <*Point 1*> 16 16 18 16 16 16 16 16 17 17 17 18 18 18 18 18 18 19 18 18 18 18 19 19 19 19 20 19 19 19 20 20 20 20 20 20 20 20 21 21 21 21 21 21 21 21 22 22 22 22 22 22 23 22 22 23 <*Point 2*> 26 11 11 53 9 <*Point 3*> 1 1 1 2 3 3 3 3 3 3 4 4 3 4 4 4 4 4 4 4 5 5 5 5 5 5 5 5 6 6 6 6 6 6 6 7 6 7 6 7 5 6 7 6 7 8 8 8 8 8 9 9 9 9 9 10 10 10 10 11 11 11 11 11 12 12 12 12 12 13 13 13 13 14 14 14 14 16 15 16 15 16 16 16 16 16 16 16 <*Point 4*> 19 <*Point 5*>30 30 30 30 31 31 31 31 32 32 32 33 33 33 33 33 33 33 33 33 33 33 33 33 33 33 33 34 34 34 34 34 34 34 34 34 34 35 35 35 35 35 35 35 35 36 36 36 36 36 36 36 37 37 37 37 37 37 37 37 37 37 37 37 38 38 38 38 38 39 39 39 39 39 39 39 39 39 39 40 40 40 40 40 40 40 40 40 41 41 41 41 41 41 41 42 41 42 41 42 41 42 41 42 41 42 43 43 43 43 44 44 44 44 45 45 45 45 46 46 46 46 47 47 47 48 48 48 49 49 49 49 50 50 50 50 50 51 51 51 51 53 53 53 53 53 53 53 53 53 53 53 54 53 54 53 54 54 54 54 54 54 <*Point 6*>.



(a) Arithmetic mode (green) and Sum-code (blue).

(b) Bitwise mode using NBC (green) and Optimised code (red).

Figure 8.2: Paths drawn while the robot was kidnapped (1 copy of each image).

modes with just one copy of each image. It can be noticed that there is no significant difference in the performance of the robot, regardless of the operation mode or number of copies stored for each image. The robot was always immune to being kidnapped in all the experiments. Even when the robot was left some centimetres away from the path, or in the wrong angle, it showed a high tolerance and ability to converge towards the correct path. In some path segments, it is clear that the robot immediately starts aligning itself when it starts following the path.

It should be noticed that the results shown were obtained because the robot was always kidnapped to known locations. If it was kidnapped to an unknown location, the robot would most probably be unable to converge to any known path. That problem could be overcome implementing algorithms to provide search or wander modes to explore the environment and try to find a known location or reference point to localise itself and proceed from there. However, the implemented algorithms do not provide those behaviours. At an unknown location the robot will proceed only forward until it finds either a known location or an obstacle. If it finds a known location it will follow the path associated with that location until the end of that path. If it finds an obstacle it will stop there.

8.2 Robustness to noise

The problem of noise in the images is very important, as explained in Section 6.2. It is minimised by image processing, as detailed in Section 5.2.2, and shown by the results presented in Section 6.4. Two important sources of disturbances that can be regarded as noise are illumination changes and scenario changes. In order to have a better insight into how the system responds to those disturbances, and the

(a) Illuminated by a 300 W projector. (b) Ambient light only.

Figure 8.3: Scenario under very different illumination conditions.

influence of the encoding method or data distribution on its response, further experiments were carried out and the results are reported in this section.

8.2.1 Illumination changes

The experiments with changing illumination conditions were performed in the same circumstances as described in Section 6.4.2: the robot was taught a path under the light of a 300 W projector, and had to follow the same path under the same light and with ambient light only.

Figure 8.3 shows the experimental setup under those two different illumination conditions. The different shadows can be clearly spotted in the two images. It can also be noticed that the light sources are placed in opposite directions. In Figure 8.3(a) the bottom is clear white, and there is only the shadow of the right wall, since the halogen bulb is projecting strong white light from the right to the left. In Figure 8.3(b), the bottom is yellowish, because the fluorescent light has a much lower temperature than the halogen light, besides being much dimmer. There are also shadows of several objects, from the left to the right. The right wall is not projecting any shadow inside the arena this time. As a consequence, the scenario wall that is in the shadow in one situation is the one receiving better illumination in the other situation. That is possibly the most challenging situation for the robot.

Results

Table 8.1 summarises the results obtained in the experiments. The first column of the table is a measure of the distribution of the data. It represents the minimum number of copies that is guaranteed to be stored into the memory for each image. The second column is the intensity of the light. Since it was not possible to have a more accurate measure, the two alternatives are qualitative: “strong” and “dim”. The third column is the operation mode. The fourth column is the average number of momentary localisation errors. The remaining columns represent the image distances and increments as defined in Chapter 7.

Table 8.1: Results obtained under the two different illumination conditions shown in Figure 8.3.

Copies stored	Light intensity	Operation mode	MLE	Distance to 1^{st}	Distance to 2^{nd}	Inc_2 (%)	Distance to average	Inc_{AV} (%)
3	Strong	Arithmetic	16.4	35777	37045	3.54	176412.15	393.09
		NBC	20.6	7520	8090	7.58	9809.46	30.44
		OPC	23.0	7514	8052	7.16	9813.67	30.60
		B09	14.4	35943	37222	3.56	176540.77	391.17
	Dim	Arit	62.6	109915	110694	0.71	167290.65	52.20
		NBC	∞	9165	9326	1.75	9789.42	6.81
		OPC	∞	9166	9326	1.74	9793.26	6.84
		B09	61.2	110292	111071	0.71	167205.93	51.60
1	Strong	Arithmetic	17.2	37094	65909	77.68	176451.03	375.69
		NBC	24.8	7638	8364	9.51	9807.73	28.41
		OPC	23.6	7641	8418	10.17	9809.60	28.39
		B09	14.6	36922	65822	78.27	176357.62	377.64
	Dim	Arithmetic	56.1	110503	130151	17.78	167257.82	51.36
		NBC	∞	9230	9404	1.89	9788.20	6.05
		OPC	∞	9183	9400	2.37	9784.51	6.55
		B09	60.9	110492	130143	17.79	166972.05	51.12

The number of momentary localisation errors for the natural binary code and the optimised code is shown as ∞, for the dim light tests, because using those methods the robot was usually unable to successfully complete the path under dim light. Therefore, it is reasonable to assume that those operation modes are not as suitable to guarantee a good immunity to illumination changes as the arithmetic and the sum code modes. That fact is also illustrated by the image distances. The average distance from the current image to all the images in the SDM is very small, only 6–7% larger than the distance to the closest prediction. That means the probability of confusion of images is very high. That probability being high, the robot easily makes wrong predictions and gets lost. In consequence, it shows an inability to successfully complete the job in a reasonable amount of time.

As for the arithmetic mode and the 8-bit sum code, their performance is almost the same. Their success in separating the best matching image from the pool of images is very similar, both under a strong light and under a dim light. The number of momentary localisation errors is also similar for both methods, 14–18 under strong light, and 56–63 with ambient illumination only. Under dim light, the increase in the image distances is much smaller, resulting in a much higher probability of wrongly retrieving a wrong image. As a consequence, the number of prediction errors increases drastically, causing about four times more momentary localisation errors. Nonetheless even with a large number of errors, the system has shown to be able to successfully complete the task in the tests.

As for distribution of the data in the SDM, the results using three copies of the images are in general

better than the results obtained using a single copy. With distribution of the data, the average distance from the best matching image to the pool of images is always larger, regardless of the illumination level. However, it should be noticed that the number of momentary localisation errors under dim light is smaller without distribution. That result has been confirmed several times. It may be a particular result of the conditions in which the experiments were carried out, or there may be some reason that is still unclear for the behaviour. It is possible that the noise added to the images when distributing them within the access circle might increase the number of prediction errors under difficult conditions, but the hypothesis has neither been confirmed nor dismissed so far.

8.2.2 Scenario changes

Scenario changes occur in almost every real environment all the time. For example, doors can be either open or closed, people can pass by or just come into the scene and stay for a while, objects can be changed from one position to another, rotated or shifted, as well as removed from or added to the scene, the colours of the walls or of some particular objects may be changed, etc. All those changes must be accounted for, because they can pose serious problems to an autonomous robot, eventually making it impossible for the robot to recognise a scene based on visual information only, even if it has already been there and correctly stored valid memories.

Scenario changes are just a particular kind of noise for a SDM. Since having only 20% of the bits correct is enough to retrieve the right datum from the memory, with high probability, the effect of a scenario change is just that the tolerance of the SDM to other forms of noise will decrease. If at least 20% of the bits of the image are correct, scenario changes and noise should not pose a serious threat to the performance of the system. How much interference is tolerable depends on other environmental conditions and on the parameters of the SDM. In order to get a better insight into the behaviour of the system, another set of experiments was performed.

One method to simulate those scenario changes is just to put random objects in the field of view of the robot. If those objects are black, they will not reflect light onto other objects, thus not interfering with the remainder of the scene. Their only interference is by occluding other objects behind them. Hence, the level of their interference can be measured by the number of pixels they occupy in the image. That method is attractive, for its simplicity. However, there are many practical questions to be solved. First, the problem of detecting the contour of the object in the image and counting the pixels without error is not trivial. Although simple in theory, in practice there may easily be a large error, specially if there are other black or dark objects in the image. And there is also another problem: how to effectively design and run some tests based on that principle? A simple test could be performed with the robot stopped at a point where it would see exactly image j of the sequence, for instance. Under normal

circumstances, the SDM will retrieve image j with just a small error. The black object could then be placed far from the camera and then approximated in small steps. The SDM prediction error will increase as the object is approximated, and at some point the SDM will possibly not be able to make an accurate prediction. That would give a rough idea of the tolerable size of the interference in the image.

But there is also another method, which in principle can produce more accurate results. The idea is to simulate the object in software, by erasing part of the image. That is easily done by setting some pixels to zero before sending the image to the SDM. The effect is the same as if there was a real object in the scene, but it is possible to have total control of the dimensions of the occlusion, and it is possible to maintain it as long as desired. The robot can even be moved while the interference is maintained in its exact size. It should also be noticed that erasing part of the image, or using a black object, is a *worst case* scenario: noise and the size and colour of objects that may appear in the scene are random, not black (zero) or white (255).

Results

Table 8.2 shows the results obtained when the memory was loaded with three copies of each image. Table 8.3 shows the results obtained when the memory was loaded with just one copy. The first column of each table is the number of lines that were erased before the image that represented Lizard's current view was used to predict the next step. The remainder columns are similar to that of Table 8.1.

The tables show the results obtained without interference (i.e., zero lines erased), for control, and then the results obtained with interferences of increasing severity. First, two vertical lines were erased. Then, up to fourteen lines, in steps of two. When fourteen lines were erased, the robot was lost in all the operation modes, both with and without data distribution. Therefore, no results are shown for an interference of fourteen lines. With an interference of twelve lines, only the bitwise modes using either the natural binary code or the optimised code were able to complete the job.

It can be seen that the performance of the system degrades slowly as more lines are erased, regardless of the operation mode. The average distance to all the images gradually approximates the distance to the best matching image, and the number of momentary localisation errors increases.

Under the conditions, once again, distribution improves the results a little bit. When the memory had three copies of each image, the system performed better than with just one copy. Surprisingly, the arithmetic and the sum-code collapsed before the bitwise modes using either the NBC or the optimised code. That was true both with and without distribution. Apparently, it may be a wise choice to use the bitwise mode to navigate the robot in environments where occlusions may be frequent and the arithmetic or sum code modes in environments where illumination changes are more likely to happen than

Table 8.2: Results obtained with different interferences and three copies stored.

Erased lines	Operation mode	MLE	Distance to closest	Distance to 2^{nd} closest	Inc_2 (%)	Distance to average	Inc_{AV} (%)
0	Arithmetic	14.6	54652	87745	60.55	182629.47	234.17
	NBC	20.6	8114	8463	4.30	9870.71	21.65
	OPC	23.0	8136	8490	4.36	9869.82	21.32
	B09	14.4	54498	87723	60.97	182487.57	234.85
2	Arithmetic	14.4	63587	97279	52.99	184280.02	189.81
	NBC	21.6	8167	8564	4.86	9853.60	20.65
	OPC	22.8	8182	8440	3.16	9858.96	20.50
	B09	14.4	63323	96854	52.95	184214.78	190.91
4	Arithmetic	18.6	73902	106323	43.87	187994.85	154.39
	NBC	25.2	8300	8497	2.37	9871.55	18.93
	OPC	22.8	8294	8601	3.70	9872.07	19.02
	B09	16.2	73961	106295	43.72	188121.07	154.35
6	Arithmetic	18.4	81679	114396	40.06	188423.93	130.69
	NBC	26.4	8267	8610	4.15	9837.13	19.00
	OPC	23.6	8306	8596	3.49	9836.61	18.43
	B09	17.0	81464	114499	40.55	188373.32	131.24
8	Arithmetic	17.6	91706	122951	34.07	187514.23	104.47
	NBC	25.0	8411	8969	6.63	9803.17	16.55
	OPC	24.0	8427	8988	6.65	9803.66	16.34
	B09	16.0	91312	122639	34.31	187271.50	105.09
10	Arithmetic	20.4	101537	131650	29.66	191837.50	88.93
	NBC	25.6	8514	9012	5.85	9820.46	15.34
	OPC	29.0	8532	9021	5.73	9817.04	15.06
	B09	19.8	101923	131934	29.44	192016.88	88.39
12	Arithmetic	∞	112638	143044	26.99	194584.77	72.75
	NBC	29.8	8601	8997	4.60	9818.45	14.15
	OPC	25.0	8591	9027	5.07	9820.78	14.31
	B09	∞	112300	142733	27.10	194599.60	73.29

Table 8.3: Results obtained with different interferences and one copy stored.

Erased lines	Operation mode	MLE	Distance to 1^{st}	Distance to 2^{nd}	Inc_2 (%)	Distance to average	Inc_{AV} (%)
0	Arithmetic	15.0	54507	87059	59.72	181947.07	233.81
	NBC	24.8	8180	8749	6.95	9868.98	20.64
	OPC	23.6	8213	8742	6.43	9869.50	20.16
	B09	14.6	54730	87174	59.28	182017.83	232.57
2	Arithmetic	15.8	65487	98889	51.01	185753.70	183.65
	NBC	25.6	8236	8574	4.10	9875.82	19.91
	OPC	22.8	8238	8534	3.60	9876.16	19.89
	B09	14.6	65473	98877	51.02	185759.18	183.72
4	Arithmetic	16.6	72507	106315	46.63	185401.72	155.70
	NBC	24.2	8281	8588	3.71	9839.08	18.81
	OPC	23.6	8279	8621	4.12	9841.07	18.86
	B09	16.2	72538	106384	46.66	185274.70	155.42
6	Arithmetic	16.0	81816	115302	40.93	186540.85	128.00
	NBC	27.0	8346	8895	6.59	9819.65	17.66
	OPC	23.8	8396	8918	6.21	9823.91	17.01
	B09	15.0	81837	115358	40.96	186616.75	128.04
8	Arithmetic	16.6	90590	124716	37.67	188241.75	107.80
	NBC	24.0	8451	9029	6.84	9807.78	16.05
	OPC	26.0	8453	8922	5.55	9807.15	16.02
	B09	14.0	90685	124776	37.59	188313.17	107.66
10	OPC	16.6	100140	131304	31.12	189157.15	88.89
	NBC	26.8	8552	9025	5.53	9780.23	14.36
	OPC	26.8	8542	9049	5.94	9785.44	14.56
	B09	14.2	100080	131072	30.97	189166.00	89.01
12	Arithmetic	∞	110772	143531	29.57	195239.57	76.25
	NBC	29.2	8609	9134	6.10	9817.04	14.04
	OPC	30.4	8621	9161	6.26	9815.77	13.86
	B09	∞	110581	143292	29.58	195178.18	76.50

Figure 8.4: Image with an occlusion of 12.5% of the pixels.

occlusions. The bitwise modes seem more robust to localised interferences than to global interferences, while the arithmetic and sum code modes are the opposite.

The arithmetic and the sum code supported an occlusion of 10 out of 80 lines of the image, which accounts for 12.5% of the total area of the image. The bitwise modes using either the NBC or the Optimised code supported up to 12 out of 80 lines, which accounts for 15.0% of the image.

Figure 8.4 shows an image where an area corresponding to 12.5% of the total was erased. It is a visualisation of how much an image can be damaged before the robot fails to accomplish its job.

The results here are consistent with the previous results, and add some practical results to the theory described in the previous chapters. As shown in Section 6.2, the average noise values are about 15% of the bits of an image, under normal circumstances. The interferences tested here are up to another 15%. Therefore, about 30% of the image is severely damaged. According to the theory described in Section 3.2.3, as low as 20% of the bits correct may be enough to retrieve the correct datum with a probability greater than 0.5, if the other 80% of the bits are set at random. In other words, more than 50% of the predictions of the SDM should be correct. In this experiment, the pixels are set to zero, not randomly, which is a worst case scenario. Therefore, the tolerance of the SDM was considerably reduced. But an interference of up to 12.5% was nonetheless supported regardless of the operation mode.

8.3 Reinforcement of an idea

Humans learn when something *different* captures their attention—i.e., something unexpected occurs, as explained in Chapter 2. Otherwise, memories are just held for a short time gap and soon vanish, possibly leaving no trace after they are gone. On the other hand, seeing the same image many times, or somehow experiencing the same situation many times, will help refine the corresponding mental model and reinforce its most important details. Since the SDM is a memory model biologically inspired, it is expected to mimic those behaviours, as described in Chapter 3.

In the original SDM model, writing a datum many times would contribute to increment and decre-

ment bit counters, thus reinforcing the experience. In the present work, however, all the models implement one shot learning and discarded the use of bit counters. Therefore, *reinforcement of an idea* can happen only by taking advantage of distribution of the data. Writing different copies of the same datum also makes it less likely to be forgotten. Therefore, the property of reinforcement of an idea of the SDM will be tested by recording multiple runs of the same sequence.

Reinforcement of an idea in the human brain also occurs when something is recalled from the memory. Recalling is associated with a kind of feedback mechanism, which rewrites the idea or sequence being recalled back into the memory, eventually strengthening its more salient features. That mechanism is more important for the study of forgetting behaviours, which is not relevant for the research question being studied.

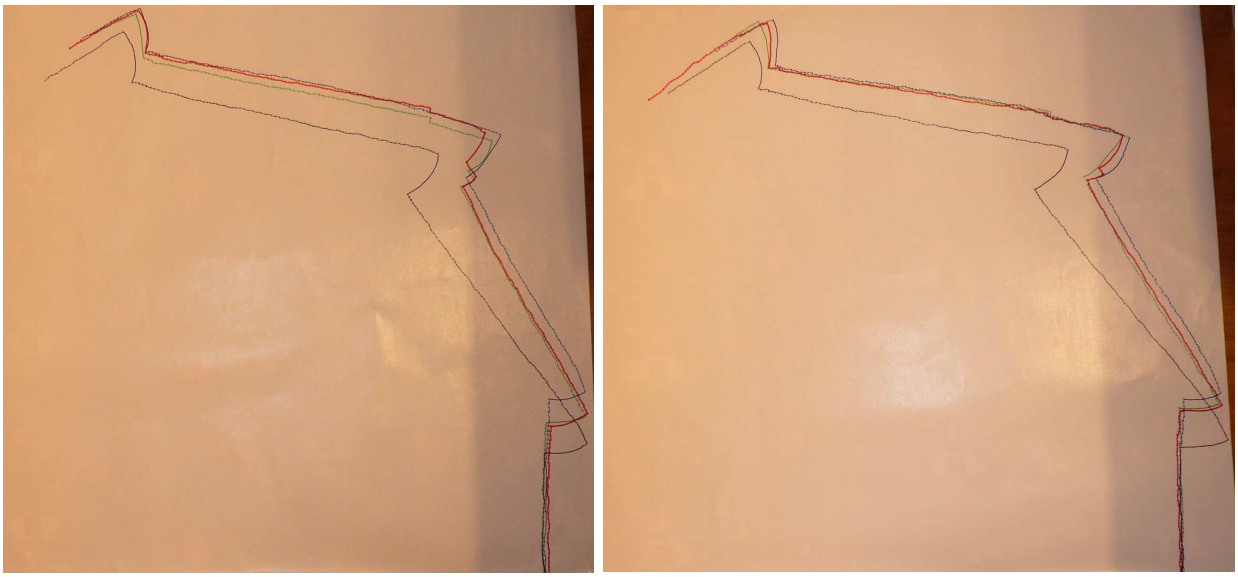
Experiments

To test the behaviour of the system with reinforcement of an idea, Lizard was taught the same path more than once. If the images were free from noise, it would most likely happen that the images of the second run would be very close to the images of the first run. In that situation, they would be partially or totally stored in the same hard locations. That would pose no problem for the performance of the system, although part of the additional overhead information (image id and image time) could be messed up. But, again, that should not compromise the performance of the system under normal circumstances. It is a feature of this kind of memory that some information may be damaged under particular circumstances. That must not compromise the ability of the system to make correct predictions most of the time. In the present case, it would not compromise its ability to successfully follow the same path, although it could result in more momentary localisation errors.

Results without distribution

Figure 8.5 shows an example of a path that was taught twice. The first time the path was described by a sequence of 56 images (black line). The other run is described by a longer sequence, consisting of 64 images (blue line). In both cases, it was guaranteed that the memory contained only one copy of each image. As the images show, there is a considerable gap between the two paths. Although they begin and end at similar points, the blue line is much longer than the black line. The black and blue lines were drawn by the pens attached to the robot when the memory was loaded with just one of the paths. When both sequences were loaded, the system tended to follow either one of the paths or somewhere in the middle of them, as shown by the green and red lines.

In Figure 8.5, the green lines were drawn when navigating using the arithmetic mode (Figure 8.5(a))



(a) Arithmetic and Sum-code. (b) Bitwise modes using NBC and Optimised code.

Figure 8.5: Paths learnt (blue and black) and performed with one copy of each image (green and red).

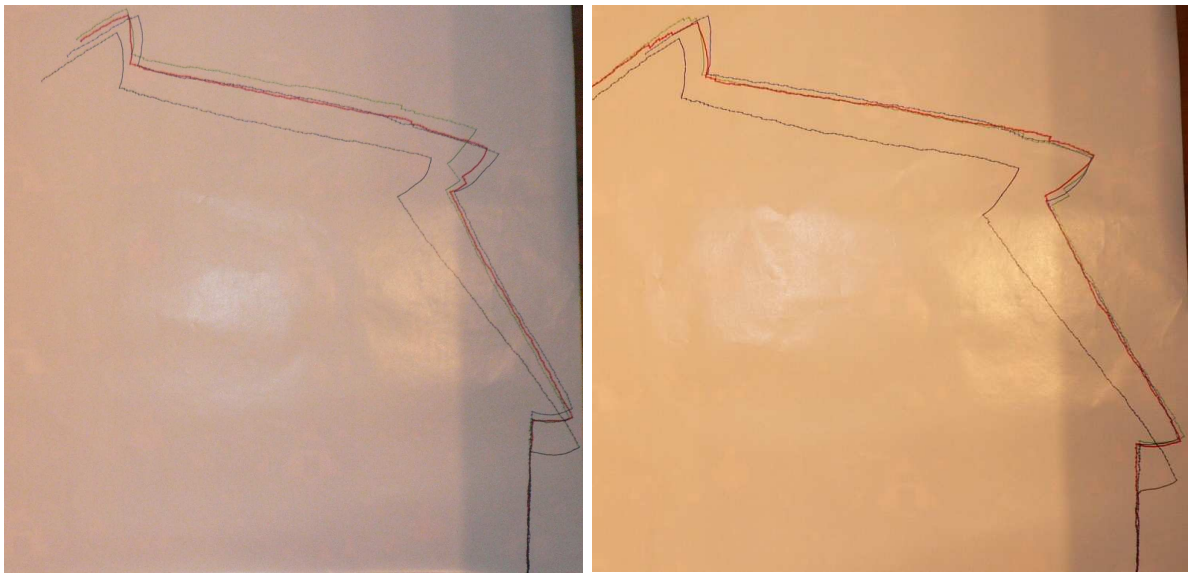
and when navigating using the bitwise mode and the natural binary code (Figure 8.5(b)). The red lines were drawn when navigating using a sum code of 9 graylevels (Figure 8.5(a)) and the optimised code (8.5(b)). The differences are marginal, as shown by the pictures, between all those modes. In all the modes, the system exhibits equivalent performance.

Results with distribution

The same tests described above were repeated enforcing distribution of the data. In the latter situation, it is guaranteed that each image had to be stored into at least three hard locations. Figure 8.6 shows the results obtained. In Figure 8.6(a), the green lines were drawn when navigating using the arithmetic mode. In Figure 8.6(b), the green line represents navigation using the bitwise mode and the natural binary code. The red lines were drawn using a sum code of 9 graylevels (Figure 8.6(a)) and the optimised code (8.6(b)). Again, the differences are marginal. With or without data distribution, regardless of the operation mode, the robot follows close to one of the paths or in between the lines.

8.4 Memory saturation

When the memory reaches saturation, part of the data stored are most likely to be corrupted. That is a feature of the SDM, and to a great extent it does not impair its ability to continue providing useful information. However, the algorithms that use that information must be robust enough to disambiguate the data if necessary, and/or deal with imprecise information. The problem is detailed in Section 8.4.2.



(a) Arithmetic and Sum-code. (b) Bitwise modes using NBC and Optimised code.

Figure 8.6: Paths learnt (blue and black) and performed with three copies of each image (green and red).

Also, as the memory becomes full, some information has to be deleted. Eventually, part of some sequences will be deleted. Section 8.4.3 discusses the problem and presents some tests and results.

8.4.1 The problem of saturation

According to the theory, the SDM should degrade gracefully and forget naturally. When some region of the memory reaches the saturation point, some vectors will be mixed up or deleted from the memory. In either case, data will most likely be corrupted. In the first case, some vectors of a sequence are altered. In the second case, part of a sequence is deleted because of lack of resources.

Distribution of the data can minimise the consequences of saturation, but sooner or later some sequences will most likely be disrupted. That is a feature of the memory that cannot be resolved without significantly changing its behaviour or characteristics. Therefore, to guarantee that navigation is as robust and safe as possible, it is important to find suitable navigation algorithms that can deal with the feature and still provide coherent information to the robot. The navigation algorithm must be good disambiguating the information that comes from the memory, possibly relying on the use of some auxiliary memory of the recent context. The recent context provides useful information about where the robot has been. Thus, the context can be used as a resource to rule out improbable candidate information, in case of multiple possibilities or sudden changes. But the algorithm cannot be inflexible up to the point where it may compromise the system's ability to solve the kidnapped robot problem, where the robot actually moves from one place to another and is expected to realise it.

8.4.2 Sequence disambiguation

When parts of a sequence have been deleted or corrupted, or even when there is a considerable amount of noise or other errors in the input, the output of the SDM may be erroneous. In those cases, the problem needs to be addressed. It is necessary to recognise the error and, if possible, to correct it, as already discussed in Section 3.6.8.

Storing the context

A k-folded memory approach, as defined in Section 3.6.8, will solve most of the problems under general circumstances, specially for large values of k, which make the system very aware of its recent context. However, it is by no means a general solution. For a k-folded memory, it is unpredictable which value of k will make the memory robust enough to never fail, and still flexible enough to respond to sudden changes in the environment. Additionally, larger values of k will also result in additional complexity and dimensionality of the system.

An alternative solution is to store a different *measure* of the context, that can somehow represent the past, but without having to store a possibly large number of episodes. That can be accomplished using some well-designed data structure specific for the purpose of storing the context. For example, a neural network, that starts empty and where all the state vectors (episodes) are summed. That approach is a general solution for the problem, limited only by the saturation of the neural network. It is largely described in [9], as already summarised in Section 3.6.8.

Context specific solution

In the present work, there is no need for a general solution. Even a k-folded memory is unnecessary. It is possible to disambiguate between a small subset of images using the additional overhead information stored with the images. The sequence number can be used for that purpose. The context can be defined as *the sequence the robot has been following recently.* For instance, if Lizard is following sequence x, and suddenly an input image activates two different images in the access radius, one image from sequence x and the other from sequence y, the latter is most likely to be wrong. Sequence y is *out of context.*

The approach above described is context-specific. It is not a general solution, such as the sequence machine or the k-folded memory described in Section 3.6.8, which are appropriate for possibly all kind of problems. However, it should work very well with the vectors that are being stored for vision-based navigation in the present work, and it has the advantage of adding just little additional complexity. Therefore, it is the solution that was implemented. An interesting characteristic of the method is that it

can be regarded as a segmentation of the address space, in many aspects similar to Jaeckel and Karlsson's idea of selecting a given number of coordinates, as described in Section 3.6.4. Jaeckel and Karlsson select a restrict subset of coordinates, and address the memory to retrieve solutions only within that subset. In the present work, the subset is the sequence that is being followed. However, Jaeckel and Karlsson select the coordinates before the prediction, while in the present work the coordinates are first used as a preference for disambiguation after the prediction being output from the SDM. That is a key difference between the two methods, because using Jaeckel and Karlsson's design it is not possible to solve the kidnapped robot problem. If the predictions are made within just a sequence, it is not possible to jump to another sequence. On the contrary, using sequence information after a non-restricted prediction, it is possible to follow the most likely sequence, but still keep track of the most probable predictions. That way, the context may be updated after each prediction. A large number of predictions within another sequence will signal a jump to another path.

The implementation of the solution above described requires an update of the navigation algorithm presented in Section 6.3. The new algorithm is called the Simple navigation Algorithm (SA), and it is shown in Algorithm 6. The differences between BA and SA are shown in bold.

The main point in which SA and BA differ is that SA has a context. Before entering the loop to follow the sequence, SA sets the context to the sequence number of the first prediction image. Then it enters the loop to follow the path. After each prediction, the sequence number of the predicted image is compared to the sequence number that is the most frequent in the context (i.e., the mode, according to the statistical definition of mode). If the predicted sequence is different from the mode, then the SDM is queried for the most likely predicted image within the context sequence. However, the predicted sequence number is pushed onto the context anyway, and the oldest sequence number is pushed out of the context. The length of the context will determine the rigidity of the algorithm. A long context will tie the robot to its history. It will be more robust to spurious errors. But it will also take longer to respond to changes such as that of the kidnapped robot.

Results

To compare the performance of the two algorithms, the memory was loaded with three different paths. Figures 8.7 to 8.10 show the results obtained in the different operation modes, for both algorithms. The three paths start at the bottom right corner of the image. The black lines represent the paths that the robot is expected to follow. Those lines were drawn by the robot itself when loaded with just one path, so that there would be no possible interferences of the contents of the other paths.

Figures 8.7 and 8.8 show the results obtained with the basic algorithm. It is clear that the robot was

Algorithm 6: Simple navigation algorithm, which maintains a context.

begin

$grabbedImg \longleftarrow$ current view;

$newImg \longleftarrow$ SDM prediction based on $grabbedImg$;

$sID \longleftarrow$ sequence number of $newImg$;

$imgID \longleftarrow$ image number of $newImg$ in sequence sID;

$lastInSequence \longleftarrow$ length of sequence sID;

context $\longleftarrow$ ***sID***;

repeat [follow while getting closer to the target]

Verify if kidnapped;

Determine expected *cSID*, according to *context*;

Add *sID* to context;

if *sID* $\neq$ ***cSID* then**

Get *newImg* from sequence *cSID*;

end

Warn user if $newImg$ predicted for the 5^{th} time or (error in similarity increases more than 35% and the robot has not been kidnapped);

$command \longleftarrow$ retrieve command from $newImg$;

Execute $command$;

if $command \neq MOVE\_RIGHT$ *and* $command \neq MOVE\_LEFT$ **then**

Correct lateral drift;

end

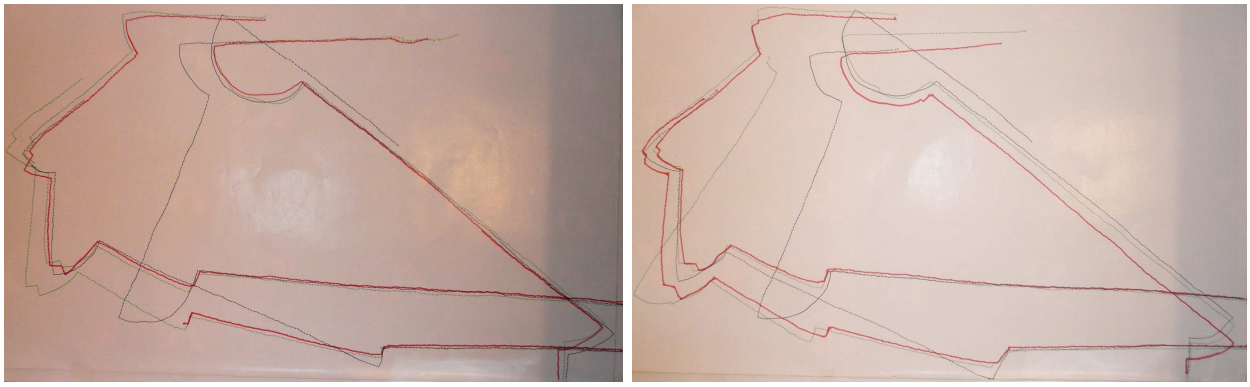
$grabbedImg \longleftarrow$ current view;

$newImg \longleftarrow$ SDM prediction based on $grabbedImg$;

$imgID \longleftarrow$ sequence number of $newImg$;

until $imgID \geq lastInSequence$;

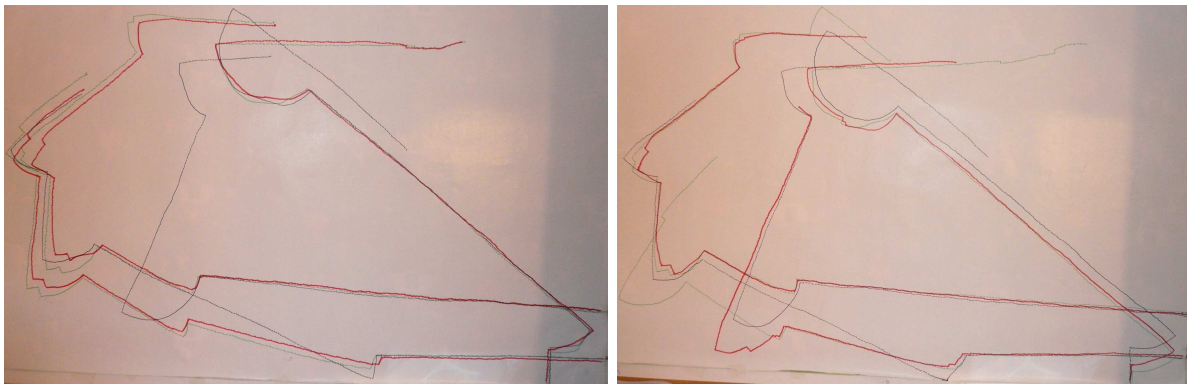
end



(a) Arithmetic (green) and Sum-code (red) modes.

(b) Bitwise modes using NBC (green) and Optimised code (red).

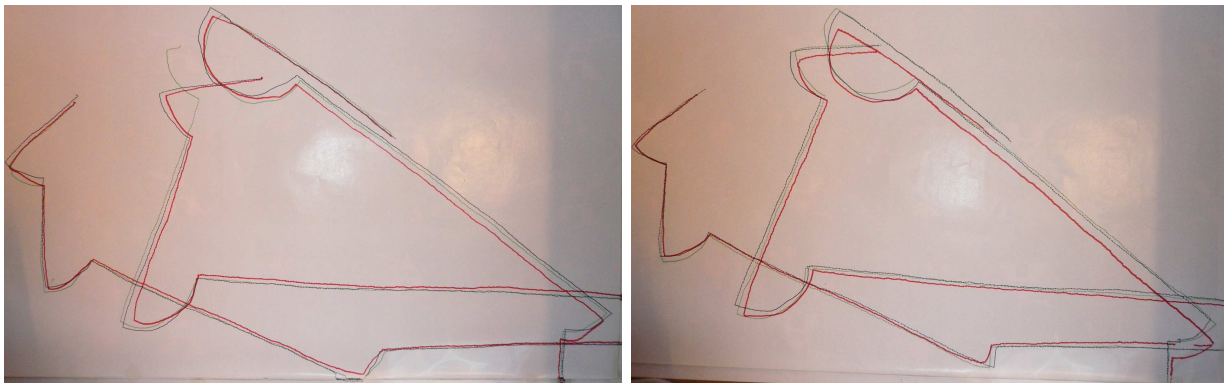
Figure 8.7: Trajectories drawn with one copy of each image and the basic navigation algorithm. The robot gets lost.



(a) Arithmetic (green) and Sum-code (red) modes.

(b) Bitwise modes using NBC (green) and Optimised code (red).

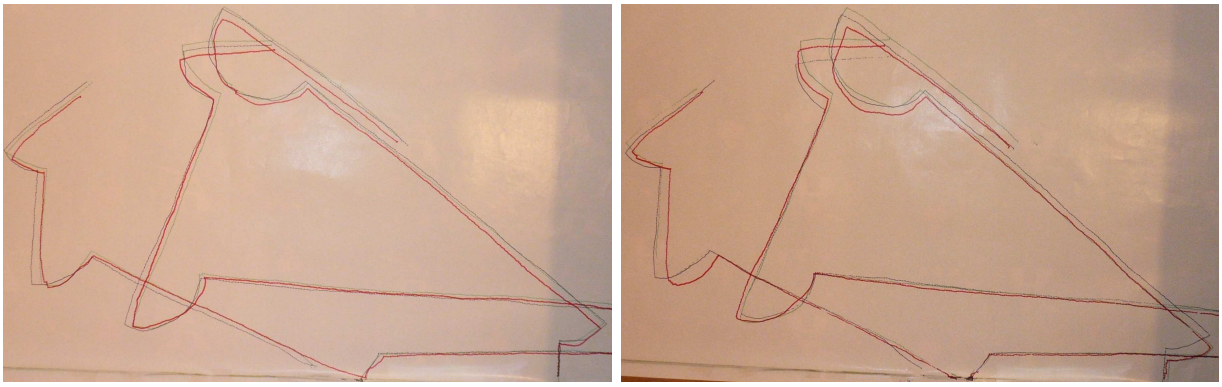
Figure 8.8: Trajectories drawn with three copies of each image and the basic navigation algorithm. The robot gets lost.



(a) Arithmetic (green) and Sum-code (red) modes.

(b) Bitwise modes using NBC (green) and Optimised code (red).

Figure 8.9: Trajectories drawn with one copy of each image and the simple algorithm. Using a context, the robot is able to disambiguate the sequences and does not get lost.



(a) Arithmetic (green) and Sum-code (red) modes.

(b) Bitwise modes using NBC (green) and Optimised code (red).

Figure 8.10: Trajectories drawn with three copies of each image and the simple algorithm. Using a context, the robot is able to disambiguate the sequences and does not get lost.

unable to successfully complete the job in all the situations. Sometimes it skipped from one sequence to another when the paths crossed each other or came very close. Other times it erroneously skipped from one sequence to another and ended up drawing a mix of the two closest paths. The behaviour seems to be independent from the operation mode or number of copies of each image that are stored into the memory. In some cases it even seemed unable to determine the end of the sequence. That happens because the last image of the path is never retrieved. Therefore, the robot continued following the commands of the closest images that were retrieved from the memory. It only stopped very close to the side of the experimental arena, when the collision avoidance algorithms prevented it from continuing against the wall.

Figures 8.9 and 8.10 show the results obtained with the simple algorithm. The context consisted of the sequence numbers of the last 15 predictions. As it can be seen, the simple algorithm exhibits a

much better performance. The robot was always able to follow any of the paths. Sometimes there are small drifts, but the robot neither got completely lost nor skipped from one sequence to another. The results shown were obtained with a context length of 15 steps, but a context length as small as 3 steps has proven to work in those particular cases.

8.4.3 Memory overflow

When it is necessary to store new information and no more hard locations are available, a memory overflow occurs and some hard locations have to be deleted. The SDM should continue to operate transparently under that circumstances. That behaviour was also tested, and the results are presented in this subsection.

The problem of memory overflow

A memory overflow occurs when the maximum number of physical locations has been reached. That does not happen in the original Kanerva model, which had a fixed number of hard locations. In that case, the limits of the memory are reached when too many vectors had to be stored around the same region, and the information gets mixed up. In the present implementations, which use the Randomised Reallocation Algorithm, as described in Section 3.6.6, new hard locations are added when necessary, up to a predefined limit. When that limit is reached, the memory can still be very sparse, but no more hard locations can be added. In that case, when a new datum needs to be stored, either it can be written to enough existing hard locations or some old hard locations have to be deleted, so that new hard locations can be added in the neighbourhood of the new datum. In either case, the SDM is going to "forget" some memories, which are rewritten or deleted to leave room to the newer information that has just arrived. The old memories are more exposed to that forgetting mechanism. Therefore, they will fade continuously over time, at the same pace as the new memories require more hard locations to be accommodated.

More complex mechanisms can be implemented to select which memories are to be deleted. One natural candidate is a model based on the time of the last access. Locations that have not been used for a long time are probably of little use. Therefore, it makes sense to discard them in order to free space for more important or up to date information. However, that requires that additional information is stored. Each hard location has to have a timestamp to store the time of the last access. Also, before deleting any hard location, it would be necessary to search all the memory to identify which was the oldest candidate for deletion. Such an algorithm requires both more memory and more processing power. On the other hand, the simple random selection is much simpler. The choice of the hard location to delete

is almost immediate. And since the probability of any hard location being selected increases over time, it is very likely that the oldest hard locations are actually the most natural candidates to be deleted under normal circumstances. Therefore, that is the method which was implemented.

Experiments

In order to test the behaviour of the memory when the maximum number of hard locations has been reached, it was loaded with three sequences. The first sequence comprised 72 images. The second sequence consisted of 61 images, and the third sequence contained 85 images. The total is, therefore, 218 different images. With the minimum number of active locations set to one and the maximum number of hard locations set to a number greater or equal to 218, it is guaranteed that all the images can be stored into the memory. With the minimum number of active locations set to three, in order to enforce data distribution, up to 654 hard locations may be required.

The memory was tested in all the operation modes, both with the minimum active locations set to one and to three. In both situations, the maximum number of hard locations was decreased in steps of 5%, up to the point where the robot was not able to follow any of the sequences. In all the situations, the context was set to 15 steps, and the learning rate for the arithmetic mode was set to 1, enforcing one shot learning.

Results

Table 8.4 shows the results obtained when the memory was loaded with just one copy of each image. The first column indicates the maximum number of hard locations, in percentage of the number of hard locations needed to store the complete sequences. The second column is the actual maximum number of hard locations. The next three columns indicate if the robot was actually able to successfully follow, respectively, paths 1, 2 or 3, in the arithmetic mode. "Y" means that it was able to follow the path with just a minimum error. "C" means that it was very close, but the drift was above its normal performance. An empty cell means that the robot got lost or erroneously switched to another path. The other nine columns show the same, for the other operation modes.

Table 8.5 shows the same experiments as Table 8.4, but in the latter the minimum number of hard locations was set to three copies of each image, thus enforcing distribution of the data to at least three hard locations.

The tables show that the system "forgets" the old paths as the new ones require more memory. Nonetheless, without distribution an overload of 15–20% was well tolerated, before the robot was unable to follow the first path that it had learnt. Distribution increases the tolerance to 20–25% before the

Table 8.4: Results of memory overflow tests, obtained for three paths and with one copy of each image stored. Y means the robot followed the path successfully, C means that it was close, and an empty cell means that it was unsuccessful.

Memory size	Memory size	Arithmetic			BW			OPC			BW09		
(%)	(hard loc.)	1	2	3	1	2	3	1	2	3	1	2	3
100	218	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y
95	207	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y
90	196	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y
85	185	Y	Y	Y	C	Y	Y	C	Y	Y	Y	Y	Y
80	174	C	Y	Y		Y	Y		Y	Y	Y	Y	Y
75	164		Y	Y		Y	Y		Y	Y		Y	Y
70	153		Y	Y		Y	Y		Y	Y		Y	Y
65	142		Y	Y			Y			Y		Y	Y
60	131			Y			Y			Y		Y	Y
55	120			Y			Y			Y			Y
50	109			Y			Y			Y			
45	98			Y									
40	87			Y									
35	76												

Table 8.5: Results of memory overflow tests, obtained for three paths and with three copies of each image stored. Y means the robot followed the path successfully, C means that it was close, and an empty cell means that it was unsuccessful.

Memory size	Memory size	Arithmetic			BW			OPC			BW09		
(%)	(hard loc.)	1	2	3	1	2	3	1	2	3	1	2	3
100	654	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y
95	621	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y
90	589	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y
85	556	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y
80	523	C	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y
75	491		Y	Y	Y	Y	Y	Y	Y	Y		Y	Y
70	458		Y	Y		Y	Y	Y	Y	Y		Y	Y
65	425		Y	Y		Y	Y		Y	Y		Y	Y
60	392		C	Y		Y	Y		Y	Y		Y	Y
55	360			Y		Y	Y		Y	Y		Y	Y
50	327			Y		Y	Y		Y	Y		Y	
45	294			Y			Y		Y	Y		Y	
40	262						Y			Y		C	
35	229											C	
30	196												

memories of the first path suffer serious damage. As for the second path, it shows that an overload of 30% is well tolerated before the robot's performance suffers noticeable damage without data distribution. Distribution increases the tolerance to 40–50%. If the sequences are stored in reverse order (i.e., seq. 3 is the first to be learnt and seq. 1 is the last), the result is similar: the first to be learnt is the first to be forgotten.

8.5 Retrieving a sequence in reverse order

Learning a path moving in one direction and following it in the opposite direction, using only visual information, is trivial for humans. However, it may be a challenging problem for a robot. But using the right hardware and a SDM, the problem can be solved. This section addresses the problem and describes the tests and results.

The problem of backwards motion

One common problem in robotics is that of following a path backwards. It is very different to follow a path in one direction or in the opposite. The sensorial information acquired while learning a path by following it in one direction is completely different and useless if the sensors are sensing in the opposite direction. As a consequence, it is often required that a robot learns the same path twice, once in each direction, because having learnt it in one direction the robot is not able to follow it in the opposite direction. A very common solution to that problem, which avoids learning the same path twice, is to sense back, not ahead, when following a path in the opposite direction. However, in that case, there is the problem of going back in the sequence of events in the memory. When following a path in the same direction, it is necessary to determine the successor of each vector in the sequence of memories. But when navigating backwards, it is necessary to determine the predecessor and perform the opposite command. For example, if the robot was at point A and moved to point B by turning left, when learning the path, when following the path backwards at point B it has to turn right to get to point A. Another common solution is the use of omnidirectional images, such as described in Section 4.3. In that case the robot has complete information of the surroundings. To navigate backwards it is necessary to rotate the images and invert motor commands, additionally to retrieve the predecessor of each image of the sequence.

The SDM's versatility makes it possible to follow a sequence in either direction: from beginning to end, or from end to beginning. In the case of robot navigation, it shall confer on the robot the ability to follow a sequence backwards, if the robot has the necessary hardware specifications. In the case of Lizard, it only has a camera on the front, but it can move back with the same precision that it

moves forward. Therefore, it is possible to implement backwards navigation capabilities in the system by exploring the SDM's ability to retrieve a sequence in reverse order and using the robot's camera to sense back instead of front.

Experiments and results

Using the SDM with the input vectors as specified in Equation 6.1, it is possible to retrieve a sequence in either direction. In that case, it is possible to use the overhead information to determine im_{i-1} from im_i of any sequence. Since the robot is also able to move back, it is possible to write another algorithm to follow a sequence backwards. Backwards navigation is not as precise as forward motion, because the camera is on the front of the robot and the drift error correction is not as efficient in correcting the heading when there are small drifts.

When heading to the end of the sequence, Lizard captures an image, retrieves from the memory the closest image and the motion it performed during the learning stage, then performs the same action. In the backwards motion, the algorithm has to be adapted. In that case, Lizard captures an image; retrieves from the memory the closest image and its number in the sequence im_i; then, it retrieves from the memory the image im_{i-1}, as well as the associated motion command m_{i-1}. The motion associated with image im_{i-1} is the best prediction of the motion that took the robot from seeing image im_{i-1} to im_i. Therefore, Lizard performs the inverse action: if the stored motion was 'f' (forward), it will move back; if it was 'b' (back), it will move forward; if it was 'r' (right), it will turn left; if it was 'l' (left), it will turn right. By performing the inverse of each action, it is expected to get back to the point where the sequence starts, step by step.

Figure 8.11 shows an example path that was performed forward and backwards. The black line was drawn when Lizard was moving forward. The green line was drawn when Lizard was moving backwards. The result shown was obtained using the arithmetic mode, with three copies of each image. However, the operation mode or the algorithm used apparently do not have any special impact on the performance of the system for the specific problem being studied. It can be seen, in the figure, that the precision in backwards motion is remarkably poorer than in forward motion. Lateral drifts correction is not as effective, for it is done based on images which are captured by the camera at the front of the robot, which in that case is its back. Additionally, the effect of the pen attached to the robot is also more important: the interference of an object being dragged behind is smaller than the same object being pushed ahead. Nonetheless, the experiments and results show that the hypothesis is correct and it works quite well in practice.

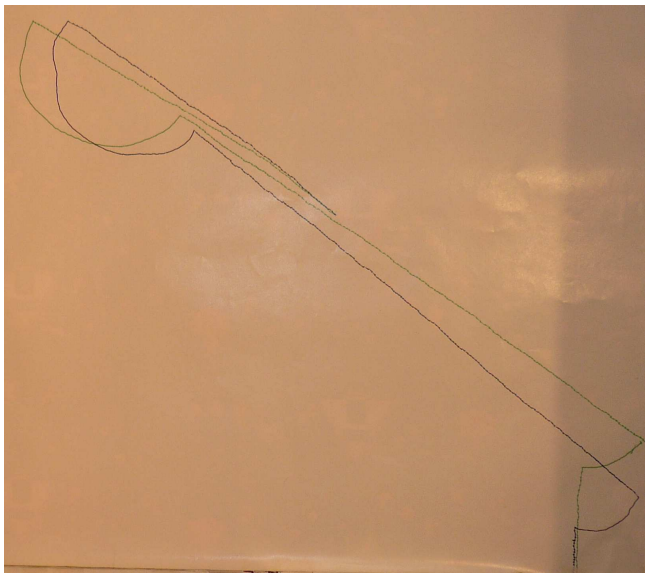


Figure 8.11: Paths of the robot forward (black line) and backwards (green line), in the arithmetic mode and data distribution to at least three locations.

8.6 Outdoor experiments

Lizard is a very small robot, appropriate for indoors use only. However, in order to have an idea of its performance out of closed arenas, additional experiments were made in an outdoor balcony, depicted in Figure 8.12. The balcony is 7.5 × 3.2 m^2 and the pavement is made of granite blocks. At the East side there is the house the balcony belongs to. At the North side there are trees, a short wall and a pillar—and the same for the South side. At the West side there are more large trees, as well as three large vases with baby trees.

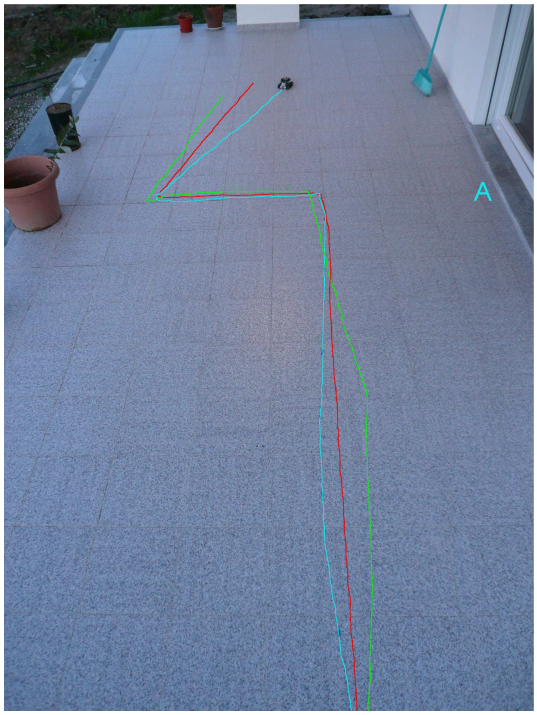
The experiments were performed during a sunny winter afternoon. Due to the length of the path and the robot's slow speed of operation, the system was optimised for speed rather than performance: the experiments were executed using the arithmetic mode, with equalised images and storing only one copy of each image into the memory. The length of the robot's step was also quadrupled (set to approximately 4.5 cm), so that Lizard took only about 15 minutes to complete the path during the autonomous run.

The path was taught at approximately 14h30m. It had a length of approximately 7 meters and was described by 147 pictures. While following the path, the field of view of the robot was filled with different objects, such as walls and a pillar of the house, trees, vases with flowers and baby trees and clouds in the sky. There were also shadows of many different objects, including the author's shadow. Those shadows moved with the sun and the wind, and the author's shadow moved even swifter. Figure 8.13 shows three images captured by the robot while learning the path.

Figure 8.12(a) shows the trajectory that the robot followed, running autonomously at 15h00m, some

(a) Path drawn by the robot starting at approximately 15h, with the shadows pointing Northeast.

(b) The green path was drawn starting at approximately 16h, with the shadows pointing almost East.



(c) The blue path was drawn starting at 17h, when the sun was almost set.

Figure 8.12: Trajectories followed by the robot at different hours of the day.

(a) Image 1. (b) Image 52. (c) Image 87.

Figure 8.13: Images captured by the robot while learning the path in the outdoor experiment.

minutes after finishing learning the path. In the image, the path has been manually highlighted, because the pen leaves only a very faint line on the granite. As the picture shows, at that time the scenario was well illuminated and there were sharp shadows pointing North-east. Figure 8.14(a) shows a picture taken by the author when the robot was drawing the path, at approximately 15h10m. The picture was taken at approximately the height of the robot's camera, with the camera on the floor, at the point marked with letter A in Figure 8.12(a). While following the path, 43 momentary localisation errors were generated. Since the path is described by 147 images, during the autonomous run the robot is expected to perform at least $147 \times 16 = 2352$ steps. Hence, 43 MLEs is just 1.83%, which is almost the same error rate that measured indoors for equivalent settings (1.18% measured in Section 6.5.3).

One hour later, at 16h00m, the robot was made to follow the same path again. The trajectory followed is shown in Figure 8.12(b), marked by the green line. As the picture shows, the shadows are pointing in a different angle, because the sun had moved to the West. There is also much less light, because the sun was setting fast and there were also some clouds appearing on the sky. Figure 8.14(b) shows a picture taken from the point marked with letter A in Figure 8.12(b). In that experiment, 103 momentary localisation errors were counted. That is an error rate greater than 4.38%, which is more than twice the error rate measured at 15h00m. Additionally, the robot stopped at a distance of approximately 20 cm from the end of the path that had been marked one hour before. It should be noted that a 20 cm error in distance is a good achievement, since there are no clear landmarks in the last picture to help distinguish it from the other close pictures. Besides, it is an absolute error, and not an incremental one. If the trajectory was longer, the error might be similar or less.

At 17h00m the robot was made to follow the same path one more time. Figure 8.12(c) shows the result. When the robot had finished, the sun was very low on the horizon and the sky was very cloudy. Hence, there was much less light and there are almost no shadows in the picture. Nonetheless, the robot successfully completed the path and only 81 momentary localisation errors were counted. That is an error rate of 3.44%, which is larger than that obtained indoors, but less than that measured at 16h00m, when the sun was projecting much more shadows into the scene. Again, the robot stopped at a distance of approximately 20 cm from the end of the first path.

(a) Picture taken from the point marked with letter A in Figure 8.12(a).

(b) Picture taken from the point marked with letter A in Figure 8.12(b).

(c) Picture taken from the point marked with letter A in Figure 8.12(c).

Figure 8.14: Pictures taken at the height of the robot's camera, while it was following the paths shown in Figure 8.12, at different hours of the day, with different shadows and illumination conditions.

8.7 Summary

This chapter describes some experiments that were done to assess the performance of the system under typically challenging situations.

The first problem studied was that of the kidnapped robot, which happens when a robot is suddenly removed from one location to another. Lizard performs navigation based on sequences of images grabbed from a small camera on its front. Therefore, it is naturally immune to being kidnapped: as long as it is moved to a known location, the environment remains unchanged and the context stored is small enough, it is always able to localise itself.

The second problem studied was that of illumination changes. Illumination changes are a kind of noise to the SDM. The system is very tolerant to large variations of light, using the arithmetic and the sum code operation modes. Another problem that can also be regarded as a kind of noise is that of scenario changes. The tests performed show that Lizard is very tolerant to changes that occupy, in some cases, more than 12% of the whole image.

An important characteristic of the SDM is that of reinforcing an idea. Since the memory is able to naturally forget over time, ideas that are reinforced are less likely to be forgotten. In the case of navigation based on sequences of acquired images, reinforcement of an idea means that a sequence can be taught more than once. Some images of the second and posterior runs will most probably overlap already learnt images, others will be stored independently. The results show that when following paths learnt more than once, the robot tends to follow close to one of the paths learnt, or in between the two versions of the path.

Another problem also studied is that of memory saturation. Saturation may cause undesired mix and corruption of data in the memory. Therefore, it may be necessary to disambiguate the information, based on the context. In the case of navigation based on sequences of views, the context can be just the number of the sequence that is being followed. A long context makes the system less flexible. A short context may make it less robust. The results show that even a short context can make the system much more robust.

Memory overflow is another serious problem. When the memory reaches its limit, it has to forget something in order to accommodate new information. The results obtained show that the SDM forgets the older data rather than the more recent, and it is highly tolerant to overflows—up to 40% of its size, before being unable to predict the first sequence.

The problem of backwards motion was also studied. The SDM is very flexible, so it is possible to retrieve the predecessor of each element of a sequence. Using the right hardware, it is, therefore, possible

to run a path in a direction opposite to that in which it was learnt. The results show that the system worked as expected, with just a small change to the navigation algorithm.

Finally, outdoors navigation was also experimented. Lizard was taught a path and made to follow it at different times of the day, with moving shadows of neighbouring objects. It was able to successfully follow the path until the sun was completely set.

The experiments show that, in general, the system exhibits a better performance using the arithmetic and sum code modes, with some distribution of the data. It is very robust, even facing difficult situations, both indoors and outdoors.

Chapter 9

Navigation using view sequences

To achieve great things, two things are needed; a plan, and not quite enough time.
(Leonard Bernstein, American composer and conductor)

Contents

Previous chapters described methods to improve image quality and make the most of a SDM, implementing a system that navigates a robot based on sequences of images stored into the memory. However, the algorithms previously described do not consider the problem of planning paths. Path planning is a natural human ability and a necessary ability for any autonomous robot, unless it is confined to a very small and limited environment. This chapter describes the problem in more detail and presents an algorithm for path planning based on sequences of images and a SDM. Part of this work has been presented for the first time in 2010 [70].

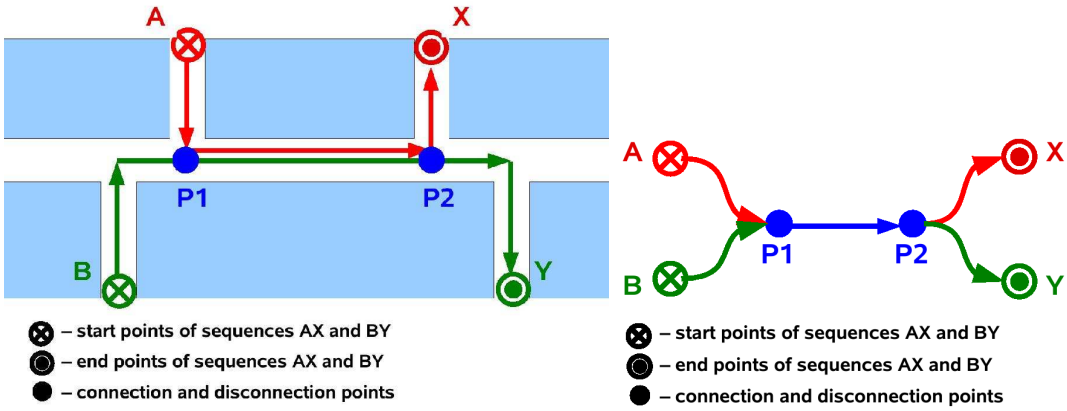
9.1 The problem of path planning

Path planning is necessary to solve problems such as that of skipping from one path to another, in order to follow a shorter path or to reach some goal which was not reachable following the previous path. That requires goal-directed navigation, as well as the use of a topological map and a path planning algorithm.

9.1.1 The problem

The "teach and follow" approach *per se* is very simple and powerful. However, using that approach as described in the previous chapters requires that every sequence be unique and unrelated to other sequences. It is not possible to detect when two paths cross each other, come together or split apart. Problems such as that of following path ABCDE and XBCDE have to be dealt with as if they were unrelated, despite the fact that the sequences have 4 elements in common (and those elements represent 80% of each sequence).

For robust navigation and route planning, it is necessary to extend the learning and navigation algorithms, as well as implement a path planning strategy. The robot must be able to perform tasks such as detection of connection points between paths that converge to a common segment, and disambiguation when there are paths that diverge after a common segment. Using those abilities, the robot will be able to avoid learning common segments many times, and also plan routes more efficiently. To accomplish that goal, the robot must be able to identify and merge common segments, such as segment P1–P2 shown in Figure 9.1. In the figure, paths A–X and B–Y share a common segment P1–P2. A smart learning algorithm must be able to detect the confluence at P1 and the divergence at P2. That way, segment P1–P2 needs to be stored in the memory only once, instead of being repeated. Using the strategies described in the previous chapters, images taken in the segment P1–P2 can only be assigned either to path A–X or B–Y. In consequence, at least one of the paths has to be ill described, because information of the second path will overwrite information of the first path, or information of the first path will prevent information of the second path from being written. While following an ill described path, the system can encounter many difficulties. For example, it can erroneously skip from one sequence to another, possibly loosing its goal and/or being unable to complete the assigned task. The use of a context as described in Section 8.4.2 can minimise that problem. However, an intelligent algorithm will be powerful enough to recognise the common segment, thus avoiding conflicting and unnecessary information in the memory. Additionally, an algorithm that identifies and merges those points is also powerful enough to fulfil the requirements necessary for higher level path planning, using algorithms such as A*.



(a) The environment.

(b) Topological map built by the robot.

Figure 9.1: Example of paths that have a common segment.

9.1.2 Building a topological map

Figure 9.1(a) shows a situation in which two paths share a common segment. In the paths A–X and B–Y, the segment P1–P2 is common. If the SDM memorises the segment P1–P2 for path A–X, for example, then it does not need to memorise it again for the path B–Y. It can build a representation such as the one depicted in Figure 9.1(b). When learning the second path, the system only needs to memorise it until point P1. At point P1 it can store an association between the first and the second paths and skip the memorisation of all the images until point P2. At point P2, it should again record a connection between paths 1 and 2. After P2, it must continue memorising the new path again until the end. Following that procedure, it builds a valid representation of the environment and stores connection points between the known paths. At those connection points, the robot can skip from one path to the other. That is a kind of topological representation of the environment, which is very powerful for path planning and navigation.

The main problem with the approach described is that of detecting the points where the paths come together or split up. Although that is apparently a simple problem, it is very difficult to solve based only on visual information coming from a single front view. In the present work, a simple algorithm was established to detect merging and splitting points based on visual similarity of a subset of consecutive images. Using the example depicted in Figure 9.1, the point where the paths come together (marked point P1) can be detected after a *reasonable* number of images of path A–X have been retrieved, when the robot is learning path B–Y. When that happens, the robot stores the connection in its working memory and stops learning path B–Y. From that point onwards, it keeps monitoring if it is following the same path that it has learnt. After a *reasonable* number of predictions have failed, it adds another connection point to the graph and starts learning again the new path. Therefore, the system will store the whole path A–X. But path B–Y will not be stored completely. The SDM will learn segments B–P1

and P2–Y. Then the system will be able to recover path B–Y as the sum of three segments: B–P1; then P1–P2 using the information stored when learning path A–X; and finally P2–Y. Complete information about the merging and splitting points can be stored in a data structure such as a linked list or table, thus its processing is very fast and can be done in real time. In the present work, a table is used.

In the experiments with the SDM, it was determined that a number of 3 to 5 consecutive images within the access radius usually sufficed to establish a connection point, and 3 to 5 images out of the access radius was a good indicator that the paths were diverging again. Therefore, the pattern recognition features of the SDM can be used in the mapping algorithm to detect the important nodes in the graph. The complete mapping strategy can be summarised as follows:

- When learning a new sequence S_k, if at point C_i^k, n or more consecutive images of S_k retrieve learnt images of sequence S_y, establish a node connecting sequences S_k and S_y at point C_i^k, so that C_i^k is recognised as being coincident with point C_j^y and stop learning;

- Proceed until there are at least q consecutive images that do not retrieve images of sequence S_y. At that point, create a *disconnection* point D_m^k, coincident with D_p^y and resume learning the new path.

In the description above, the capital letter C is used to represent *connection* points, and the capital letter D is used to represent *disconnection* points. The exponents ^{k} and ^{y} mean that the points belong to the respective sequences S_k and S_y, and the indices $_i$, $_j$, $_m$ and $_p$ mark images of those sequences.

It should be noted that there is no limit on the number of associations *per point*. For example, the point C_i^k can be simultaneously associated with point C_j^y and C_a^t, if three different paths come together at a given point. The graph is built incrementally, and can contain any number of associations for any given point. Connection points can also be associated with disconnection points, if a path segment terminates where another path segment begins.

Figure 9.1(b) shows the topological map that is created to represent the example paths shown in Figure 9.1(a). As the diagrams show, the paths are split in the topological representation, so that at point P1, regardless of where the robot comes from, it will only have one way to proceed, that is following path P1–P2. At point P2, it has to make a decision on whether to follow towards point X or towards point Y. That requires that it must have a goal, and decide to follow the path that will most likely guide it towards the goal.

9.1.3 Goal directed navigation

If all the paths were unique and independent from each other, the robot's only achievable goals would be inside its current path. However, using a graph with connection points between paths it is possible to pursuit other goals that lay in the known paths. Traversing the graph the robot can determine which end points can be reached. The complete list of reachable goals comprises all the end points of all the sequences in the graph, as well as any other points in the middle of the paths.

In the example depicted in Figure 9.1(a), the definition of a goal is actually necessary at point P2. The decision on whether to follow towards X or towards Y depends on where the robot is set to go. The navigation algorithm can, therefore, be extended to include the definition of a goal. Besides, since the robot can move backwards, the list of goals can comprise both goals which require the robot to move ahead, and also goals which require it to move backwards at some point.

In the example depicted in Figure 9.1, the robot can pursuit several goals. Let's assume that it is placed somewhere in the middle of the path A–P1, and that it has learnt all the paths marked and also constructed the graph shown in Figure 9.1(b). After acquiring one or more images, it is able to "localise" itself correctly and recognise that it is somewhere in the path A–P1. Using the topological representation and the sequences of images it is possible to infer that two different end points can be reached: X and Y. Actually, since the robot can also move backwards, the points A and B can also be reached from anywhere in the map. It is possible to determine which end points can be reached, and the shortest path to follow, traversing the graph using search algorithms such as A*, or other search methods. The number of images of each segment can be used as an estimation of the length or cost of that segment. In the experimental platform used, the possible goals are presented by the computer and chosen by the user. But that behaviour can easily be adopted by a more sophisticated automated system, so that the robot can choose the goal points by itself, as discussed in more detail in Section 9.3.

9.2 Navigation experiments and results

The algorithm described in Section 9.1.3 was tested, first with two sequences taught in the arena testbed surrounded by the countryside scenario described in Chapter 5. Later, the scenario was changed to mimic a urban environment with some scenario changes. The experiments and results are described in this section. Unless stated otherwise, the operation mode used by the memory is the default arithmetic mode, for all the experiments.

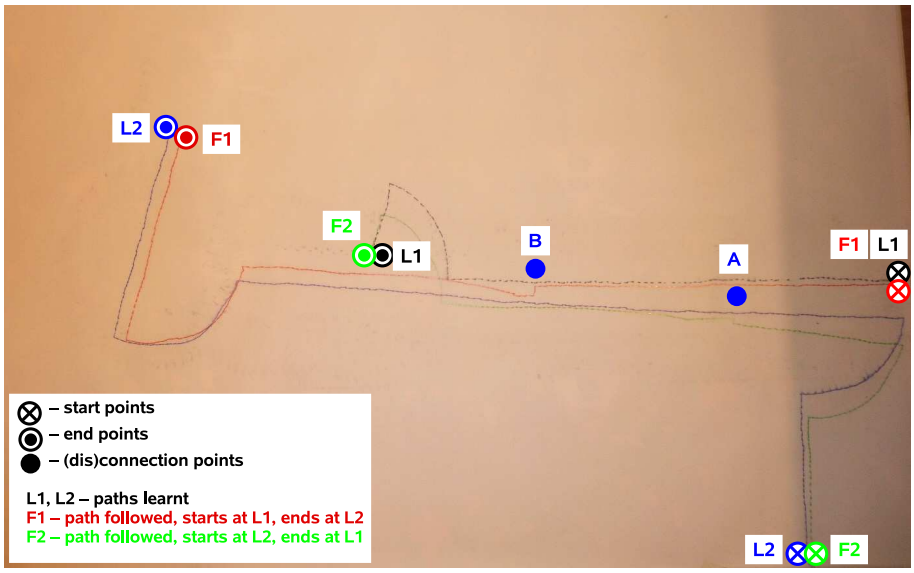


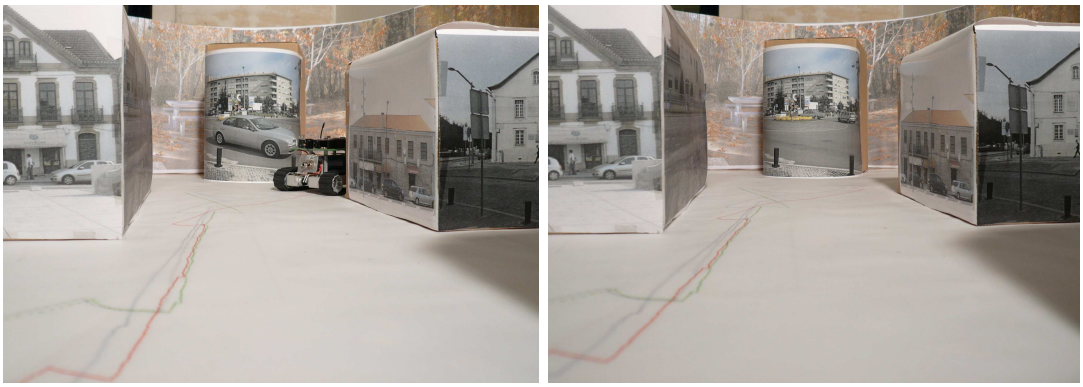
Figure 9.2: Lines showing the paths taught and followed.

9.2.1 Experiments in the empty arena simulating a country environment

The first test that was performed consisted in analysing the behaviour of the navigation algorithm in the empty arena. The surrounding wall showed a composition of images of mountain views, as described in more detail in Chapter 5.1.3.

Figure 9.2 shows an example of the results obtained. In the example shown, the robot was first taught paths L1 and L2, independently from one another. That was guaranteed by teaching path L1 first; then erasing L1 before teaching L2. That procedure made it possible to reproduce L1 and L2 exactly when it was necessary for some experiments. Later, the memory was loaded with the two sequences simultaneously. As expected, connection points were established at points A and B. The minimum *overlapping* images required for establishing a connection point was set to 3 consecutive images. The minimum number of different images necessary for splitting the paths at point B was also set to 3 consecutive images out of the access radius. As the lines in the picture show, using the connection points the robot was able to start at the beginning of the taught sequence L1 and finish at the end of the taught sequence L2, and vice-versa. Regardless of its starting point, at point A it always defaulted to the only known path L1. That is the reason why there is a small arc at point A in path F2. That arc, in fact, represents an adjustment of the heading direction when the robot defaulted to path L1.

The direction the robot takes at point B depends on its goal. If the goal is to follow towards the end of path L1, it continues along that path. If the goal is to follow towards the end of path L2, it will disambiguate the predictions to retrieve only images from path L2. That behaviour also explains the changes in direction that appear in the red line (F1) at point B. The arcs were drawn when the robot



(a) First image. (b) Second image, a few seconds later.

Figure 9.3: Typical city view, where the traffic turn is temporarily occluded by passing cars.

started at path L1, but with the goal of reaching the end of path L2. At point B it had the opportunity and did so, adjusting the direction slightly to the right.

9.2.2 Experiments in a reconstruction of a urban environment

In a second, and possibly more challenging experiment, the scenario was filled with images mimicking a typical city environment[1]. Urban environments change very often. Ideally, the robot should learn one path in a urban environment but still be able to follow it in case there are small changes, up to an *acceptable* level. For example, Figure 9.3 shows two pictures of a traffic turn, taken only a few seconds one after the other. Although the remaining scenario holds, one picture captures only cars in background. The other picture captures a side view of a different car in foreground.

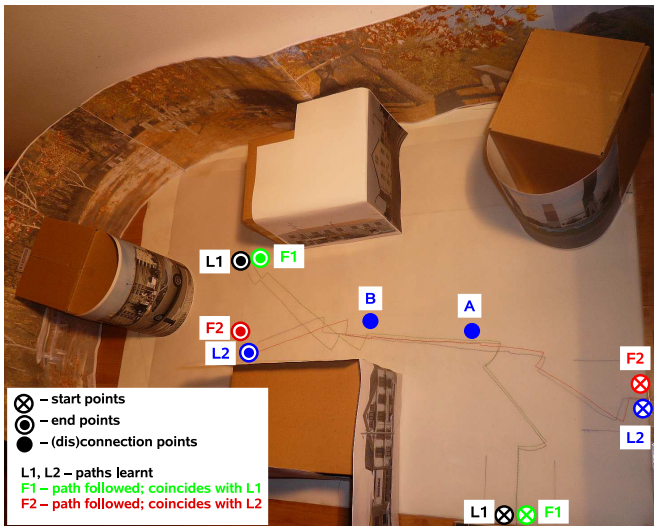
Figure 9.4 shows the results of tests performed in a fake urban environment[2]. Figure 9.4(a) shows the first scenario, where the robot was taught. In that scenario the robot, while navigating in the direction A–B, is guided essentially by the image of the traffic turn without the car. The black and blue lines correspond to paths named L1 and L2, respectively. As the pictures show, L1 and L2 share a common segment A–B. When the memory is loaded with the two paths simultaneously, a connection is created between image 38–40 of sequence L1, and image 30–32 of sequence L2 (the exact image may vary from one experiment to another, due to the randomness characteristics of the problem). That is point A of the map. Point B is around image 48 of path L1, and image 38–39 of sequence L2.

In the second part of the same experiment, the picture of the traffic turn was replaced by the other picture of the same place with the car in foreground, and the robot was made to follow the same paths. It was made to follow paths L1 first and then L2, while the memory was loaded with both paths. When

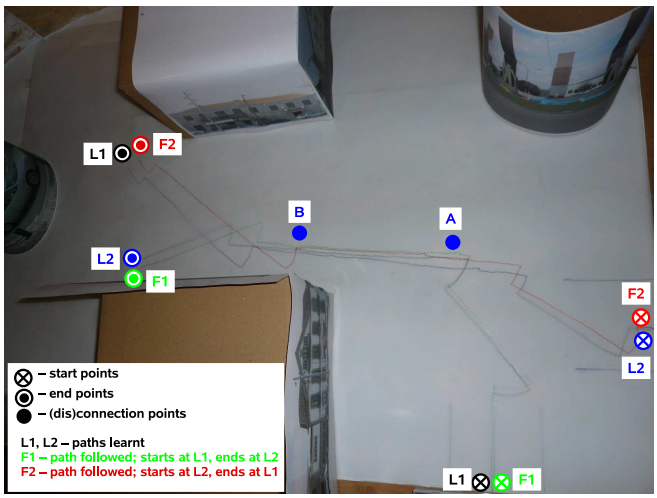
[1]The experiments were not run in a real city environment due to the small dimensions of the robot. However, there is apparently no reason, other than the scale, for it to behave differently in a real environment.

[2]Pictures taken in main streets of Oliveira do Hospital, Portugal.

(a) First scenario, showing the paths followed successfully.



(b) Scenario with changes, also showing paths successfully followed (F1 and F2).



(c) Scenario with changes, showing paths where the robot switches from one path to another at disconnection point B.

Figure 9.4: Paths learnt (blue and black) and followed, after small scenario changes.

the robot started at L1, the goal set was the last image of L1. Then the same was repeated for path L2. As Figure 9.4(b) shows, it was able to successfully complete the tasks. There are just small errors, because the robot actually anticipated the direction changes after the node B. That is a consequence of the disambiguation that has to occur at point B. Using a higher operation frequency and a smaller access radius in the SDM, the problem may be minimised.

In the third part of the experiment, the robot was made to start at the beginning of path L1 with the goal of reaching the end of L2, and vice-versa. Figure 9.4(c) shows the results in that cases. Again, as the pictures show, the robot was able to successfully switch from one path to another, at the disconnection point, with a minimum error.

9.3 Hypothetical examples

Although out of the scope of the present work, the system presented here can easily be extended. Using the proper hardware, and the algorithms developed in this project, it is very straightforward to build a functional robot for tasks such as surveillance or simple services.

Surveillance

Surveillance is an example of a task in which the use of autonomous robots is deemed necessary and appropriate. There are three main reasons: it is boring; it is simple to do; and it is expensive when done by humans. Besides, in some cases it can also be hazardous for human beings.

As for being boring, the problem is that the vigilants just have to stand and wait to see what happens. Most of the time, nothing happens. But the vigilants just have to wait, because they cannot predict if anything will happen or not. It is necessary that they are there, if not for anything else, then just to wait and make sure that nothing happens. Many humans just do not like to wait endless hours watching, sometimes on foot because they must not be seated. In some cases, they can do something to entertain them, but often that is not possible for some reason. For a robot, that is no problem. Robots can wait hours, days or forever, without getting bored.

As for being a simple task, indeed few humans would say it is not. The vigilant only has to stay attentive to unusual sights or sounds. Depending on the complexity of the environment, that could also be a simple task for a robot. For example, in an environment where no motion is expected, any movement could be reason for a warning. An intelligent robot would be able to distinguish between harmless movements, such as the motion of tree leaves in the wind, and suspect motions, such as a human running. The SDM can be used to tell suspect images apart, since it is a good pattern matching

tool. Other sensors or techniques can also be used to improve the performance of the system (e.g., infrared sensors as heat detectors, face recognition techniques, etc.).

As for being expensive, indeed human surveillance requires at least one person. Companies that have to hire vigilants would rather avoid that cost if there was a safe alternative. Often it also requires some equipment, such as a closed circuit TV (CCTV) system. Using a CCTV system large areas can be monitored by less people than if all the work had to be done by people moving around. However, humans can perform other tasks, such as turning off computers, lights, air conditioning or other equipments that may have been left on by mistake. An intelligent robot might do the same as humans do, and the maintenance of a robot should be less expensive than the salary of a human.

In some cases, there is the additional problem of safety. Surveillance is often necessary to prevent theft and vandalism acts. Violent theives or vandals may injure or kill people, vigilants included. Damage to a robot is more acceptable, from the human point of view, than injuries or death of a human vigilant.

In summary, if the job is done by intelligent robots, many of those problems faced by vigilants can be overcome or minimised. First, robots do not get bored, regardless of how long they have to stay or wait. Second, the most significant cost is that of purchasing the robot, which, over time, should be much cheaper than that of hiring a qualified person. Third, there are not strong concerns about the safety of a machine, since the human factor is removed out of the equation—robots can easily be repaired or replaced, contrary to humans.

A surveillance robot based on the use of a SDM has many advantages. One advantage is that the SDM itself can be used to detect the presence of an intruder, based on visual or auditory information. Specially during night surveillance in outdoors environments, as the sun sets and rises, it is necessary to compensate the illumination changes. But sudden changes, such as the light of a lantern or someone switching on the lights, have to be detected as an intrusion. Using a SDM, the illumination can be monitored and the compensation for sun light can be made over time using an algorithm such as the dynamic radius described in Section 6.2. That can be accomplished in a natural way, just as Lizard does to compensate for the noise levels present in the images. Sounds can also be monitored, and unexpected patterns can be detected in the same way that is done for illumination. The idea can be extended to other sensorial readings, namely passive infrared for heat detection.

Another advantage of using a SDM as the basis for a surveillance robot is that the SDM can deal naturally with sequences. Hence, it is just natural for a surveillance robot based on a SDM to be programmed to follow intruders and record the sequence of images and sounds grabbed during the occurrence. That way it is possible to have a complete movie, without discontinuities that may occur

in situations where CCTV cameras are used. Such a robot could be programmed with an algorithm to follow any unexpected person. Of course, there is the problem that the robot could be stolen itself, if it is mobile and small. The probability of theft could be minimised in various ways. First, the robot could be programmed to trigger an alarm in case it detected an intruder or was kidnapped. The alarm, as well as the presence of the robot itself, would most probably discourage many intruders. As an additional level of security, the robot could be programmed to upload relevant images or sounds to a remote website via a wireless link. In that case, even if the robot was badly damaged or stolen, evidence of the facts would be safely stored. Yet another improvement to the robot could be a way for it to make clear its localisation. For example, the robot could use a GSM modem to send a SMS with its GPS coordinates from time to time, or even inertial data that could be used to reconstruct its trajectory in case it was put into a situation where no GPS data was available.

Service robot

Service robots are an emerging market. Just as Japan and other countries have invested heavily in industrial robots in the mid 20^{th} century and that made a huge difference to the industry, there is now a strong investment, specially in Europe, in service robots [39]. Those service robots are expected to be the first of a new generation of intelligent machines. Those intelligent machines should serve humans by performing simple and often boring tasks, such as carry small objects from one point to another, or guide visitors through an exhibition or large building, among others. Surveillance can be considered a particular example of a service. A more demanding job for a service robot is that of assisting people with special needs, such as elderly or disabled people.

There is currently heavy research in the field of service robots. Most of the tools and techniques used have been known for several years. But putting them all together and making the resulting robot work smoothly and efficiently is a complicated problem. Perhaps it is more a technological challenge than it is a scientific challenge, but it is still an open question.

A simple service robot should be able to solve problems such as "carry a book from Laboratory 1 to Laboratory 2". Though very simple, the task requires high level skills. Namely, an efficient robot to perform such a task needs to handle: i) easy communication with the human masters, both to receive the order correctly and to interact with other humans that may be at Laboratory 2 or in the way; ii) ability to plan a proper path to take it from Laboratory 1 to Laboratory 2; iii) ability to carry an object safely; iv) ability to navigate the path safely; v) ability to deliver the object; and finally, vi) ability to get back safely to the starting point, if no other task is assigned meanwhile.

The easiest means of communication with humans, for this kind of tasks, is undoubtly voice. Voice

is one natural and fast means of communication. Therefore, a powerful and easy to use service robot has to include voice control. Speech recognition and synthesis is still an open research question, but there are already available software tools that can be easily integrated into a robot. For example, Sphynx[3] is one of those tools. It is unknown to the author any work using a SDM for voice recognition, but it is possible that the SDM is also appropriate to recognise and reproduce voice commands. However, the integration of already available tools such as Sphynx may be appropriate in order to get faster application results.

As for the abilities to plan and follow paths, that is the main theme of the present work. As shown, the SDM is quite appropriate for those tasks. It can confer on the robot the ability to navigate in a robust way, provided it is first taught the paths. Implementing a wander mode, or an autonomous mapping behaviour, the robot may also be able to create topological maps in an easy way.

The abilities to carry and deliver objects are also out of the scope of this work. But that should not be difficult to accomplish, for simple objects and tasks. The simplest service robot can simply be loaded and unloaded manually, which makes it mostly an object carrier. Using a manipulator it is easy to program basic tasks. Actually, the SDM has already been used to learn and reproduce object manipulation sequences, as described in Appendix A.

Using the SDM as a pattern recognition tool, features such as face recognition can also be implemented. Although it is unknown to the author any work in that field, in theory there is reason to believe the SDM is appropriate for the task, considering its versatility. Extending the model, it is reasonable to assume the SDM-based navigation can be applied in service robots to perform many different tasks. The most simple is just to guide people in large buildings or spaces, such as in museums, shopping centres, factories, libraries, gardens, etc. That can be accomplished adding just voice command to a mobile robot. Perhaps with little extensions such a robot can be made to execute tasks such as help waiting tables at restaurants or bars. Adding the ability to manipulate simple objects, the range of possibilities is almost endless. The beauty of the SDM is that it is a general purpose model of the human memory. It can be used to perform almost any task that can be described using sequences of data vectors.

9.4 Summary

Navigation based on view sequences is still an open research question. This chapter described a novel method that can provide view-based navigation and path planning using a Sparse Distributed Memory. Connection points are established when two paths come together or split. That way, a topological representation of the space is built, which confers on the robot the ability to plan routes and to switch

[3] http://www.speech.cs.cmu.edu (last checked 2010.02.20).

from one path to another when they come together or split apart. Therefore, it is possible to determine which goal points can be reached and plan paths based on the topological representation. The main advantage of using the SDM for that purpose is that it can be used to perform the two most important tasks that are required by the algorithm: storage of sequences of images, and pattern recognition to detect confluence and divergence points.

The algorithm was tested in two different environments: one that is a reconstitution of a countryside environment, the other that is a reconstitution of a changing urban environment. The robot was always able to complete the tasks, even when there were small changes in the environment, such as a car passing by.

The previous chapters showed that navigation based on sequences of images stored in a SDM is feasible and very robust. This chapter extended the approach, by presenting a way to build a topological map and perform path planning using A* or an equivalent planning algorithm. Using this approach, it is possible to program a robot to perform simple tasks, such as surveillance, carrying small objects from one place to another, or guide people in large buildings or spaces.

Chapter 10

Conclusions

An expert is a man who has made all the mistakes which can be made in a very narrow field.
(Niels Bohr, Danish physicist)

Contents

The previous chapters described modern theories of what intelligence is and a system to navigate a robot based on visual images stored into a Sparse Distributed Memory. This chapter briefly summarises all the relevant aspects of the work developed. Then it emphasises the most important contributions to the state of the art and enumerates important advantages and disadvantages of the proposed approach. Finally, it also points out lines of research that could not be fully covered and which must be explored for future work.

10.1 Overview of the work developed

For several decades scientists have been working on building intelligent machines, and the present work is yet another contribution towards that goal. This section summarises the main topics covered in the previous chapters, from the initial motivation to the work developed, and then to the results obtained and projects for future work.

10.1.1 What intelligence is

Chapter 2 presents a brief research on the modern theories of what intelligence is and how the brain achieves intelligent behaviour. Although the concept of "intelligence" is part of common sense and human everyday life, truth is that it is still surrounded by an aura of mystery and there is currently no clear definition of the term. Even experts disagree on the subject. While Gottfredson states that intelligence is the result of "understanding" and "comprehending", Legg defines it in mathematical terms as a sum of rewards in a variety of environments. Gottfredson leaves open the definitions of "understanding" and "comprehending", while Legg's definition leads to a formula which is not computable. Therefore, the definition of intelligence is still an open issue. However, if it is the result of applying a computable algorithm, then it must be possible to build a machine to simulate intelligent behaviour, as proven by Alan Turing.

The source of human intelligence is assumed to be the brain, which is physically made of gray matter and white matter. The gray matter is made of neurons, the white matter is the result of the neural connections between those neurons. The neurons are responsible for both tasks that are considered to be on the basis of intelligence: storage and processing. In the past the emphasis has been put more on processing than storage. However, the brain is a very slow processor. It operates at just 200 Hz. Yet, it solves complicated tasks very swiftly. Even considering parallel processing, it does not have time to process complete solutions to many problems it solves incredibly fast. Hence, the most plausible hypothesis is that the brain solves many tasks by analogy with past situations. It retrieves many of the solutions from the pool of memories, instead of computing them from scratch. Therefore, intelligence may be more the result of using a sophisticated memory than the result of performing highly complicated calculations at incredible speed.

10.1.2 Brain models

As for computational models of the brain, so far there are only some theories. Two remarkable theories are those proposed by Jeff Hawkins and Pentti Kanerva. Hawkins proposes the Memory Prediction

Framework and the Hierarchical Temporal Memory. According to the MPF model, the brain is constantly predicting what will happen next, based on its previously acquired knowledge. When one prediction fails, then the brain rewires (learns), in order to accommodate the unexpected experience. According to the HTM model, the brain is hierarchically organised. The lower layers have more neurons and deal with raw sensorial inputs. The higher layers deal with more refined representations of the world. The highest layer stores highly abstract concepts, which may not even have accurate physical representation, such as "circle" and "square".

Kanerva proposes a model that mimics only the behaviour of the human cerebellum. It is the Sparse Distributed Memory, which is thoroughly described in Chapter 3. The SDM is a model based on the properties of high dimensional boolean spaces. Therefore, it is suitable to store raw binary vectors of sensorial data, such as images from digital cameras. The properties of the SDM are those of the boolean space. To a great extent, they are coincident with the properties of the human memory, in features such as the ability to one-shot learning, forgeting over time, dealing with incomplete and noisy data, working naturally with sequences and establishing associations between partially related concepts.

Both HTM and SDM are very plausible brain models. However, the HTM was just an idea at the time this work was being developed, not a final model ready to implement. On the contrary, Kanerva has worked out the mathematical and practical basis of the SDM since the 1980s, and other authors have also made important contributions to improve the model. Since the goal of this work was to program a robot to perform intelligent navigation using visual memories, it was deemed appropriate to use a Sparse Distributed Memory to try to accomplish the goal.

10.1.3 Towards intelligent robot navigation using view sequences

Navigation is something humans do *naturally*, but for a robot it can be a very challenging task, depending on the complexity of the environment. Many different approaches have been tried, as described in Chapter 4. Some of them include building a map, others, such as navigation based on view sequences, do not. In the present work, since the goal was to navigate a robot based on visual memories, it was deemed appropriate to build a map for path planning and use visual memories for localisation and path following. The approach used for path following is very straightforward. During a learning stage, the robot acquires images and stores them in the SDM. Each image is tagged with a sequence and image number, and also stored with the motion the robot performed just before the image was grabbed. During the autonomous run stages, the robot follows the same paths it has learnt by comparing its current view with the images stored, and retrieving the associated motion commands.

The algorithms were implemented and tested using a small Surveyor robot, as described in Chapter

5. The robot has a front digital camera and a radio communication module. The camera provided the necessary images. The cumunication module made it possible to control the robot in real-time from a laptop computer. For the sake or repeatability, and also due to the characteristics of the small robot, the tests were run in a structured environment, in an arena surrounded by pictures of a countryside landscape. In order to improve the quality of the input to the SDM, the images were equalised and/or contrast stretched.

As soon as the first experiments were performed, some problems became very clear, as described in Chapter 6. First, there was the problem of noise. Even equalised images contain large amounts of noise. In this case, the problem was overcome using the noise as the basis for calculating the access radius to the SDM. The radius is computed and adjusted automatically in function of the noise, considering noise the distance between two consecutive images taken without neither moving the robot nor changing the scenario. It is a fairly simple procedure, and it makes the system robust and adaptable.

Another problem soon devised was that of encoding the information. All the SDM theory is founded on the properties of boolean spaces. However, sensorial information is rarely purely boolean, and the 8-bit PGM images used are made of bytes in which the bits have different weights. Hence, the Hamming distance, proposed by Kanerva as a good measure to compute the similarity between two memory items, might not be the appropriate method.

10.1.4 Studying the encoding problem

The problem of using different methods to encode the information and to process it inside the SDM is extensively studied in Chapter 7. The main problem arises because the Hamming distance considers only the number of ones and zeros in the binary vectors, while the natural binary code also considers the position of each bit. In order to have a better insight into the performance of the SDM using different encoding methods, four different operation modes were implemented: 1) one using the Hamming distance; 2) another one using the Hamming distance, but in which the binary code used was optimised, by sorting some bytes in a more convenient manner; 3) another operation mode using the sum of the differences of the bytes as the similarity measure (arithmetic mode); and 4) another mode in which the images were represented using a sum code, so that the Hamming distance was proportional to the arithmetic distance.

The four operation modes were studied and the results have been compared. In order to have a better insight also into how distribution of the data influenced the performance of the SDM, the experiments were initially conducted without data distribution (i.e., storing just one copy of each image), and later repeated forcing distribution of the data. As expected, the arithmetic and the sum code modes exhibit

better performance, and data distribution improves the robustness of the memory. The number of bits used in the sum code does not seem to have a noticeable effect in the performance of the system for the type and size of vectors used.

The problem of data encoding was also tested in a different domain (described in Appendix A). Navigation using sequences of images is a domain of large vectors and noisy data. On the contrary, manipulation based on the absolute positioning of a robotic arm's joints is a domain of very short vectors and sensorial readings almost free from noise. But even in that domain the use of an arithmetic or sum code modes improves the selectivity of the memory and reduces prediction errors, as described in Appendix A.

10.1.5 Testing the limits of the SDM

Most of the work done before using SDMs was theoretical. Part of the present work consisted in putting the theory into practice and testing the limits of the SDM, as described in Chapter 8. The tests show that navigation based on view sequences stored into a SDM naturally confers on the robot the ability to solve problems such as that of being kidnapped, or navigating backwards. It was tested under different illumination conditions, and the arithmetic and sum code modes proved to enhance the robustness of the memory. Different scenario changes were also experimented, and the robot was able to successfully navigate with occlusions of more than 12% of the image, summing to the usual image noise.

Overflow was another important memory limit that was tested. The memory handled quite well overflows of up to 30% without noticeable performance degradation.

10.1.6 Path planning

Navigation based on visual memories is suitable for short trips. But planning for long trips requires representations of the environment more sophisticated than raw images. Often, those representations are topological maps, in which the paths are arcs and the path intersections are nodes. Chapter 9 describes an algorithm to build a topological map of the environment using a SDM and visual information. The SDM is used simultaneously for two different purposes: to store the sequences of images, and as a pattern matching tool. The nodes are detected when a predetermined number of consecutive images of a path being learnt match images of a previously learnt path. That way, a topological representation of the environment is built. Using an algorithm such as A* it is possible to plan long routes, and the robot can switch from one path to another at node points.

10.2 Main contributions

The present work offers at least three major contributions to the state of art: 1) the study of the encoding problem; 2) tests to the limits of the memory; and 3) a path planning method for navigation based on view sequences.

To the best of the author's knowledge, this is the first time the encoding problem is studied in detail and those solutions are compared. Other authors have already pointed out some weaknesses of the SDM, and proposed alternative methods which point to the encoding problem. For example, the arithmetic mode was indeed first proposed by Ratitch et al. [85]. And Vanhala [101] uses a somehow radical encoding method to improve immunity of the SDM to noisy images. But this is the first time that different methods of encoding the information and operation modes have been implemented and tested to solve the same problem, in order to compare the results and draw scientifically valid conclusions under different circumstances and in different domains.

As for the limits of the memory, many authors have recognised the immunity of the memory to noisy data, its ability to deal with incomplete information, natural forgetting and other characteristics. However, those conclusions were drawn from theoretical evidence only. To the best of the authors' knowledge, this is the first time any tests were performed in order to assess the limits of the memory. It is now possible to affirm with some certainty how much damage the SDM can handle, in itself or in the data processed.

Finally, the third major contribution of the present work is the path planning algorithm based on view sequences stored into the SDM. The versatility of the SDM makes it suitable both to store the sequences of images and to be used as a pattern matching tool to establish node points when paths come along each other.

The proposed approach is *intelligent*, in the sense that it is suitable to have a good performance in a wide variety of environments, and it also adapts its operating parameters to improve the performance in function of the environmental characteristics. The system tolerates large illumination changes, occlusions, being kidnapped and other scenario changes. Besides, the access radius is automatically adjusted in function of the noise, and it infers new paths from the paths that it has been taught. Those are signs of "intelligence," as discussed in Section 2.1.

10.3 Advantages and disadvantages of the proposed approach

The approach proposed in the present work has some advantages over other popular methods. One advantage is the low complexity of the overall system. The more complex module is the SDM itself.

The remainder of the system is very straightforward. Some properties of the SDM become properties of the system, in a way that navigation is very robust to high amounts of noise, illumination and scenario changes. The method proposed is not cognitive in nature, but the SDM can also be used as a tool to build higher level representations of the environment, such as topological maps. Actually, the SDM can be used simultaneously as a storage tool and a pattern recognition tool. That is, in part, also what happens in the human brain, where processing and storage occur at the same time in the same organ. Yet another advantage of the proposed approach is that it is based on the use of sequences. Therefore, it is possible to recall any path whole or in part, since all the relevant details should be stored into the memory, which is another characteristic of the human memory. However, data that has been learnt cannot be forgotten. Of course, it is possible to design an algorithm to selectivity delete special hard locations which may need to be removed. Nonetheless, removal of any hard location may interfere with more than one datum, which may have undesirable or unexpected consequences in some cases. The original model does not foresee a mechanism for selective removal of undesired memories. That characteristic may be an advantage in some cases, when it is desirable that the model holds at least traces of what happened. In other cases, however, it may be an undesirable feature, if some memories need to be deleted, in order to free space for new data or for other reason.

There is one big disadvantage in using a SDM: low storage, compared to traditional memory, specially if distribution of the data is enforced or allowed. Of course, distribution improves the robustness of the model, as shown in Chapter 8, but it has the cost of requiring a large number of additional hard locations. Another disadvantage is the amount of time that may be necessary for normal operation when large amounts of data are stored into the memory. If the memory is implemented using a parallel architecture, that may not be a problem. But for a model simulated in software using linked lists, as is the case of the memory used in these experiments, the time necessary to store or retrieve a datum grows proportionally to the amount of data stored. The problem is minimised using strategies such as the approach proposed in Section 6.1.2—the similarity between two memory items is computed using just part of the vector—, or in Section 8.4.2—the search is done in just part of the addressing space.

As for the path planning algorithm described in Chapter 9, it is very limited, in the sense that it only detects nodes when two paths come across in the same direction. That limitation could only be overcome using metric information, which has been disregarded so far in the model.

10.4 Future work

As shown in Chapters 3 and 9, navigation using view sequences is just one of the possible uses of the SDM. The SDM can be used for many other robot tasks, such as manipulation or language processing.

Future work will start with applying the model developed to a more powerful robotic framework. Lizard is an excellent platform to make experiments in a controlled environment in a small testbed, but it is too small and too slow to make experiments in large buildings or campus. Therefore, the software framework will be adapted to a larger robot, possibly a Scout[1]. The medium-term goal is to implement an intelligent agent such as that described in Section 9.3. The first behaviours to implement are the skills necessary to guide people inside a building. Using a voice interface it will be able to guide visitors through any known building, as well as answer possible questions. A second function is the ability to carry small objects or messages from one department service to another. Those two abilities will make the robot useful for simple practical applications. The vigilant robot, for surveillance tasks, also described in Section 9.3, is more complicated to implement, because it requires more hardware and complex algorithms.

On the technical side, a quick improvement to the current system can be achieved by using the timestamp stored with the images. The timestamp can be used for purposes such as improving disambiguation and localisation abilities, or giving precedence of the more recent memories over the older information. Another important improvement requires the use of ultrasound sensors, which provide very useful distance information. Ultrasound sensors can detect glass doors or walls, and other objects which reflect sound but are invisible or hard to detect using cameras and infrared sensors.

Another detail that may still be subject to further research is that of the minimum image size. A. Torralba [98] argues that every image can be reduced to a small "signature" of just a few hundred bits. Image retrieval in a database can then be performed using just the signature instead of the real image, therefore increasing the speed of processing significantly, and with success rates similar to the performances obtained when using the whole image. Other authors have also used very small images. Matsumoto, whose work is described in Section 4.3.2, used images as small as 32×32, or 1024 bytes. Therefore, it is expectable that the vectors input to the SDM can be greatly reduced without noticeable degradation of the performance. However, with such a small resolution, a lot of details are lost. It would be necessary to test again the limits of the SDM in situations of increased noise, occlusions and other problems.

Another open line of research is that of adding cognitive capabilities to the system. This model is non-cognitive in nature, in the sense that no objects or details of the environment are extracted or learnt from the images. The images are merely used as patterns for robot localisation. Extracting features and doing some interpretation of the contents might greatly increase the potential of the system. A robot with cognitive capabilities will be able to, at least, recognise some objects. That could be achieved in a

[1] X80SV robot, available at http://www.drrobot.com/products_item.asp?itemNumber=X80SV (last checked 2010.10.23).

relatively simple way. First, there has to be a list of features f that are familiar to the system. Then, every image grabbed by the robot has to be scanned for those features, using modern computer vision techniques [76]. Then, an additional SDM can be used to store associations between the images stored in the first SDM and the features detected, in vectors such as $< im_i, f_j, x, y, scale, rotation >$, where f_j is feature j and x and y are the coordinates of f_j in the image im_i. The use of another SDM controlled by the same focus will make it possible to deal with just raw images or cognitive features, as well as both simultaneously. In part, that is also what probably happens with the human brain, since it can operate in more abstract or more concrete terms, depending on the kind of task that needs to be performed, as debated in Chapter 2.

Appendix A

Impact of using different encoding methods in robotic systems

Contents

One of the major contributions of the present work is the study of the problem of encoding data, as described in Chapter 7. The impact of the encoding method was also studied in the domain of robot manipulation, as described in this appendix.

A.1 Introduction

Images grabbed through digital cameras are always very noisy, even if the hardware is top quality, as explained in Section 5.2.2. In other domains, using other sensors, the readings may be more exact. In that case, the encoding method may be less important to the performance of the SDM, because there are less errors and bit mutations between equivalent sensorial readings. That behaviour has been discussed with other researchers, and it was the subject of a joint work with Sascha Jockel, from CINACS, University of Hamburg [48]. Jockel's research was focused on building a system based on an episodic

memory [49]. That episodic memory was intended to store sequences of events (episodes) of a robotic arm, so that the robot was later able to perform the same actions again by following the sequences of positions stored into the SDM. In many aspects, that is equivalent to the problem of navigation based on view sequences. However, the domain of manipulation based on readings from position sensors is much more precise. The readings are almost exact and free from noise. Also, the robotic arm's position is completely described by very short vectors, compared to the long vectors used to store images. Therefore, the behaviours of both Lizard and TASER, using different operation modes, were tested and compared, as described in this appendix. Section A.2 describes the robotic arm hardware and software. Section A.3 describes and compares the results obtained in both domains. Section A.4 discusses the results, and finally Section A.5 briefly summarises the problem, experiments and results.

A.2 TAMS Service Robot

The experiments on manipulation were performed at the division of Technical Aspects of Multimodal Systems (TAMS), of the University of Hamburg. The experimental platform was TAMS SErvice Robot, called TASER.

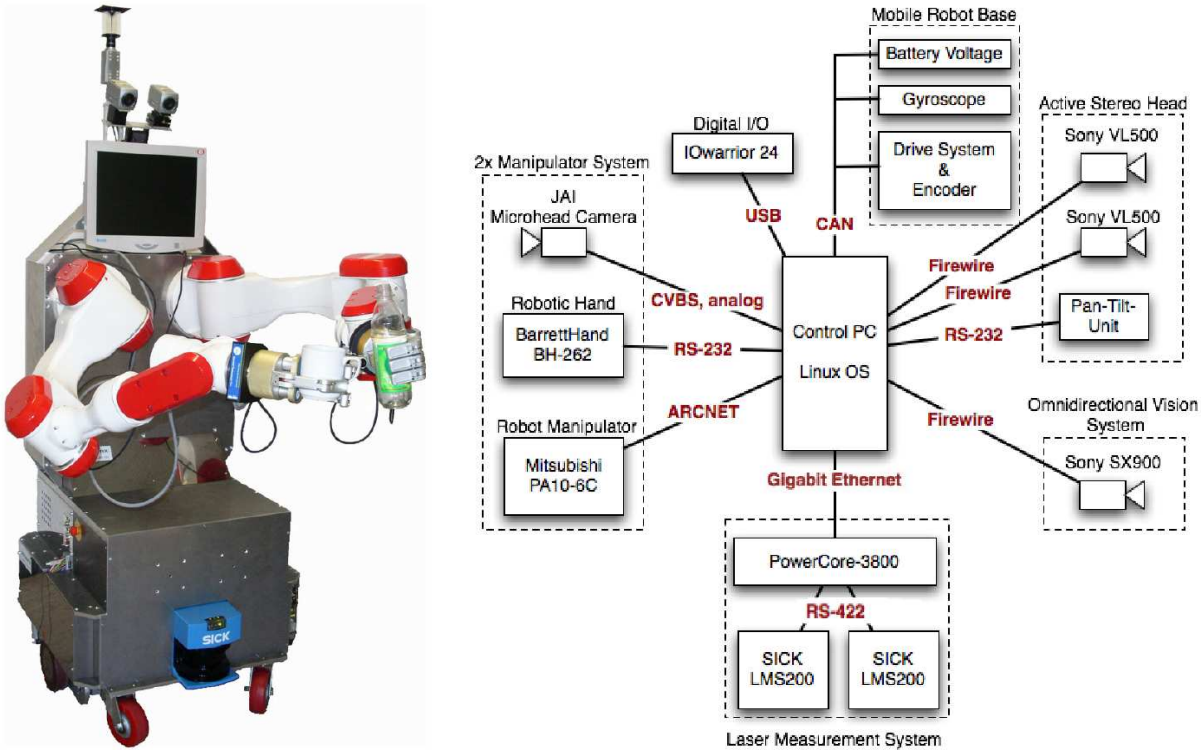
A.2.1 The hardware

Figure A.1(a) shows a picture of TASER, completely assembled. Figure A.1(b) shows a diagram of the architecture of the complete system. In these experiments, only the left arm of the robot was used. The hardware is described in more detail in [47].

TASER is a service robot of approximately human height (about 2 m). The robot system contains a mobile platform with a differential drive and wheel encoders, two laser range finders and a 2.4 GHz industrial computer as central control unit. It also contains a PA10-6C manipulator, two three-finger robotic hands and several cameras for stereo and omnidirectional vision system.

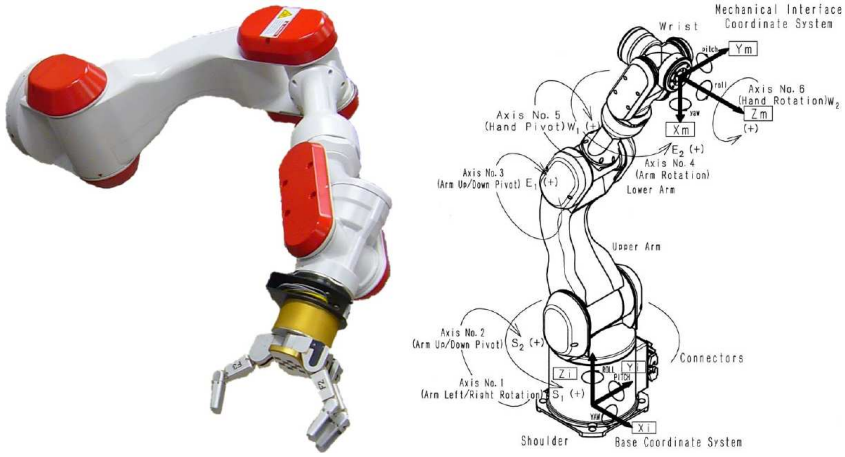
The robotic arm has six degrees of freedom, as shown in Figure A.2(a). Its main purpose is that of fine-positioning the grip, so that the grip can grasp or push objects with precision. The total length of the arm is 1317 mm, and the total weight is 39 Kg. The maximum payload is 10 Kg. The joints are positioned using brushless DC motors, and each joint has a DC resolver for a positioning repeatability of ± 0.1 mm.

The robotic hand is mounted as an end effector onto the robotic arm. It has eight degrees of freedom and is driven by four brushless DC motors. The dimensions are 298 mm $\times$ 149 mm $\times$ 42 mm, and the total weight is 1.2 Kg. The maximum payload is 6 Kg. The maximum payload of the arm together



(a) TASER in its final form. (b) TASER I/O architecture.

Figure A.1: TASER robot picture and architecture.



(a) Arm and gripper. (b) TASER arm's degrees of freedom.

Figure A.2: TASER's robotic arm and gripper.

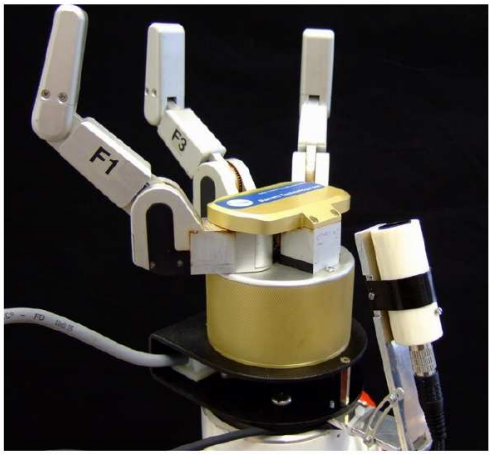


Figure A.3: TASER's hand.

with the grip is approximately 4 Kg. A single motor controls both joints of each finger of the hand, through a gear mechanism that stops one joint when it finds resistance, keeping the other joint going until it also finds resistance. Finger F3 is fixed, but a third motor rotates the other two fingers, F1 and F2, equably, up to 180^o around the palm. The hand also incorporates optical incremental encoders at the base of the fingers that lead to a sensing resolution of up to 0.008^o.

A.2.2 Arm control

The motion of the robotic arm and hand is described by sequences of vectors which include 3-D coordinates, roll, pitch and yaw angles. The inverse kinematics to route from one point to another is computed by the Robot Control C Library (RCCL). During a learning phase, the six joint angles[1] of the robot arm as well as the tool centre point (TCP) and the tool orientation are stored to the SDM with a sampling rate of 6–10 Hz[2]. The robotic arm can be controlled either by joint angles or by TCP and tool orientation. For practical reasons, the latter parameters are used during the learning phase for robot arm control, but the joint angles are also stored into the memory. During an autonomous execution, the robot is supposed to recognise its current joint constellation and follows the learnt trajectory from its memory. The input and output vector consist of arrays of double precision floating point numbers, with the following structure:

$$\vec{x}_i =< j_1, j_2, j_3, j_4, j_5, j_6, x, y, z, \chi, \psi, \omega, seq\_id, i >, \tag{A.1}$$

In A.1, each j_n is the angle of the corresponding arm joint. The 3D coordinates of the tool centre mounted at the end of the robot arm in relation to the robot coordinate system are described by x, y

[1]Each joint has a software limited operation range up to $\pm 255^\circ$.

[2]Due to software difficulties in coordinating the time of sampling in real time, because the operating system is a linux and not a real time operation system, the sampling rate may vary from 6 to 10 Hz.

and z. The letters χ, ψ and ω describe the roll, pitch, and yaw tool orientation. Each of those variables is an 8-byte double precision value. The number *seq_id* is a 4-byte sequence identification number, unique for each sequence. Finally, i is a 4-byte vector identification number, unique for each vector in the sequence *seq_id*. Therefore, the total length of the vector is 104 bytes, or 832 bits. The vectors used for manipulation are, therefore, about 50 times smaller than the vectors used by Lizard for navigation, described in Section 6.1.1.

Just as happens in the case of navigation, addressing the memory is also done using just part of the vector. In this case, the address is a single arm position described by its corresponding 6-joint angles. During an autonomous execution phase, the SDM will predict the j_n from the corresponding j_{n-1}. Hence, the address vector contains just 48 bytes, or 384 bits.

A.3 Experiments and results

Different tests were performed in order to assess the behaviour of the two systems using the approaches described in Chapter 7. Both TASER and Lizard were made to follow sequences of approximately the same duration (40 seconds), and different performance measures were recorded during the execution. The results are summarised in Table A.1. The results presented for navigation were obtained using a sequence of 55 images, which were equalised before being stored into the SDM. The results presented for manipulation were obtained using a sequence of 256 arm positions. Since Lizard operates at about 1.25 Hz and TASER operates at about 6 Hz, in both cases the sequences contain approximately 40 seconds of information.

In Table A.1, the upper half shows results obtained while the memory contained only one copy of each datum (i.e., there is no distribution of the data). The bottom half contains the results of the same tests, but obtained with 5 copies of each datum stored (i.e., distribution of the data is enforced to 5 hard locations). The values shown are: the distance from the input address to the closest hard location (the most similar image); the distance from the input address to the second closest hard location; and the average distance from the input address to all the hard locations in the memory. There is also a measure of the increases, in percentage, which somehow expresses how successful the system is in separating the desired datum from the pool of information in the SDM (columns 6 and 8). The table also shows the number of momentary localisation errors, as defined in Section 6.5.3 (column 9), and the length of the access radius (column 10).

The results for navigation using the sum-code mode were obtained using 16 bits per pixel[3], or 17

[3]Note that the dimensionality of the vector almost doubles. The size of the image doubles, the size of the overhead information remains the same.

Table A.1: Performance of the SDM in two different robotic domains: navigation and manipulation.

Domain	Op. mode	Distr. copies	Dist. to 1^{st}	Dist. to 2^{nd}	Inc_2 (%)	Dist. to average	Inc_{AV} (%)	MLE	Access radius	Vector bits
Manip.	Arit.	1	86.40	103.97	20.33	1797.54	1980.49	0	10	832
	BW		14.17	33.07	133.41	63.52	348.37	7		832
	SC		13.10	82.40	529.01	1274.55	9629.36	0		22384
Nav.	Arit.	1	18282	118892	550.32	166406.53	810.22	13.2	Dyn.	41064
	BW		6653	9186	38.07	9724.80	46.17	15.3		41064
	SC		569	3791	566.26	5257.01	823.90	13.8		82024
Manip.	Arit.	5	12.35	14.4	17.32	1275.30	10227.33	0	15	832
	BW		2.87	6.03	110.47	62.83	2091.90	0		832
	SC		12.67	12.67	0.0	1271.31	9936.67	0		22384
Nav.	Arit.	5	33349	33503	0.46	145765.32	337.09	15.3	Dyn.	41064
	BW		7167	7298	1.83	9359.14	30.59	16.9		41064
	SC		2214	2346	5.97	9184.19	314.84	15.2		82024

gray levels. In each case, the access radius was computed dynamically (indicated "Dyn." in the table), as described in Section 6.2. In the case of manipulation, the access radius was experimentally chosen and set by the users. In the operation mode using the sum code, each angle was described by 255 bits[4], which lead to a total of 22384 bits in the input vector. For better accuracy, the results shown in the table are the average of 30 predictions.

A.4 Discussion

Table A.1 clearly illustrates the different natures of the problems. In the case of manipulation, the data vectors are much shorter—4 times shorter in the sum code mode, and 50 times shorter in the other operation modes. However, in both cases the vectors are within the range 2^n, for $100 < n < 10^5$, as explained in Section 3.1. In the case of manipulation, the access radius is fixed. In the case of navigation, a dynamic access radius is used. However, the experiments were done this way strictly for practical reasons, and that decision should have no impact on the results or conclusions. The same algorithm proposed in Section 6.2 to compute the radius in function of the noise levels could be applied in the case of manipulation.

The momentary localisation errors were computed using the same algorithm in both cases (the algorithm is described in Section 6.5.3). The results obtained for navigation are coherent with the results obtained in other experiments, as presented in previous chapters. In manipulation, there are much less localisation errors, despite the fact that the sequence has much more elements and the sample

[4]Note that 255 bits to describe an angle may not be enough if high precision is needed. But for these experiments, 255 is enough to get reliable results.

rate is much higher. That is an expected result, since the amount of noise in the domain of manipulation is almost inexistent. However, without distribution of the data the bitwise operation mode presented a total of seven momentary localisation errors. With distribution, those errors disappear. The result is also according to the previous results obtained in Chapter 8, and stresses the weakness of the bitwise mode and the advantages of data distribution for robust operation.

The results show the feasibility of the approach: the SDM can be used in robotics, to guide robots in various tasks using a learn-and-follow approach. One interesting characteristic of the method, in the domain of manipulation, is that regardless of the initial position, the robot arm is always able to converge to the closest point in the sequence, thus always executing the assigned manipulating task with success. That happens in manipulation, because the absolute positioning of the joints is used. In navigation, that is not possible, because the robot has no information of absolute positioning. In navigation, the robot has to start at a point where it sees at least part of a recognised image. Using global positioning coordinates (GPS), the behaviour could be the same. However, while the robotic arm is expected to move in an empty or free environment, navigation is more complicated, because the robot is expected to encounter and deal with many obstacles.

Another advantage of using the SDM in the domain of manipulation is that it is straightforward to associate sounds or even images with some manipulation sequences. For example, the sound of a bell ringing could trigger a door opening sequence. Or the image of a bottle could trigger the sequence to grab and open it. In the domain of navigation similar behaviours may be more difficult to implement, due to the absence of an absolute positioning system and the inherent difficulty of converging to specific sequences. In manipulation, learning those behaviours should be fast and efficient.

A.5 Summary

This chapter describes two experiments in which the SDM was used to solve different robot problems. The goal was to assess the performance of the SDM in two different domains, and using different encoding methods.

In the case of robot navigation based on visual information, data has large amounts of noise, and the vectors are of very high dimensionality. In the case of robot manipulation using absolute positioning, the problem is of very low dimensionality and the data has only small amounts of noise. Both problems were solved using the same approach: store sequences of sensorial information into the SDM during a learning stage, and follow those sequences later during the autonomous runs. In both cases, the systems worked perfectly. The SDM showed to be suitable for the jobs. The results obtained in both domains confirm the results obtained for navigation alone in previous experiments: the operation mode and the

distribution of the data may affect the performance of the system in both cases. The SDM seems to be more robust in the arithmetic mode, or in the sum code mode. Also, data distribution, in general, makes the system more robust.

Acronyms and symbols

Abbreviation	Meaning
AGV	Automatically Guided Vehicle
AI	Artificial Intelligence
AM	Address module (Stanford model)
AR	Arithmetic mode
AU	Action Unit (facial movement responsible for part of a facial expression)
BA	Basic navigation Algorithm
BBAI	Behaviour Based Artificial Intelligence
BBR	Behaviour Based Robotics
B.C.	Before Christ
BC	Binary Code
BW	Bitwise mode
CCTV	Closed-circuit television
CM	Control Module (Stanford model)
CS	Contrast Stretching
DL	Dim Light
Eq	Equalisation
EM	Executive Module (Stanford model)
fps	Frames per Second
GA	Genetic Algorithm
GM	Genetic Memory
HD	Hamming Distance
HMM	Hidden Markov Model
HN	Hopfield Network
HTM	Hierarchical Temporal Memory
IQ	Intelligence Quotient
IR	Infrared
ISM	Industrial, Scientific, Medical [radio frequency band]
KNN	k-nearest neighbour
MLE	Momentaneous Localisation Error
N	The space $\{0,1\}^n$, also referred to as 2^n
N'	The subset of hard locations
n	Dimensions of 2^n
NBC	Natural Binary Code
NN	Neural Network

Abbreviation	Meaning
OF	Optical Flow
PCA	Principal Component Analysis
PF	Potential Field
RCCL	Robot Control C Library
RL	Reinforcement Learning
RRA	Randomised Reallocation Algorithm
SA	Simple navigation Algorithm
SCSI	Small Computer Systems Interface
SDM	Sparse Distributed Memory
SM	Sequence Machine
SM	Stack module (Stanford model)
SL	Strong Light
SOA	State Of the Art
SSDA	Sequencial Similarity Detection Algorithm
SVM	Support Vector Machine
TAMS	Technical Aspects of Multimodal Systems (Univ. of Hamburg)
TASER	TAMS Service Robot
TCL	Tool Center Point
TM	Turing Machine
UTM	Universal Turing Machine
VD	Voronoi Diagram
WN	Willshaw Network
ZNCC	Zero mean Normalised Cross Correlation

List of Figures

List of Tables

References

[1] James Albus. *Brains, Behaviour, and Robotics*. Byte Books of McGraw Hill, 1981. 3.1

[2] Paul Almond. A proposal for general ai modeling. http://www.paul-almond.com/Modeling.pdf (last checked 2009.11.26), April 2009. 2.2.6

[3] Bernard Ans, Stéphane Rousset, Robert M. French, and Serban Musca. Self-refreshing memory in artificial neural networks: learning temporal sequences without catastrophic forgetting. *Connection Science*, 16(2), June 2004. 3.6.8

[4] Ashraf Anwar, Dipankar Dasgupta, and Stan Franklin. Using genetic algorithms for sparse distributed memory initialization. In *International Conference Genetic and Evolutionary Computation (GECCO)*, July 1999. 3.6

[5] Ashraf Anwar and Stan Franklin. Sparse distributed memory as a tool for conscious software agents. http://www.msci.memphis.edu/~cmattie/. 3.6

[6] Nikolai Axmacher, Michael X. Cohen, Juergen Fell, Sven Haupt, Matthias Dümpelmann, Christian E. Elger, Thomas E. Schlaepfer, Doris Lenartz, Volker Sturm, and Charan Ranganath. Intracranial eeg correlates of expectancy and memory formation in the human hippocampus and nucleus accumbens. *Neuron*, 65(4):541–549, February 2010. 2.2.4

[7] Sven Behnke and Albert Ludwigs. See, walk and kick: Humanoid robots start to play soccer. *International Journal of Computer Science in Sport*, 5(1), 2006. 1.1.2

[8] Leonard M. Blumenthal and Karl Menger. *Studies in Geometry*. Freeman, San Francisco, 1970. 3.2

[9] Joy Bose. A scalable sparse distributed neural memory model. Master's thesis, University of Manchester, Faculty of Science and Engineering, Manchester, UK, 2003. 3.6.7, 6, 3.6.8, 8.4.2

[10] Joy Bose. A neural network for recognition and prediction of temporal sequences of patterns. In *Proceedings of the PREP 2005*, Lancaster, UK, April 2005. 3.6.8

[11] Joy Bose, Steve B. Furber, and Jonathan L. Shapiro. A spiking neural sparse distributed memory implementation for learning and predicting temporal sequences. In *International Conference on Artificial Neural Networks (ICANN)*, Warsaw, Poland, September 2005. 3.6.8

[12] Rodney A. Brooks. A robust layered control system for a mobile robot. *IEEE Journal of Robotics and Automation*, 2(1), March 1986. 4.1.1

[13] Bruce G. Buchanan. Timeline: A brief history of artificial intelligence. http://www.aaai.org/AITopics/pmwiki/pmwiki.php/AITopics/BriefHistory, May 2010. 1, 1.1

[14] Neil Burgess and Graham Hitch. Computational models of working memory: putting long-term memory into context. *Trends in cognitive sciences*, 9(11), 2005. 3.6.8

[15] Karel Capek. *R.u.r (Rossum's Universal Robots)*. http://www.gutenberg.org/etext/13083, 1921. 1.1.2

[16] Ronald Chrisley and Sander Begeer. *Artificial Intelligence: critical concepts*. Routledge, December 2000. 1.1.2

[17] Eric Chudler, Ellen Kuwana, Melissa Philips, and Marge Murray. Neuroscience for kids. http://faculty.washington.edu/chudler/neurok.html (as of March 03, 2007), 2007. 2.2.2

[18] W.B.Toms D.A.Edwards, W.J.Bainbridge and S.B.Furber. Delay in sensitive, point-to-point interconnect using m-of-n codes. In *Proceedings of Async 2003*, May 2003. 6

[19] Fan Dainian and Robert S. Cohen, editors. *Chinese studies in the history of philosophy of science and technology*. Kluwer Academic Publishers, 1996. 1.1.1

[20] Peter J. Denning. Sparse distributed memory. *American Scientist*, (77), July-August 1989. 3.1, 3.5

[21] Edsger W. Dijskstra. A note on two problems in connexion with graphs. *Numerische Mathematik*, 1(1):269–271, 1959. 4.2.4, 4.3.2

[22] Dimiter Dobrev. Formal definition of artificial intelligence. *Information Theories and Applications*, 12(3):277–285, 2005. 2.1.2

[23] Albert Elfes. Using occupancy grids for mobile robot perception and navigation. *Computer*, 22(6):46–57, 1989. 4.2.1

[24] Robert Fisher, Simon Perkins, Ashley Walker, and Erik Wolfart. Hypermedia image processing reference. http://homepages.inf.ed.ac.uk/rbf/HIPR2/, 2003. 5.2.2

[25] M. J. Flynn, Pentti Kanerva, and N. Bhadkamkar. Sparse distributed memory: Principles and operation. Technical Report CSL-TR-89-400, Computer Systems Laboratory, Departments of Electrical Engineering and Computer Science, Stanford University, Stanford, California 94305-4055, 1989. 3.6.1

[26] Otto Foster. *Algorithmische Zahlentheorie*. Vieweg-Verlag, 1996. 1

[27] Matthias O. Franz, Bernhard Schölkopf, Hanspeter A. Mallot, and Heinrich H. Bülthoff. Learning view graphs for robot navigation. *Autonomous Robots*, 5(1):111–125, March 1998. 4.3.6

[28] Rolf Klein Franz Aurenhammer. Voronoi diagrams. Technical Report 198, FernUniversitat Hagen, Department of Computer Science, Germany, 1996. 4.2.3

[29] Stephen B. Furber, John Bainbridge, J. Mike Cumpstey, and Steve Temple. Sparse distributed memory using n-of-m codes. *Neural Networks*, 17(10):1437–1451, 2004. 3.6.7, 6, 3.7, 6.1.2

[30] Dileep George and Jeff Hawkins. A hierarchical bayesian model of invariant pattern recognition in the visual cortex. In *Proceedings of the International Joint Conference on Neural Networks*, 2005. 2.2.5, 2.4, 2.5, 2.2.6

[31] Malcom Gladwell. *Blink*. Back Bay Books, Hachette Book Group, New York, USA, 2007. 1.2.1, 2.2.6, 3.6.8

[32] Linda S. Gottfredson. Mainstream science on intelligence: An editorial with 52 signatories, history and bibliography. *Intelligence*, 24(1):13–23, 1997. 2.1.2, 3

[33] Frank Gray. Pulse code communication. U.S. Patent 2,632,058, March 1953. 7.1.2

[34] Stephanie Haack. A brief history of artificial intelligence. http://www.atariarchives.org/deli/artificial_intelligence.php. 1.1.1

[35] Robert M. Harnish. *Minds, brains, computers: an historical introduction to the foundations of cognitive science*. Wiley-Blackwell, 2002. 2.2.3

[36] Peter Hart, Nils Nilsson, and Bertram Raphael. A formal basis for the heuristic determination of minimum cost paths. *IEEE Transactions on Systems Science and Cybernetics*, 4(2), July 1968. 4.2.4

[37] Jeff Hawkins and Sandra Blakeslee. *On Intelligence*. Times Books, New York, 2004. 1.3, 2.2.2, 2.2.3, 2.2.4

[38] Tim A. Hely, David J. Willshaw, and Gillian M. Hayes. A new approach to kanerva's sparse distributed memories. *IEEE Transactions on Neural Networks*, pages 101–105, 1999. 3.6, 3.10, 3.6.5

[39] Eva Henkel. International robot exhibition 2009: Germany leads europe in service robotics segment. *Germany Trade & Invest*, November 2009. 9.3

[40] Richard N. A. Henson and David J. Willshaw. Short-term associative memory. In *Proceedings of the INNS World Congress on Neural Networks*, Washington DC, 1995. 2.3.2

[41] Andrew Hodges. *Alan Turing: the enigma*. Vintage, Random House, London, UK, 1992. 1.1.1

[42] J. J. Hopfield. Neural networks and physical systems with emergent collective computational abilities. In *Proceedings of the National Academy of Science*, volume 79, pages 2554–2558, 1982. 2.3.3

[43] Amelia R. Hunt and Patrick Cavanagh. Looking ahead: The perceived direction of gaze shifts before the eyes move. *Journal of Vision*, 9(9):1–7, 2009. 2.2.6

[44] Hiroshi Ishiguro, Takahiro Miyashita, and Saburo Tsuji. T-net for navigating a vision-guided robot in real world. In *ICRA*, pages 1068–1074, 1995. 4.3.4

[45] Hiroshi Ishiguro and Saburo Tsuji. Image-based memory of environment. In *in Proc. IEEE/RSJ Int. Conf. Intelligent Robots and Systems*, pages 634–639, 1996. 4.3.3

[46] Louis A. Jaeckel. An alternative design for a sparse distributed memory. Technical report, Research Institute for Advanced Computer Science, NASA Ames Research Center, July 1989. 3.6.4, 3.7

[47] Sascha Jockel. *Cross-Modal Learning and Prediction of Autobiographical Episodic Experiences using Sparse Distributed Memory*. PhD thesis, Department Informatik, MIN-Fakultät, Universität Hamburg, Hamburg, Germany, 2010. 1.3, 1, A.2.1

[48] Sascha Jockel, Mateus Mendes, Jianwei Zhang, A. Paulo Coimbra, and Manuel M. Crisóstomo. Robot navigation and manipulation based on a predictive associative memory. In *Proceedings of the 2009 IEEE 8th International Conference on Development and Learning (ICDL)*, Shanghai, China, June 2009. A.1

[49] Sascha Jockel, Martin Weser, Daniel Westhoff, and Jianwei Zhang. Towards an episodic memory for cognitive robots. In *Proceedings of 6^{th} Cognitive Robotics workshop at 18^{th} European Conference on Artificial Intelligence (ECAI)*, pages 68–74, Patras, Greece, July 2008. IOS Press. A.1

[50] Steven Johnson. *Mind wide open*. Scribner, New York, 2004. 1.2.1, 1.3

[51] Stephen D. Jones, Claus Andresen, and James L. Crawley. Appearance based processes for visual navigation. In *Proceedings of the IEEE International Conference on Intelligent Robots and Systems*, Grenoble, France, September 1997. 4.3.6

[52] Pentti Kanerva. *Sparse Distributed Memory*. MIT Press, Cambridge, 1988. 1.3, 3.1, 3.2, 3.6

[53] Roland Karlsson. A fast activation mechanism for the kanerva sdm memory. In *Proceedings of the 95 RWC Symposium*, pages 69–70, Tokyo, June 1995. 3.6.4

[54] James D. Keeler. Comparison between kanerva's sdm and hopfield-type neural networks. *Cognitive Science*, 12(3):299–329, 1988. 3.6, 3.6.3

[55] Oussama Khatib. Real-time obstacle avoidance for manipulators and mobile robots. In *Proceedings of the IEEE International Conference on Robotics and Automation*, volume 2, Stanford University, Stanford, California, USA, March 1985. 4.2.2

[56] Sven Koenig and Reid G. Simmons. Xavier: A robot navigation architecture based on partially observable markov decision process models. In *Artificial Intelligence Based Mobile Robotics: Case Studies of Successful Robot Systems*, pages 91–122. MIT Press, 1998. 4.4.3

[57] Y. Koren and Johann Borenstein. Potential field methods and their inherent limitations for mobile robot navigation. In *Proceedings of the IEEE International Conference on Robotics and Automation*, pages 1398–1404, Sacramento, California, April 1991. 4.2.2

[58] Benjamin J. Kuipers and Tod S. Levitt. Navigation and mapping in large-scale space. *International Journal of Artificial Intelligence*, 9(2):25–43, 1988. 4.2.4, 4.2.4, 4.2.4

[59] Shane Legg and Marcus Hutter. Universal intelligence: A definition of machine intelligence. *Minds and Machines*, 17(4):391–444, November 2007. http://www.vetta.org/about-me/publications/. 2.1.2

[60] Jianhe Lei, Quanjun Song, Jinghua Ma, Liankui Qiu, and Yunjian Ge. Application of svm in intelligent robot information acquisition and processing: A survey. In *Proceedings of the 2005 IEEE International Conference on Information Acquisition*, Hong-Kong and Macau, 2005. 4.4.3

[61] Mao-Hai Li, Bing-Rong Honga, Ze-Su Caia, Song-Hao Piaoa, and Qing-Cheng Huanga. Novel indoor mobile robot navigation using monocular vision. *Engineering Applications of Artificial Intelligence*, 21(3), April 2007. 4.3.6

[62] Hanspeter A. Mallot, Heinrich H. Bülthoff, Philipp Georg, Bernhard Schölkpf, and Ken Yasuhara. View-based cognitive map learning by an autonomous robot. *Adaptive Behavior*, 3(3):311–348, 1995. 4.3.5

[63] David Marr. Simple memory: A theory for archicortex. *Philosophical Transactions of the Royal Society of London*, (262), 1971. 3.1

[64] Yoshio Matsumoto, Kazunori Ikeda, Masayuki Inaba, and Hirochika Inoue. Exploration and map acquisition for view-based navigation in corridor environment. In *Proceedings of the International Conference on Field and Service Robotics*, pages 341–346, 1999. 4.3.2

[65] Yoshio Matsumoto, Masayuki Inaba, and Hirochika Inoue. View-based approach to robot navigation. In *Proceedings of 2000 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS 2000)*, 2000. 4.3.2

[66] Yoshio Matsumoto, Masayuki Inaba, and Hirochika Inoue. View-based navigation using an omniview sequence in a corridor environment. In *Machine Vision and Applications*, 2003. 4.3.2

[67] James Matthews. Histogram equalization. *Generation 5*, November 2004. 5.2.2

[68] John McCarthy, Marvin Minsky, Nathaniel Rochester, and Claude E. Shannon. A proposal for the dartmouth summer research project on artificial intelligence. http://www-formal.stanford.edu/jmc/history/dartmouth/dartmouth.html, August 1955. 1.1, 1.1.2

[69] Mateus Mendes, A. Paulo Coimbra, and Manuel Crisóstomo. AI and memory: Studies towards equipping a robot with a sparse distributed memory. In *Proceedings of the IEEE International Conference on Robotics and Biomimetics*, pages 1743–1750, Sanya, China, December 2007. 3.7, 6

[70] Mateus Mendes, A. Paulo Coimbra, and Manuel Crisóstomo. Path planning for robot navigation using view sequences. In *Lecture Notes in Engineering and Computer Science: Proceedings of The World Congress on Engineering 2010, WCE 2010*, London, UK, 30 June–2 July 2010. 9

[71] Mateus Mendes, Manuel Crisóstomo, and A. Paulo Coimbra. Robot navigation using a sparse distributed memory. In *Proceedings of the 2008 IEEE International Conference on Robotics and Automation*, Pasadena, California, USA, May 2008. 3.7, 6

[72] Mateus Mendes, Manuel Crisóstomo, and A. Paulo Coimbra. Assessing a sparse distributed memory using different encoding methods. In *Proceedings of the 2009 World Congress on Engineering*, pages 37–42, London, UK, July 2009. 7

[73] Mateus Mendes, Manuel Crisóstomo, and A. Paulo Coimbra. *Electronic Engineering and Computing Technology*, chapter Encoding Data to use with a Sparse Distributed Memory. Springer, Netherlands, April 2010. 7

[74] Vernon Mountcastle. An organizing principle for cerebral function: the unit model and the distributed system. In Gerald M. Edelman and Vernon Mountcastle, editors, *The mindful brain*. MIT Press, Cambridge, Mass., 1978. 2.2.2

[75] Gabriel Avi na Cervantes, Michel Devy, and Antonio Marín-Hernández. Lane extraction and tracking for robot navigation in agricultural applications. In *Proceedings of the 11th International Conference on Advanced Robotics*, Coimbra, Portugal, 2003. 4.4.3

[76] Vishvjit S. Nalwa. *A guided tour of computer vision*. Addison-Wesley Pub, February 1993. 10.4

[77] Randal C. Nelson. Visual homing using an associative memory. *Biological Cybernetics*, 65(4):281–291, August 1991. 4.3.7

[78] Nils J. Nilsson. *Artificial Intelligence: A new synthesis*. Morgan Kaufmann Publishers Inc., San Francisco, USA, 1998. 1, 1.1

[79] Gerard O'Reagan. *A Brief History of Computing*. Springer, 2008. 1.1.1

[80] Marlene Oscar-Berman and Ksenija Marinkovic. *Alcoholism and the Brain: An Overview*. National Institute on Alcohol Abuse and Alcoholism (NIAAA), Bethesda, USA, July 2004. 2.3

[81] Rajesh P. N. Rao and Dana H. Ballard. Object indexing using an iconic sparse distributed memory. Technical Report 559, The University of Rochester, Computer Science Department, Rochester, New York, July 1995. 3.6.3, 3.7

[82] Rajesh P.N. Rao and Olac Fuentes. Hierarchical learning of navigational behaviors in an autonomous robot using a predictive sparse distributed memory. *Machine Learning*, 31(1-3):87–113, April 1998. 4.4.1, 4.12

[83] Christopher Rasmussen and Gregory D. Hager. Robot navigation using image sequences. In *In Proc. AAAI*, pages 938–943, 1996. 4.3.1

[84] Bohdana Ratitch, Swaminathan Mahadevan, and Doina Precup. Sparse distributed memories in reinforcement learning: Case studies. In *Proceedings of the Workshop on Learning and Planning in Markov Processes - Advances and Challenges*. AAAI, 2004. 3.6.6

[85] Bohdana Ratitch and Doina Precup. Sparse distributed memories for on-line value-based reinforcement learning. In *ECML*, 2004. 3.6.6, 3.7, 10.2

[86] Evan Ratliff. The thinking machine. *Wired*, March 15 2007. 9

[87] David Rogers. Predicting weather using a genetic memory: A combination of kanerva's sparse distributed memory with holland's genetic algorithms. In *NIPS*, 1989. 3.1, 3.6.2

[88] Allen Rubin. *Statistics for Evidence-Based Practice and Evaluation*. Cengage Learning, March 2009. 3.1

[89] Stuart J. Russell and Peter Norvig. *Artificial Intelligence: A modern approach*. Prentice Hall, New Jersey, USA, 1995. 1, 1.1, 1.1.2, 1.2.3

[90] S. Ryan and J. Andreae. Improving the performance of kanerva's associative memory. *IEEE Transactions on Neural Networks*, 6(1):125–130, 1995. 3.3.3

[91] Joel Schmidt. *Larousse Greek and Roman Mythology*. Mcgraw Hill, 1983. 1.1.1

[92] Friedhelm Schwenker, Hans A. Kestler, Steffen Simon, and Günter Palm. 3d object recognition for autonomous mobile robots utilizing support vector classifiers. In *Proceedings of the 2001 IEEE International Symposium on Computational Intelligence in Robotics and Automation*, Alberta, Canada, 2001. 4.4.3

[93] John Searle. Minds, brains, and programs. *Behavioral and Brain Sciences*, (3):417–424, 1980. 2.1.2

[94] Hagit Shatkay and Leslie Pack Kaelbling. Learning geometrically-constrained hidden markov models for robot navigation: Bridging the topological-geometrical gap. *Journal of AI Research*, 16:16–167, 2002. 4.4.3

[95] David Smith. *History of mathematics*, volume 2. Dover Publications, 1958. 1.1.1

[96] C. E. Spearman. *The abilities of man, their nature and measurement*. Macmillan, New York, 1927. 2.1.2

[97] Sebastian Thrun and Arno Bucken. Integrating grid-based and topological maps for mobile robot navigation. In *Proceedings of the Thirteenth National Conference on Artificial Intelligence*, Portland, Oregon, USA, August 1996. AAAI. 3, 4.2.4

[98] Antonio Torralba, Rob Fergus, and Yair Weiss. Small codes and large image databases for recognition. *IEEE Computer Vision and Pattern Recognition*, June 2008. 10.4

[99] Edmondo Trentin and Roldano Cattoni. Learning perception for indoor robot navigation with a hybrid hidden markov model/recurrent neural networks approach. *Connection Science*, 11(3), December 1999. 4.4.3

[100] Alan M. Turing. On computable numbers, with an application to the engscheidungsproblem. *Proceedings of the London Mathematical Society*, 2(42):230–265, 1936. 2.1.1

[101] Jukka Vanhala, Jukka Saarinen, and Kimmo Kaski. Sparse distributed memory for multivalued patterns. In *IEEE International Conference on Neural Networks*, 1993. 3.6.9, 6.2, 7.1.1, 7.1.4, 10.2

[102] Jan Vascak. Navigation of mobile robots using potential fields and computational intelligence means. *Acta Polytechnica Hungarica*, 4(1), 2007. 4.2.2

[103] Dizan Vasquez. Growing hidden markov models: An incremental tool for learning and predicting human and vehicle motion. *The International Journal of Robotics Research*, 28(11–12), 2009. 5

[104] G. M. Voronoi. Nouvelles applications des parametres continus a la theorie des formes quadratiques. *Math*, 1908. 4.2.3

[105] Michiko Watanabe, Masashi Furukawa, and Yukinori Kakazu. Intelligent agv driving toward an autonomous decentralized manufacturing system. *Robotics and computer-integrated manufacturing*, 17(1-2):57–64, February-April 2001. 4.4.2

[106] Joseph Weizenbaum. Eliza – a computer program for the study of natural language communication between man and machine. *Communications of the ACM*, 9(1):36–45, January 1966. 1.2.2

[107] David J. Willshaw, O. P. Buneman, and H. C. Longuet-Higgins. Non-holographic associative memory. *Nature*, 222(5197):960–962, June 1969. 2.3.2

[108] Niall Winters and José Santos-Victor. Mobile robot navigation using omni-directional vision. In *In Proc. 3rd Irish Machine Vision and Image Processing Conference (IMVIP'99)*, pages 151–166, 1999. 4.3.6

[109] Kai M. Wurm, Rainer Kuemmerle, Cyrill Stachniss, and Wolfram Burgard. Improving robot navigation in structured outdoor environments by identifying vegetation from laser data. In *In Proc. of the IEEE/RSJ International Conference on Intelligent Robots and Systems*, 2009. 4.4.3