AIBO-R:
Integrating Cognitive Models and Robotics

Anton Wijbenga

Date:	June 25, 2007
e-mail:	antonw@ai.rug.nl
Student nr.:	1211005
Supervised by:	dr. D.H. van Rijn
	dr. B. de Boer

Artificial Intelligence
University of Groningen

Abstract

Over the years, the field of robotics has primarily focused on low-level behaviour, like navigation and exploring, while cognitive modelling has primarily focused on high-level cognition. As a result, each field often disregards the other field's level of cognition. Combining a cognitive model with a robot, such that multiple levels of cognition are implemented, might be beneficial for both robotics and cognitive modelling. To explore the possible benefits of that combination, an embodied cognitive model of route learning was developed. The model was developed using ACT-R, which has been expanded to interface with a Sony AIBO robot.

The study shows that creating an interface between a cognitive model and a robot is challenging, especially when low- and high-level cognition have to interact in a plausible and useful way. From the cognitive modelling perspective, it forces one to include several aspects which in conventional models might have been overlooked. At the same time, the study showed that robotics could benefit from the unified representations and learning mechanisms of a cognitive architecture, which result in useful top-down control. As a result, a robot is able to learn in a cognitively plausible way and is able to improve its performance by adapting to a new environment.

TABLE OF CONTENTS

1. Introduction

The field of artificial intelligence is very broad, but the different research areas within artificial intelligence all share a common goal: to explain and apply intelligence. Each research area uses its own approach to work towards that goal, but unfortunately different research areas often do not combine their knowledge.

An example of two areas that can learn a lot from each other is cognitive modelling and robotics. Whereas cognitive modelling attempts to model intelligence at a relatively high level, often disregarding low level processes, robotics attempts to gain insight into intelligence at a rather low level usually disregarding high level influences. Since a complete theory of intelligence should include both the low and the high level processes and the nature of the interactions between them, it is a good idea to combine cognitive modelling and robotics. By combining cognitive modelling and robotics, models can be created that approach a complete theory of intelligence closer than most existing models. The primary goal of the current research is to explore what insights can be gained by combining cognitive modelling and robotics.

The remainder of this chapter will discuss the possible benefits of using robotics in the area of cognitive modelling and vice versa, a task domain to explore the combination of the two research areas and finally the research sub-goals.

1.1 Combining Cognitive Modelling and Robotics

There are several arguments for using robotics in the area of cognitive modelling, which shall be referred to as embodied cognitive modelling, besides the argument mentioned above. One of these arguments is that a simulated world does not provide all aspects that are relevant for a task. For example, when driving a car, the engine sound provides an indication on when to change gear and horizontal g-forces might indicate that one should take back some gas when taking a corner. While it is possible to simulate these aspects as well, it is almost impossible to think of every aspect that plays a role when performing a certain task. Even if one succeeds in identifying each relevant aspect, it quickly becomes technically impossible to simulate these aspects.

As will be discussed in the next chapter, actively travelling a route plays an important role when learning the route. This aspect is hard to model and to model the effort accompanying active travel one would need a detailed model of the environment including laws of physics. Besides that, one would have to simulate noise that is present when sensing the real world through sensors and deviations that arise when actions are preformed through actuators. Although modelling all these aspects is not impossible, it is challenging and eventually the model should work in the real world anyway. Embodied cognitive modelling therefore is an elegant solution to the problems just mentioned as in the real world these aspects are inherently present.

Another argument for embodied cognitive modelling can be derived from the work of Marr (1982). Marr defined three levels that are necessary to understand an information processing system. Since cognitive models are also information processing systems the three levels apply to those as well. Conventional cognitive models account for the first two levels, which are the *computational* and the *algorithmic* level. These levels respectively represent the goal of a system and the means by which the goal can be obtained. However, the third level, referred to as the *implementation* level, is usually not considered when evaluating the plausibility of a cognitive model. This third level is the medium which realizes the means defined by the second level.

In the area of cognitive modelling this medium is usually a normal PC, but a PC is physically completely different from humans. Of course this is also true for robots, but at least they are a step closer toward mimicking humans. This is important, because the capabilities (i.e., which goals can be reached) of an information processing system are first defined by the third level, which is effectively the hardware, and then by the second level, which is the software. Therefore the goals, represented by the first level, depend heavily on both the second and third level. By choosing a robot as the means to process the algorithms of a cognitive model, the model is one step closer towards humans.

Also, using a robot to implement a cognitive model, forces one to consider all the limitations that come along with the robot. This means that solutions to problems that might have been ignored have to be implemented in the model, possibly resulting in a more plausible model.

So far only benefits for using robots in the area of cognitive modelling have been discussed, but robotics can profit from the area of cognitive modelling as well. Robotics has already recognized the area of biology (i.e., animals) as a source of inspiration for certain algorithms (Franz & Mallot, 2000). However, usually these algorithms simulate relatively basic tasks. To create algorithms that simulate more complex tasks, higher cognitive processes are needed. For those tasks it makes sense to use inspiration from the intelligence of humans and thus cognitive modelling.

Examples of tasks that can be used as inspiration for more complex tasks are the water-jar (Luchins & Luchins, 1959) task and the stick-building task (Lovett & Anderson, 1996). These models show a certain learning effect that can be useful in the area of robotics. In both tasks a series of steps is used to find the solution to a number of problems. When a new, different problem is presented the model still uses the same series of steps, even if there is a better solution. Luchins and Luchins (1959) refer to this as the *Einstellung* effect. However, most importantly, the new problem is solved even though it was not previously encountered, which can be useful in the area of robotics. For example, a model can be implemented on a robot using the just mentioned cognitive models as inspiration. By using such a model, the robot can then learn a specific task. The information gained when executing that specific task can then also be used when executing a second similar task. Since the robot is capable of solving several similar problems it has not encountered before, it is very robust. Using learning mechanisms from the cognitive models, the robot can even increase its performance within the limitations of the complete system, eventually finding a near-optimal solution. This is also referred to as bounded rationality (Simon, 1957). The Einstellung effect and bounded rationality will be discussed in a later chapter in the context of the current research.

Another benefit of using cognitive modelling in the area of robotics can be demonstrated in the field of human-robot interaction (Trafton et al., 2006). Trafton et al.'s work is one of the few examples where a combination of a cognitive model and a robot is used. They argue that robots that use similar representations as humans can better collaborate with humans than robots that do not. To solidify this hypothesis they provide three arguments.

First robots usually have computationally efficient algorithms that use mathematical representations like matrices and polar coordinates. However, in general, humans do not think or reason using those representations. Therefore for robots and humans to understand each other, their expressions have to be translated which is inefficient and might cause some loss of information or confusion. These problems of communication are less likely to present themselves when the robot uses a cognitive model that has similar representations as humans.

Second, a cognitive model ensures that the robot exhibits relatively normal, for humans understandable, behaviour. Conventional robot algorithms may be able to efficiently perform certain tasks, but if humans need to collaborate with these robots and the robots' behaviour seems unnatural because of the conventional algorithms, the behaviour will detract from the interaction. Therefore when behaviours are programmed using cognitive models, the human-robot interaction might improve.

Finally, for some tasks it is beneficial to incorporate knowledge about how humans solve certain tasks. For example, when a robot has to search for people hiding, it is useful to know how humans tend to hide themselves. In this case human intelligence is not only used to create the processes of a model, but also as content for the processes to use. Thus, when programming robots using cognitive models, these robots could gain insight in the beliefs, desires and intentions of humans they collaborate with (or against).

In short, robots are good at low level behaviours but usually lack higher cognitive processes. The field of cognitive modelling has already thoroughly researched these higher cognitive processes and can therefore be beneficial as inspiration for robotics to improve their performance and/or collaboration with humans. Vice versa, the area of robotics can fill the gap on the low end of cognitive models by adding embodiment and low level processes that interact with the higher level processes of a cognitive model.

1.2 Route Learning as a Task Domain

Trafton et al. (2006) performed an experiment based on the game Hide and Seek to support their arguments regarding the benefits of using a cognitive model in the area of human-robot interaction. However, much more can be learned from examining the combination of cognitive modelling and robotics, as demonstrated by the previous section. To explore what insights can be gained from such a combination, a route-learning task was used.

The route-learning task is an interesting task, because it involves spatial reasoning processes and the learning of declarative and procedural knowledge, which are all higher processes. Also it involves movement, a component that is absent in most cognitive models and often regarded as a lower process. Finally to be able to learn a route one also has to perceive the environment, which, as will be discussed, is a difficult component to model.

Route learning as a task is not only interesting to examine the combination of cognitive modelling and robotics, but also is an interesting task for the individual research areas. In the area of cognitive modelling and also psychology in general, a lot of research towards spatial cognition is done. For example, what representations do humans use to represent spatial knowledge and by what processes do they gain that knowledge. On the other hand, in the area of robotics, navigation is researched for obvious reasons. Many robots have to move around and navigate through an environment to be able to complete their tasks. As a result, the algorithms used in robotics have a pragmatic approach that is primarily efficient and robust rather than cognitively plausible.

In short, route learning is a useful task to examine the combination of cognitive modelling and robotics, as well as to examine the psychological aspects and as a means of navigation for robots.

1.3 The Research Goals and Approach

The main goal of the current research is to explore what insights can be gained by combining cognitive modelling and robotics. This section describes what approach will be used to do that and some sub-goals.

A cognitive model can be programmed in any programming language: Lisp, C++, Java, R, etc. However, by default, these languages do not provide any psychological or cognitive constraints. Therefore in the current research ACT-R (Anderson, 2005; Anderson et al., 2004) is used. ACT-R combines several aspects of cognition into one general theory. An implementation of this theory is used in the current research. By using ACT-R, psychological constraints are provided to the second level of Marr's (1982) three levels. In addition Taatgen (2007) proposed a guideline referred to as the minimal control principle. This guideline provides additional constraints which should lead to more plausible and more robust cognitive models.

For the current research there were two robot types available, an Active Media Pioneer 2DX and a Sony AIBO ERS-7. The Pioneer is a three-wheeled robot with a camera on top and several sonar sensors along its sides. The AIBO consists of a body with four legs and a head that contains a video camera and a distance sensor. Since the AIBO's head can be controlled separately from the rest, it matches the modular approach used in the current research best. Therefore, the AIBO was chosen to be used instead of the Pioneer robot.

As mentioned the robot, in this case the AIBO, provides the strongest constraints for a cognitive model, as it defines the third level of Marr's (1982) three levels. Also, since the AIBO with its ACT-R model will operate in the real world, the model has to deal with all problems that come along with it. This, in combination with ACT-R, the minimal control principle and the AIBO, provides a set of constraints that should help to create a plausible cognitive model.

The model created is called AIBO-Route and uses an expanded version of ACT-R, referred to as AIBO-R, because ACT-R by default does not have a way to interact with the AIBO. To explore what insights can be gained by combining cognitive modelling and robotics, AIBO-Route was developed within the constraints just mentioned and with two sub-goals in mind. These sub-goals are:

1. Given a setup of several landmarks the AIBO-Route model should be able to learn a route to a predefined goal.
2. When having learned such a route and the environment changes in such a way that a shorter route is possible, AIBO-Route should be able to learn the new shorter route.

AIBO-Route is a cognitive model that simulates how humans develop route knowledge, but ACT-R also is capable of predicting reaction times and the duration of cognitive processes. However, these temporal aspects were not considered in the current research.

1.4 Outline

The next chapter, Theoretical Background, will discuss several aspects relevant to this project. It will discuss: ACT-R, Spatial Learning, Simulated Navigational Models and Mobile Robot Navigational Models. Then the current research will be discussed in the context of the Theoretical background starting with the interface between AIBO and ACT-R. Next the AIBO-Route model will be discussed, followed by the experiments and results. Then the results will be discussed and in the conclusion will be stated whether the goals were obtained or not. Finally some ideas of expansions to the AIBO-Route model and AIBO-R architecture will be discussed in the Future Work chapter.

2. Theoretical Background

In the previous chapter the goal and relevance of the current research has been discussed. This chapter will discuss the background needed to understand the route-learning model that has been developed in this project.

The chapter will begin with an introduction to ACT-R, the theory used to create the route-learning model of the current research. It will continue with a section about what is known so far about spatial learning and route learning in particular. Next a few models that have been developed to simulate navigational learning will be discussed followed by a section about some navigational algorithms specifically developed for mobile robots that implement navigation.

The different sections will illustrate the difference between robot navigation models and simulated models based on Spatial-Learning theory. The most important aspects of the sections in this chapter are summarized and put into context in the final section of this chapter.

2.1 ACT-R

Over the years, different components of the human mind have been studied in relative isolation. There have been, for example, studies of memory, motor control and decision-making. To get a better understanding of how these components work together a theory was developed. This theory has taken form in a cognitive architecture called adaptive control of thought-rational (ACT-R) (Anderson, 2005; Anderson et al., 2004). This section will discuss how cognition is integrated in the ACT-R theory.

To prevent confusion it is useful to note that the ACT-R theory is not the same as the ACT-R program implemented in Lisp. Unfortunately this distinction is rarely made, but it is important to note that the ACT-R theory can be and has been implemented in several different programming languages. However, most of the cognitive models using the ACT-R theory have been developed with standard Lisp implementation. In the current research an extended version of the standard implementation has been developed. This extended version makes it possible for ACT-R to interact with the Sony AIBO robot and is discussed in the chapter "Interfacing AIBO and ACT-R: AIBO-R".

To explain ACT-R, the general framework will be described first. The components of the general framework, relevant to the current research, will be discussed in the subsequent sub-sections.

2.1.1 General Framework

The ACT-R architecture consists of several *modules*. The default modules are: the goal module, imaginal module, declarative module, visual module and manual module. The goal module is used to keep track of the current goal of a task. During the execution of that task one must also be able to keep track of the current state or sub-goal, which is done by the imaginal buffer. The declarative module represents all factual knowledge, like for example that three plus two is five or that from one's house one has to go left to go to work. Finally, to be able to interact with the world, one has to be able to perceive the environment and act in it, which is the function of the visual module and manual module respectively.

The current state of the model is represented by the content of the *buffers*. Each of the modules has at least one buffer, which is used by a central production system, sometimes referred to as the procedural module. Buffers are used as an interface between the central production system and the modules. For example, the declarative module can retrieve one fact at a time and that fact is placed in the retrieval buffer, which belongs to the declarative module. The fact in the retrieval buffer can then be used by the central production system. All the buffers together are sometimes seen as the working memory of the ACT-R framework.

The central production system uses the contents of the buffers to recognize a pattern and then changes the content of these buffers. In what way the content has to be changed, is determined by production rules. The production rules are if-then rules and together form the procedural memory. If the content of the buffers matches the *if* part, also known as the left-hand side, the *then* part, also known as the right-hand side, is executed by the central production system. When this happens it is said that the matching production rule *fires*.

The buffers and the central production system together form a serial bottleneck for the ACT-R architecture. The buffers limit the processing speed, because they can hold only one fact, called a chunk, at a time and the central production system limits the processing speed because only one rule can be matched against the buffers at a time. Besides that, the matching of a production rule against the buffers always takes fifty

milliseconds. The serial bottleneck represents the single stream of thought one has (i.e., one cannot think of two things at the same time).

Although buffers can hold only one chunk at a time, each module has its own buffer. All these buffers can be used by a single production rule. As a result parallel processing of, for example, memory retrieval and visual perception is possible. Also, it is known that the visual system is divided into a "what" and "where" part, which can also operate in parallel. The "where" part processes the location of objects and the "what" part classifies them. To make the parallel processing in the visual system possible, the visual module has two buffers, visual and visual-location, that respectively represent the "what" and "where" parts of the visual system.

To summarize, ACT-R has several modules that represent components of the mind of which some provide an interface with the world outside the mind. The modules have buffers that can contain a chunk. The chunks from all buffers together form a pattern that can be compared to production rules by the central production system. The matched production rule specifies which modifications have to be made to the buffers. In turn the modules respond to the changes in the buffers and the process repeats.

2.1.2 The Goal and Imaginal module

As mentioned, the content of the buffers determine which production rules can fire. When a certain fact is in the retrieval buffer of the declarative module and another fact is in the visual buffer, several different production rules might match. Not all these production rules serve the current goal and therefore some production rules should be excluded from the possible matches. By adding extra constraints, the number of matching production rules is limited to those that serve the current goal. The content of the goal buffer and imaginal buffer provide these additional constraints. The goal and imaginal buffer can therefore guide a model towards the goal.

Since technically a chunk can have any number of elements, called *slots*, it is possible for a chunk in the goal to have a condition for each possible situation. This is of course not a plausible way to create a cognitive model. The goal buffer should therefore hold a chunk representing the global goal of the current task. To keep track of which actions need to be taken and what sub-goals need to be reached, the imaginal buffer can be used to provide additional constraints.

To help researchers build plausible models, a guideline was designed by Taatgen (2007), which is called the "minimal control principle". The idea of this principle is that one should use as few control states as possible, that is, the number of possible values for the goal and imaginal buffer should be as few as possible. The model should primarily be guided by stimuli from the environment and state of the mind, rather than some artificial state.

Models created by adhering to the guideline tend to simulate bottom up processing rather than top down. A clear illustration of the minimal control principle is given through the task of making tea (Taatgen, 2007). Instead of specifying the sequence of steps that need to be taken to make tea, the steps can be given individually with their conditions (the conditions being the state of the environment when the individual step is appropriate). Thus instead of specifying the list on the left without conditions, it is better to specify the list on the right:

1. put water in kettle	**[if empty kettle]** put water in kettle
2. after step 1, put kettle on stove	**[if kettle with water]** put kettle on stove
3. after step 2, put leaves in teapot	**[if empty tea pot]** put leaves in teapot
4. after step 3, wait until water boils	**[if water boils and leaves are in teapot]** pour water in teapot
5. after step 4, pour water in teapot	

The specification to the right is also able to cope with the situation where there already was water in the kettle, in which case the first step can be omitted. Also, the specification to the right is more flexible with regard to the order in which the steps need to be taken. The step to be taken can be determined entirely by observing the environment. Finally the specification adhering to the minimal control principle also makes it possible to interrupt the task and pick it up later since there is no internal state to keep track of.

All the advantages of adhering to the minimal control principle, such as environmentally driven and a higher robustness, are very important in combining cognitive modelling with robotics. Also, in the field of robotics it is common practice to do bottom-up processing. Through the minimal control principle, cognitive models also tend to have bottom-up processing in addition to top-down processing. As a result, cognitive models are better fit to be used with robots. This is an additional reason why the model in the current research was developed with the minimal control principle in mind.

2.1.3 Procedural Memory

Even though the content of the buffers limit the number of production rules that can fire, it might still be possible for several rules to match the content of the buffers. The process that determines which production rule will fire is called *conflict resolution*. Which production is chosen by the conflict resolution mechanism is determined by the *utility* of the production rules. The utility of a production rule is based on how high the chance is that the production rule will result in a successful completion of the goal and the cost of obtaining it through that rule. It is important to note, however, that these utilities are noisy and the production rule with the highest utility might loose from a production rule with a slightly lower utility, because of the noise. The utility of a production rule i is defined as

$$U_i = P_i G - C_i + \varepsilon, \qquad \text{(production utility equation)}$$

where P_i is an estimate of the probability that if production rule i is chosen the current goal will successfully be achieved. G is the value of the current goal and C_i is an estimate of the cost to achieve the goal using production rule i. Both P_i and C_i are learned from experience. The ε is the noise added to the utility and is determined by the parameter s (see below).

If there are a number of production rules that match, the probability of production rule i to be chosen from all matching production rules n, is calculated by:

$$P_i = \frac{e^{U_i/\sqrt{2}s}}{\sum_{j}^{n} e^{U_j/\sqrt{2}s}}, \qquad \text{(production choice equation)}$$

The probability for a production rule i to be chosen, therefore depends on the utility of all matching production rules n and the noise parameter s which is distributed according to a logistic distribution with a mean of zero and a variance of:

$$\sigma^2 = \frac{\pi^2}{3} s^2, \qquad \text{(logistic distribution variance)}$$

The chance of a production rule to be successful is derived from the times it was successful with respect to the total number of applications of that rule:

$$P = \frac{Successes}{Successes + Failures}, \quad \text{(probability of success equation)}$$

The cost to achieve the goal is obtained in a similar manner by the formula

$$C = \frac{Efforts}{Successes + Failures}, \quad \text{(cost equation)}$$

where *Efforts* is the accumulated time over all the successful and failed applications of a production rule. It is useful to note that the *Successes* are determined by a final production rule that indicates that the goal has been reached. The success counter of all production rules that fired to successfully reach the goal is increased by one. Similarly, the failure counter of production rules is increased by one when the firing of those rules led to the final production rule, which is marked as a failure to reach the goal. The initial values of *Successes*, *Failures* and *Efforts* are respectively, one, zero and 0.05 (seconds).

The above set of equations cause the utility of a production rule to increase when its application led to successful completion of the goal. The utility decreases when the cost (i.e., *Efforts*) becomes higher or the production rule led to a failure. The cost corresponds to the period from the time of the application of a production rule to the time of completing the goal. This time period (i.e., the cost) is averaged over the subsequent use of a production rule. As a result when a production rule causes to quickly reach the goal, the utility of that rule becomes higher.

Most production rules are defined at the start of the execution of a model. However, during the execution of the model it is possible that new production rules are formed, a process called *production compilation*. This process causes two successive production rules to merge into one production rule that has the effect of both. Since the execution of a production rule always costs fifty milliseconds and there might be processes that cost additional time between the rules that are compiled, the goal of a model can be reached faster when such production rules merge. This speedup corresponds to the speedup of the execution of a task as the result of gained experience.

The compilation of two rules is possible only when the output of the first is predictable. Imagine there are two rules where the first rule would request a chunk C from memory representing an action A when encountering situation S, and the second rule would retrieve the chunk C and then perform action A. These rules can compile into a rule that immediately performs actions A upon encountering situation S, thereby eliminating the retrieval of chunk C. If, however, the output of the first rule is not predictable, for example, when there is not a retrieval between two rules, but a perception event, the rules cannot compile. If the rules were to compile anyway, it would lead to a hallucination of the perceived object.

Since new production rules compete with the first of the two production rules it was compiled from, the utility of the new rule cannot be determined in the usual way. Also it would seem likely that the new rule borrows some experience from the two old rules. Therefore the utility of a compiled production rule is calculated from the two rules that formed it. This is partly done by using the following two equations for the chance of success:

$$P = \frac{n * priorP + Successes}{n + Successes + Failures}$$

$$priorP = priorP_{preivous} + \alpha(\mathbf{Old1}P - priorP_{previous})$$

The *n* in the first equation is the initial experience of a new production rule and defaults to ten. **Old1***P* is the *P* value of the first of the two production rules that formed the new production rule. From the equations one can derive that α is a parameter that sets the learning rate. The higher α is, the faster the utility of the new rule will converge to the utility of the first of the two old rules. Since the utility of the new rule will only approach that of the old rule, noise is needed for the new rule to overcome the utility of the old rule. Once the new production rule has been chosen a few times, the ratio of *Successes* and *Failures* might cause it to obtain a higher utility then the old rule. The equations for the cost *C* are derived from the original equations in the same way as those for the chance of success.

2.1.4 Declarative Memory

As mentioned a few times before, the buffers of the modules can contain chunks. Chunks can be defined at the start of a model, acquired through the vision module or learned through reasoning processes. All chunks are stored in the declarative module. What information a chunk can contain can be defined by *chunk-types*. A chunk is therefore always an instantiation of a chunk-type. A chunk-type usually has a number of *slots* which can be filled by other chunks. An example of a possible chunk-type is `addition-fact`, which contains three slots. Two slots, `addend1` and `addend2`, represent the numbers that need to be added and a slot containing the sum, `sum`. A chunk of chunk-type `addition-fact` that represents 7 + 2 = 9 would look like:

```
fact7+2
   isa addition-fact
   addend1 seven
   addend2 two
   sum nine
```

In chunk "`fact7+2`", `seven`, `two` and `nine` are other chunks representing the corresponding numbers.

Production rules can make a request through the retrieval buffer of the declarative module for certain facts. By partly specifying which chunk needs to be retrieved, a match can be found by the declarative module. Of the matching chunks the chunk with the highest *activation* is retrieved and placed in the retrieval buffer. The *activation* of chunks is similar to the utility of production rules and represents the likelihood that a chunk will be retrieved. The activation of a chunk is defined as

$$A_i = B_i + \sum_k \sum_j W_{kj} S_{ji} + \varepsilon, \quad \text{(activation equation)}$$

where B_i is the base-level activation of the chunk *i* and ε is the noise determined by *s* (see below). The other part of the equation specifies the *spreading activation* to chunk *i* from other chunks *j* that are present in buffers *k*. The amount of spreading activation is determined by the sum of strengths of association from chunks *j* to chunk *i* (S_{ji}), weighted by W_{kj}. The idea of spreading activation is to account for context when retrieving facts from memory. It is easier to remember in which direction one has to go at a crossroad when one has the crossroad in view, that is, when a chunk representing the crossroad is in the visual buffer.

The base-level activation of a chunk rises and falls with practice and delay according to the equation

$$B_i = \ln(\sum_{j=1}^{n} t_j^{-d}), \qquad \text{(base-level learning equation)}$$

where t_j is the time since the jth practice of a chunk. As indicated by the formula the base-level activation decays as time progresses. The decay-rate is determined by the parameter d. The effect of the base-level activation is that the more time has passed since a fact (i.e., chunk) was encountered the less likely it becomes to remember it.

The probability that a chunk can be retrieved from memory depends on noise and on the retrieval threshold. The probability of retrieval is determined as

$$P_i = \frac{1}{1 + e^{\frac{\tau - A_i}{s}}}, \qquad \text{(probability of retrieval equation)}$$

where τ is the retrieval threshold and s is the noise parameter, which is distributed according to a logistic distribution with a mean of zero and a variance as defined before. When there is no noise a chunk can only be retrieved when the activation of a chunk is above the threshold.

Besides that a higher activation increases the probability of retrieval, it decreases the time needed to retrieve the chunk. The time needed to retrieve a chunk is defined as

$$T_i = Fe^{-A_i}, \qquad \text{(latency of retrieval equation)}$$

As one can see from the equation the time needed to retrieve a chunk i decreases exponentially with the increase of its activation. As a result, it not only becomes less likely to retrieve a fact as more time passes, but it also takes longer to retrieve the fact.

2.1.5 Remarks regarding ACT-R

Now that the components of ACT-R that are relevant for the current research have been discussed, the theory on spatial learning can be discussed in the next section. In the section discussing navigational models that implement that Spatial-Learning theory, some similarities and differences with respect to the ACT-R theory will be noted. Also, extensions to the ACT-R architecture developed in the current research will be discussed in the chapter discussing the Interface between AIBO and ACT-R.

2.2 Spatial Learning: The Landmark, Route, Survey (LRS) Model

Every day people travel from one place to another. This can be from home to work or the supermarket, but also from one room to another. They travel by foot, bike, car or otherwise. To be able to do this, one needs spatial knowledge, but how exactly does one gain this knowledge? What information do we select from the environment and how do we store and use it? Numerous studies have been conducted to answer these questions (e.g., Aginsky, Harris, Rensink, & Beusmans, 1997; Allen, 1981; Appleyard, 1970; Cohen & Schuepfer, 1980; Darken & Peterson, 2001; Gale, Golledge, Pellegrino, & Doherty, 1990; Goldin & Thorndyke, 1982; Golledge, Gale, Pellegrino, & Doherty, 1992; Heft, 1979; Lynch, 1960; Schweizer, Herrmann, Janzen, & Katz, 1998; Siegel & White, 1975). In this section an overview of these studies will be given to give an idea about how one learns spatial knowledge and a route in particular.

Many of the researchers of spatial learning place their studies in the context of the Landmark, Route, Survey (LRS) model developed by Siegel and White (1975). This section will also discuss Spatial-Learning theory in the context of the LRS model as it provides a solid framework in which the different aspects of spatial learning can be filled in.

The LRS model (Siegel & White, 1975) is the longest standing model of spatial knowledge representation to date. Siegel and White developed the LRS model using many aspects of the work of Lynch (1960), who defined five basic types of spatial knowledge (paths, edges, districts, nodes and landmarks) and their role in spatial knowledge representation.

The LRS model exists of three learning phases. First, while travelling in an environment, one stores objects at important locations. These objects are usually referred to as landmarks and the locations are usually known as waypoints or nodes. In the second learning phase these landmarks are associated with bearing changes and with other landmarks. Finally, various routes that have been learned converge into a network. This network can be seen as survey knowledge of the environment, often referred to as a cognitive map or mental map.

Although one does first gain landmark knowledge, then route and then survey knowledge, it is not the case that they are learned entirely separate. This point is elaborated in the next sub-section. Several aspects of spatial learning will be discussed next, starting with landmark knowledge, followed by route knowledge, survey knowledge and some remarks regarding Spatial-Learning theory in general.

2.2.1 Landmark Knowledge

According to the LRS model, the formation of spatial knowledge begins with landmarks, but what exactly are landmarks? In general sense they are physical objects or properties of objects, which are used to identify a certain location, which Lynch (1960) defines as nodes. Nodes are locations where decisions on how to continue one's journey have to be made. Also landmarks can sometimes be used to keep on the right path. Landmarks usually have unique characteristics. That way they can be distinguished from other objects and identify a specific node. Although usually landmarks have a unique appearance, a series of similar landmarks could also be useful. Such a series could, for example, be used to estimate the distance travelled or guide a traveller along a path to the next node.

The size of a landmark can differ greatly, from a coloured tile to a mountain. Also it does not matter whether the landmark is distant or local. As long as the landmark is

visible from a certain location, it can help identify the role of that location in the environment. Steck and Mallot (2000) call the two types of landmarks local and global landmarks. They conducted several experiments in a virtual environment and found that one uses both types. Some participants relied on one of both types and some switched between the two types, depending on the location. However, the knowledge of both types of landmarks was always present, since when one type was removed the participant could easily switch and use the other type.

According to Lynch (1960) especially people who are unfamiliar with a certain environment, use global landmarks. As an example he states that people who are new to Boston seem to use the John Hancock Building, a very tall building, as an important landmark, whereas people who are familiar with Boston rely more on local landmarks.

According to Steck and Mallot (2000) and also Heft (1979) an important factor for selecting a landmark, is its saliency. Besides that, Heft (1979) and Lynch (1960) also found that the participants of their experiments use any environmental cue available; for example the topography of a location or characteristics of a path. This means that landmarks do not necessarily need to be distinctive objects, like a statue, but might also be another specific property of the environment; a change in the pavement or a specific curvature of the road.

When people travel through an environment they might encounter various landmarks, but not all objects used as landmarks are equally distinctive. Several experiments show that landmarks that are present at a location where a decision has to be made (i.e., nodes), are the ones that are remembered. Researchers (Gale et al., 1990; Golledge et al., 1992; Lynch, 1960; Siegel & White, 1975) explain this by the fact that one's attention is heightened, because of the decision one has to make at a certain location. Because of this heightened attention people are more receptive of their environment and therefore more likely to remember objects at that location sometime later. Landmark knowledge therefore develops as an integral part of route knowledge. One might even state that one develops less landmark knowledge when a route is travelled passively, for example as a passenger of a taxi. This means that landmarks are learned much better in the context of a route then when learned individually, which raises two points.

One point is that the three learning phases that Siegel and White (1975) proposed cannot be too demarcated. Since route learning facilitates the learning of landmarks, it is unlikely that the learning of landmarks entirely precedes the learning of a route. Siegel and White (1975) are not entirely clear on this matter, but some researchers (Aginsky et al., 1997) state that according to Siegel and White's theory one first gains landmark knowledge and when landmark knowledge is complete one gains route knowledge, which is unlikely.

Aginsky et al. (1997) state that if an object is to be stored as a landmark and it does not have prominent characteristics, it is only learned as a landmark when one is learning a route. In other words, if indistinctive objects are to be stored as landmarks, they have to be in the context of a route. Although Aginsky et al. (1997) criticize the LRS model of Siegel and White (1975) of being too demarcated, Aginsky et al. and Siegel and White seem to agree anyway. This is shown by the quotation from Siegel and White's work below.

> *"The prominent role of landmarks in early spatial representations seems to require a special kind of figurative memory. We may call this a "recognition-in-context" memory. It is insufficient when one sees a landmark to know, "I've seen that before." One must know something about that landmark, what it implies, what it is next to, when it last occurred, what its connection is with other landmarks."* (Siegel & White, 1975, p. 27)

Although even Siegel and White themselves are not entirely clear on how the three phases of their theory interrelate, is seems to be the general consensus that they are not entirely separate. Thus, it is important to note that one phase does not have to be completed to start the next phase, neither does one phase end when the next phase has begun. The learning phases are interwoven and have the same goal: to navigate through an environment in a rather efficient way.

The second point raised by the fact that landmarks are stored because of heightened attention as a result of a decision being made, is that active travel is important for spatial learning (Cornell & Hay, 1984; Darken & Peterson, 2001; Gale et al., 1990; Gibson, 1979; Goldin & Thorndyke, 1982; Siegel & White, 1975). Active travel means that one really moves through an environment and one makes decisions on where to go.

Active travel also makes it possible to gain procedural knowledge (Gale et al., 1990). The knowledge of landmarks and their associated decisions is declarative knowledge. However, to use this declarative knowledge, procedural knowledge is needed. Knowing how to move from a given location to another and the ability to identify the routes that facilitate such actions, are examples of such procedural knowledge. Since active travel is not a part of an experiment using a video presentation, participants of such an experiment will have less procedural knowledge than those who learn the same route through a real environment.

Unfortunately there are several studies that base their results on experiments that lack this active component. The results of these studies regarding procedural knowledge can therefore be debated.

In short, several objects can function as a landmark, usually these objects are distinctive with respect to their environment, but not necessarily so. Also a landmark might not be a specific object at all, but rather a distinctive feature of the environment. The heightened attention, caused by a decision being made along a route, is the main cause for an object or feature to be stored as a landmark. If an object is very distinctive however, it may be stored even if it is not at a decision point.

The landmarks can aid in the recognition of nodes or other parts of the route. This means that even when one is lost one might recognize a landmark and remember where one is along a route and continue one's journey. This is also a nice illustration of the minimal control principle (Taatgen, 2007). The traveller is guided by the environment (i.e., landmarks), instead of learning a series of decisions. The difference between these two possibilities will be discussed further in the next sub-section.

2.2.2 Route Knowledge

Landmarks are the building blocks of route knowledge. There are two theories on how landmarks are used for route knowledge. One is that they are used as a part of paired-associate learning or stimulus-response learning (Darken & Peterson, 2001; Golledge et al., 1992; Heft, 1979; Lynch, 1960; Schweizer et al., 1998; Siegel & White, 1975). This kind of learning uses landmarks as a trigger to remember the next node along the route together with a change of direction of travelling (Lynch, 1960).

The second theory is that a series of decisions on how to proceed at each node is memorized, this is referred to as queue or sequence learning (Tlauka & Wilson, 1994). In this case the landmarks at a node are not associated with the decision. This means that when a wrong turn is taken one will continue its journey, but the decisions do not correspond to the succeeding nodes. The most supported theory is the first. There is

however evidence for the second theory as well. As we will see, the environment used in an experiment determines for a large part which theory fits best.

Tlauka and Wilson (1994) support the second theory and also Heft (1979) found that when a location lacked distinctive features, the participants of his experiment resorted to a strategy in accordance with the queue learning theory. Tlauka and Wilson's results even indicated that the participants of their experiment had a preference for queue learning over paired-associate learning, in spite the fact that the landmarks used in their virtual environment were very salient.

However, the navigation task of Tlauka and Wilson's experiment was rather simple. They used a computer simulation of a series of rooms connected by doors. Each room had a landmark and two doors, from which the "unlocked" door had to be chosen. The unlocked door would then lead to the next room. In such a setup it is easier to learn a sequence of decisions than in a real-world environment used by other researchers (Golledge et al., 1992; Heft, 1979; Schweizer et al., 1998), because in those experiments there were more options available than right and left. Also when the participants of Tlauka and Wilson were forced to count backwards during the experiment, they did use paired-associate learning instead of queue learning.

Besides the fact that the possible choices in their simulated environment were limited, the environment lacked differentiation. Each room looked the same except for a landmark. As mentioned before, Heft (1979) found that in an environment, which lacks differentiating cues, the queue learning strategy might be used.

Given the current research it seems, depending on the environment, both learning strategies are used. The paired-associate learning strategy however seems the most likely to be used, since environments usually offer several distinctive features. Only in an undifferentiated setting, for example a part of the city with similar repetitive building style, a forest where it is hard to distinguish trees and paths from one another or a desert, the queue learning strategy is used. Also the paired-associate learning is able to explain the effect that it is easier to remember the next node along a route than the previous node. The reason for this is that there is a strong association between the current node and the next, but a weak association between the current node and the previous. This is explained in detail by Schweizer et al. (1998).

There is one other interesting strategy proposed by Cornell, Heth and Alberts (1994). They propose that in the very beginning of learning a route a rather simple strategy is used. Imagine you have travelled a route once. When you then have to travel this route again and you encounter a crossroad, you simply look down each possible path and take the one that seems most familiar. The right path will seem the most familiar because the first time you travelled it, you spent more time observing features from this path than from the other paths.

The difference between this strategy and the paired-associate strategy is that in the latter, the choice of direction is associated with a landmark and in the first the choice is represented directly by the landmark. Therefore to travel a route, the knowledge of landmarks with its associated decisions does not necessarily have to be present in one's memory. One can imagine however that this strategy is not very robust. Also people are very uncomfortable when they are uncertain of their location. It therefore seems more plausible as an initial strategy and later on as a backup strategy, rather than an alternate strategy.

Given the current research is seems plausible to conclude that, since the environment humans travel in is usually quite differentiated, the most applied technique is the paired-associated technique. When learning the association between landmarks and choice of direction the strategy proposed by Cornell, Heth and Alberts (1994) might help

in learning the associations. In the event of the environment being too undifferentiated humans switch back to queue learning.

It is mentioned several times in the literature that segmentation takes place when learning a route (Allen, 1981; Gale et al., 1990; Golledge et al., 1992; Lynch, 1960; Siegel & White, 1975). This means that the route is segmented into smaller parts. The segments can be placed between individual landmarks and a complete route. Segments are paths between two nodes. The smallest segments are those where there are not any nodes between the two nodes forming a segment. However there are also segments that stretch over a larger distance, for example from one district to another district of a city (Allen, 1981; Lynch, 1960). The different segments therefore from a hierarchical structure. In this structure, the top segment is the entire route and at the bottom is a segment with no intermediate nodes.

Those segments at the bottom are rather simple and the path between them is usually short. There are however exceptions. One exception is when edges (Lynch, 1960) are used. Edges are linear breaks in continuity for example the outer edge of a park, shorelines, railroads or channels. Instead of a series of nodes, such an edge can be followed as part of a route. Besides the fact that edges can serve as guidelines, they can also restrict the possible space in which a route has to be formed. One clear example is a shoreline. Also they can force the route to include a certain node. A good example here is a river with only a few bridges. Edges can therefore assist in the acquisition of route knowledge by limiting the possibilities.

Segmentation also helps to determine the distance travelled (Allen, 1981; Golledge et al., 1992; Lynch, 1960; Siegel & White, 1975). One can estimate this distance by observing the number of segments travelled or by investigating the hierarchical structure representation of the route. Another important factor for the distance estimation is the effort needed to travel the distance. The effort depends on the distance, number of turns and the quality of the path; pavement, slope. The effort therefore not only depends on the time travelled but also on the cognitive effort. Sometimes people prefer a simple route over a more complex shorter route (Lynch, 1960).

Again it becomes obvious that active travel is important with respect to spatial learning. Without active travel one has a poor sense of the effort involved to travel a certain distance. Without the knowledge or inaccurate knowledge of the effort needed to travel a route, it is in most cases probably quite difficult to determine an optimal route.

Active travel also provides proceduralization of declarative route knowledge (Gale et al., 1990). The landmarks and nodes, which are part of a route, are declarative knowledge just as the associated decisions. However to use the declarative knowledge, procedural knowledge is needed. Gale et al. showed that participants who learned a route through a video representation barely developed procedural knowledge. The notion of declarative and procedural knowledge with respect to spatial learning is also described by Golledge et al. (1992) and Colle and Reid (1998).

2.2.3 Survey Knowledge

Survey knowledge is the final stage of the LRS model. Survey knowledge is knowledge that represents an overview of an environment and therefore contains certain relations between objects and places. When people have obtained survey knowledge they can easily locate themselves in the environment. Also they are able to plan new routes not previously travelled or switch between routes. Unfortunately the development of survey knowledge with respect to that of landmark or route knowledge is a slow process (Gale et

al., 1990; Goldin & Thorndyke, 1982; Golledge et al., 1992; Golledge, Ruggles, Pellegrino, & Gale, 1993; Heft, 1979). A cartographical map can speed up the development of survey knowledge, but in that case gained knowledge is of less quality than when gained through direct experience (Darken & Peterson, 2001; Goldin & Thorndyke, 1982).

Most researchers agree on theories about the development of landmark and route knowledge. However, when it comes to survey knowledge, there is some discussion. The main reason for this is probably that there is still little known about how humans exactly form a representation of their environment. This is also one of the conclusions of Golledge et al. (1992; 1993) and Gale et al.'s (1990) experiments.

Gale et al. give three reasons why it is hard to gain insight in the acquisition of survey knowledge. One is pragmatic; it takes a lot of time to have a group of people travelling around several times in a new environment. The second reason is that it is hard to test which survey knowledge the participants have gained. The last reason is that the environment usually cannot be controlled and therefore forces a lot of restrictions upon the experiment. An exception is a computer simulated environment. But those are only recently available.

Another reason for uncertainty regarding survey knowledge might be the fact that one's representation of the environment is not always as good as some people think it is. Siegel and White (1975) for instance report that the representation is often fragmented. Areas of little detail are connected to areas with a lot of detail. Often these areas are even completely separate. Also they mention the fact that when two locations are at the same distance of one's home, the one located downtown is reported closer to home. Besides that Lynch (1960), Appleyard (1970) and Aginsky et al. (1997) found that when people draw a map, the topological and projective relations are usually not retained.

The LRS model explains the learning of survey knowledge as an integration of several routes. In other words after several routes have been learned, these can be integrated into a map. This kind of map would be defined by Appleyard (1970) as a route map. Appleyard also mentioned a different kind of map, namely the survey map. The existence of these two kinds of maps is confirmed by Aginsky et al. (1997).

The route map is a sequential dominant representation. People who use this kind of map would draw a map based on routes (i.e., lines with certain points along them). The survey map is a spatial dominant representation. People using this representation draw maps that resemble regular cartographic maps and include specific landmarks, buildings and districts.

Aginsky et al. (1997) point out that Siegel and White's LRS model (1975) first requires people to obtain route maps and then survey maps. However the results of their experiment indicate that people can directly form both kinds of maps depending on the strategy they use for route learning. They define two kinds of strategies. One is a visually dominated wayfinding strategy and is similar to the theory described in the Route Knowledge sub-section.

The other strategy is a spatially dominated wayfinding strategy, which relies on a mental map. People who use this strategy start with a rough map-like representation and use landmarks to position themselves on their mental map. The idea is that one has a sense of the distance travelled and one's change of direction. This information is used to determine the current location with respect to a previous location and store the relation between the locations in the mental map. Landmarks are used as additional indicators where one is on one's mental map. Since a new environment cannot be mapped before travelled through, people expand their mental map as they go.

Aginsky et al. do not suggest that people use only one strategy. It is possible for people to switch between strategies for different parts of a route. Also, the existence of two strategies does not exclude Siegel and White's (1975) idea of route maps preceding survey maps. Therefore Aginsky et al.'s findings could be used to expand the LRS model.

It is interesting to note that Appleyard (1970) found that of people who lived in an area for longer than one year forty percent drew spatial dominant maps, while of people living in an area for less than one year only twenty percent drew spatially dominant maps. Appleyard also found that of people who travelled by bus only twenty percent drew coherent maps, whereas almost all of the car travellers drew maps that were coherent and continuous. In addition, Gale et al. (1990) found that survey knowledge of participants observing a route through a video representation was inferior to that of participants who travelled the same route through the real environment. The results of Appleyard and Gale et al. again stress the importance of active travel.

For survey knowledge the segmentation process, discussed in the Route Knowledge sub-section, is again very important (Allen, 1981; Lynch, 1960). It helps to create a structure that is similar to the one used in route knowledge. In route knowledge the segments exist of one-dimensional parts whereas in survey knowledge they exist of two-dimensional parts. Thus people do not have a single comprehensive image, but rather sets of images which are interrelated in a hierarchical structure (Lynch, 1960). The images can be of different levels, from street level to the levels of district or city depending on the reasoning level. That means when someone travels through a city they tend to think of streets and districts, but when travelling across country they tend to think in terms of cities and highways.

Allen (1981) also describes that the learned route segments can help estimate distances. These distances can then be used to place points on a map. In Allen's experiment, participants developed survey knowledge rather quickly while in Golledge et al.'s (1992) experiments, although the participants had learned several routes through an environment, survey knowledge was only partially available. The participants could judge some relative directions and distances between points, but still made a lot of errors. The results of Golledge et al. therefore indicate that knowledge of multiple routes alone is not enough.

There are some differences between the experiments of Allen and Golledge et al.. Golledge et al.'s participants were children of 9- to 12-year-old, while Allen also tested participants of 18- to 24-year-old. Since Allen also tested older subjects his results might be more reliable. However, Allen used a series of slides while Golledge et al. used a real environment, which in turn might make Golledge et al.'s experiment more reliable.

Because of the series of slides the time between nodes was a lot shorter in Allen's experiment as well as the overall time of the experiment. Besides these differences, the environment represented by Allen's slides was also much more diversified. Golledge et al. speculate that the differences indicate that it might take more time to learn survey knowledge when the time between nodes is longer. Also a more distinctive environment might make the development of survey knowledge easier.

2.2.4 Remarks regarding Spatial Learning

A few remarks can be made about the study of spatial knowledge. First the methodology used in studying spatial knowledge differs from slide presentations to video presentations to virtual environments and real environments. Real environments are hard to control, but are of course the best since they are equal to the real world. Virtual environments come close to the real world, but lack the richness of detail of the real environment. Virtual environments, just as video presentations and slide presentations, also lack a sensorimotor

component, which is important for route learning (Cornell & Hay, 1984; Darken & Peterson, 2001; Gale et al., 1990; Gibson, 1979; Goldin & Thorndyke, 1982; Siegel & White, 1975). Video presentation and slide presentations both lack decision making and slide presentations also lack the sense of time passed during travel. The results obtained by experiments using these different methodologies therefore have to be studied very closely to determine their validity.

A second remark that can be made is about the LRS model. Over the years there has been some critique. However, the general framework is still widely accepted, though some modifications or elaborations can be made to it as discussed in the previous sub-sections. Also some researchers (Aginsky et al., 1997; Colle & Reid, 1998) seem to disregard the fact that the LRS model is intended for large scale, outdoor environments.

Colle and Reid (1998) describe an alternative model which uses two modes of learning. One mode is used when inside a single room and the other for between-room relations. However the latter has several components very similar to components of the LRS model. The first mode is about the representation of objects within a single room and explains that learning their spatial relations is a different process than that described by the LRS model. This is supported by Aginsky et al.. They describe a girl who after brain damage is still able to name the locations of objects within a room, but not of landmarks in a city (Clarke et al. 1993 in Aginsky et al., 1997).

A final remark can be made about testing for spatial knowledge. It is very hard, especially testing for survey knowledge. Usually map drawings are used, but these also depend on the skill of someone to draw its own spatial knowledge. Also Heft (1979) suggested that map-like survey knowledge might only be produced when asked for.

2.3 Simulated Navigational Models

Now that the theory regarding spatial learning has been discussed, we can move on to some models that implement this theory. As we will see some models focus on implementation of the theory and others have a more pragmatic approach. The models that will be discussed are respectively: TOUR (Kuipers, 1978), Qualnav (Kuipers & Levitt, 1988), NAVIGATOR (Gopal, Klatzky, & Smith, 1989), PLAN (Chown, Kaplan, & Kortenkamp, 1995) and ARIADNE (Epstein, 1997).

2.3.1 TOUR (Kuipers, 1978)

The TOUR (Kuipers, 1978) model is one of the earliest cognitive models of spatial learning developed. The environment used in this model is a simulated city block. The model uses several kinds of descriptions that represent this environment. They contain several elements like streets and places and, as we will see, they do not have to be complete, but can be completed using the remaining descriptions. After the model has learned most of the descriptions that can be learned, this set of descriptions can be seen as a cognitive map.

2.3.1.1 The Model

The goal of the TOUR model (Kuipers, 1978) is to gather information to form a complete cognitive map. To accomplish this, the model divides spatial knowledge into five categories. For each of the categories the model defines a representation for that kind of knowledge.

1. Routes. Routes are represented by a sequence of actions that take a traveller from one place to another. The description of a route represents knowledge from three sources, namely: observations from the environment, recalled versions of previously travelled routes and intermediate states of the route-planning process.
2. The topological structure of a street network. This structure represents the order of places along a street and the local geometry of the intersection of two streets. This information is obtained from the route descriptions.
3. The relative position of two places. This is defined by a vector with respect to a coordinate frame. The vector is used to indicate the direction of a street with respect to the coordinate frame. As a result, the angle between two crossing streets can be obtained by using their vectors.
4. Dividing boundaries. This kind of knowledge is used to separate two regions from each other. Boundaries can be very useful when planning new routes since they limit the search space.
5. Regions. Regions are separated by boundaries and contain several elements, such as places and paths. The relations between these elements are again very useful to plan new routes.

These five categories are used by three representations, namely: the representations for knowledge about a particular environment (i.e., declarative knowledge), a description of the current position of a traveller and representations of inference rules (i.e., procedural knowledge). The knowledge about a particular environment is divided into the five categories just described.

Usually information about a particular environment is not complete. To add information to the representation of this particular environment (i.e., to improve the cognitive map), inference rules are used. The inference rules use information about the

current location and actions to complement partial knowledge of one kind with partial knowledge of another kind. To explain this in more detail, first a few examples of representations of environmental knowledge will be discussed followed by a description of the process by which the inference rules add information to the cognitive map.

Route knowledge in the TOUR model consists of a series of TURN and GO-TO descriptions. The TURN description provides a selection of the next path to follow given a previous path at the location of an intersection. The GO-TO description describes a segment from such an intersection to a next intersection.

The TURN description consists of a PLACE, which is the location of the intersecting two paths, the two paths themselves, directions of the paths, and the angle between the paths. The GO-TO description consists of two places, the path they are on, a direction indicating which way of the street the next place is, and the distance between the two places. A PLACE is a description of an intersection. It contains the name of the place, the involved paths and the angle between the paths. A PATH description contains the name of the path and the places on them. As mentioned before the descriptions do not need to be complete, which means that each of the elements just discussed can be missing from a description.

The transfer of knowledge can be described as follows. The inference rules can copy elements from one description to another description, for example from the current location and a PLACE description to a TURN description. Imagine an action is given to turn right and the current PLACE is X and on PATH Y. Since a right turn indicates ninety degrees and the PLACE X contains information that indicates that PATH Y crosses PATH Z with ninety degrees at PLACE X, all the elements of a TURN description can be filled. The resulting TURN description will contain PLACE X, PATH Y, PATH Z, the direction of Y and Z and the angle ninety degrees. In the same way a GO-TO description could be filled using the current location, which is known, and the location moved to.

A more elaborate example is given in (Kuipers, 1978). The basic idea is that while moving around in the environment information from the several descriptions that are incomplete are completed using elements from the remaining descriptions. These complete descriptions then form the cognitive map. The descriptions can form a cognitive map because there is an overlap in the descriptions of paths, turns and places. Since the descriptions contain distances and angles, there is enough information to draw a map consisting of a network of several streets and places with accurate relative distances.

2.3.1.2 Remarks

It is interesting to note that the TOUR model is modelled after the “production systems” of Newell and Simon (1972) as Kuipers mentions in (Kuipers, 1977, p. 81). Several descriptions about the environment can be seen as part of the long-term memory and the working memory contains descriptions currently operated on by the inference rules. Also the environmental descriptions are very similar to the syntax of ACT-R chunks discussed earlier where the elements contained by a description are several slots of such a description chunk. Also the current location can be seen as the content of the imaginal buffer.

Kuipers (1978) also mentions that the angle used in environmental descriptions cannot contain all 360 degrees, as that would be implausible. Humans are more likely to be accurate to about eight headings and therefore the 360 degrees are split up in 45 degree intervals. Also the second category of environment knowledge only mentions topological representations, but since distances and angles are available in TOUR, metric representations are also available.

Unfortunately the TOUR model lacks sensory impressions. The TOUR model recognizes locations simply because they are labelled. No landmarks are used to identify a location, nor is there any perception when moving from one place to another. Kuipers (1978) considers the perception as very important, but also as primitive and opaque.

In more recent work Kuipers (1983; 1988) introduces the concepts of *views* and *actions*. *Views* are defined as the sensory image received by an observer at a particular point. It is used to identify particular points so that the appropriate *actions* can be taken. *Actions* are defined as a motor operation that changes the current *view* by changing the heading of location of an observer. With *views* Kuipers again acknowledges that the sensory impressions of a place are important. Also he admits that in the TOUR model uses the concept of *views* only to compare different places.

The analysis of *views* and *actions* of Kuipers (1983) is quite interesting. He defines several combinations of *views* and *actions*. A route is described as a series of them; $V_0\ A_1\ V_1\ \ldots\ A_n\ V_n$. Each *view* is associated with an *action* and the combination of both leads to a new *view*. This idea is defined by Kuipers through the following syntax: (1) $V \rightarrow A$ and (2) $(V\ A) \rightarrow V'$. First knowledge of type (1) is obtained and later on knowledge of type (2). This is very close to the paired-associate learning theory previously discussed in the Route Knowledge sub-section. This representation leads to a possibility that humans are able to take someone somewhere, but are not able to tell them how to get there themselves. The reason is that the *views* along the way are needed to remember the correct action at the place with which the view is associated.

He also investigates a few other combinations of *views* and *actions*. One of the more interesting alternatives is: (1) $V \rightarrow V'$ and (2) $(V\ V') \rightarrow A$. This representation causes people to be able to recall the sequence of landmarks encountered during a specific route, but not always which *action* is needed to get from one landmark to the next.

The interactions between *views* and *actions* are part of what Kuipers and Levitt (1988) describe as the sensorimotor interactions level. This is one of four levels (sensorimotor interaction, procedural behaviour, topological map and metrical map) that according to Kuipers and Levitt can facilitate robust navigation and mapping systems (i.e., computational models and robots). The TOUR model is also described in terms of these four levels, but that description will not be discussed here. Instead the Qualnav model (Kuipers & Levitt, 1988) will be discussed in terms of the four levels.

2.3.2 Qualnav (Kuipers & Levitt, 1988)

While TOUR (Kuipers, 1978) is developed for an urban environment, the Qualnav (Kuipers & Levitt, 1988) model is meant for open terrain. Such a terrain might be a region containing forest, mountains and plains. The environment is simulated and the model Qualnav navigates in it. Also where the TOUR model almost completely ignores perception, the Qualnav model focuses on landmarks and perceived relations between them. Finally TOUR is developed as a cognitive model of spatial learning, while Qualnav focuses on using the Spatial-Learning theory to create a navigation model for use in a robot. The Qualnav model will now be discussed in the terms of the four levels mentioned above (sensorimotor interaction, procedural behaviour, topological map and metrical map).

2.3.2.1 The Model

The **sensorimotor interaction** level contains the relations between perception, action and the environment. The most important element of the sensorimotor interactions level of the Qualnav model is the *viewframe*. A viewframe is a data structure that encodes the observable landmarks around the model at a certain location. The viewframe can

therefore uniquely identify locations associated with it. The action associated with a viewframe, is represented by a vector indicating the angle and distance to the next viewframe.

The landmarks are marked in the simulated environment by the experimenter. The experimenter can choose points that would resemble a landmark in the real world (e.g., a mountain top or distinguishable tree). The Qualnav model can perceive landmarks whenever the robot has a line of sight to it and perceive and store their angle and distance. To keep the model realistic an error is always added to the angle and distance. Less realistic, with respect to humans, is the 360 degree view the model has, since humans can only see what is in front of them.

The **procedural behaviour** level contains stored and learned procedures defined in terms of the elements from the sensorimotor level. The procedures facilitate route finding. In the case of the Qualnav model the most important element at the procedural level is a route *heading*. A heading is constructed in a learning phase in which the model is guided by the heading direction specifier. While moving through the environment headings and other environmental information is stored by the model for future planning.

A heading consists of four elements: type, destination goal, a direction function and termination criteria. There are three kinds of headings: absolute, viewframe and orientation. These three types will now be discussed in the terms of the remaining three elements.

The absolute type uses coordinates that correspond to a fixed absolute coordinate frame. The information used to create this type of heading can come from GPS, dead reckoning or a previously defined map-like representation. The destination goal of an absolute heading is a pair of absolute coordinates. The direction function is used to keep the model on track towards the destination goal. In the case of an absolute heading, the direction function calculates the distance between the current absolute position and the destination goal. Finally, the termination criterion is the error of the estimation of the current absolute position. If the error is too high, the execution of the heading is terminated. Of course, for all heading types the arrival at the destination goal counts as a termination criterion as well.

The second heading type is the viewframe heading type. The destination goal is the viewframe data structure discussed earlier. The angles between the heading vector and the observed landmarks are calculated and used as the direction function. Another possibility for the direction function is to maintain visibility of landmarks in the destination viewframe and use a hill-climbing strategy. The termination criteria again follow directly from the direction function. For example when the model loses sight of the landmark or the hill-climbing strategy fails.

The last heading type is the orientation heading type. As a destination goal it has an orientation region. An orientation region is an area on the ground enclosed by a set of Landmark-pair boundaries (LPBs). A LPB is a virtual boundary that can be drawn between two landmarks. It divides an area into two regions, one on either side of the virtual boundary. A set of these LPBs can define several orientation regions and determine relations between several landmarks. As just mentioned, one such orientation region can act as a destination goal. The direction function of an orientation heading type keeps track whether the model is right, left or between landmarks. The execution of an orientation heading is terminated when the models loses sight of the landmarks or an LPB has to be crossed a second time, which means that the model is travelling back.

To reach a certain goal, several algorithms are used to calculate a path and the needed headings. These algorithms include A* and an algorithm that can use LPBs. The last algorithm is explained in more detail in (Kuipers & Levitt, 1988). The model operates

in a loop that contains three stages. The first is to determine a destination goal, the second to compute and select a current heading and the third and last builds representations of the environment while travelling.

The **metric map** level contains a description of the environment in terms such as places, paths, landmarks and orientation regions. At the metric map level these are linked by metric relations like distances and angles. The primary data structure of the metric level in Qualnav has already been discussed and is the viewframe. As discussed, the viewframe contains distances and angles with respect to landmarks. These values can be used for all sorts of vector calculations to obtain new relations between landmarks perceived. From the results of these calculations and the existing viewframes a metric map can be formed. This map would contain distances and angles of landmarks with respect to a coordinate frame.

The **topological map** level also defines relations between elements from the environment, but this time they are linked by topological relations. Among these relations are relations of containment, order and connectivity. In the Qualnav model these relations are derived from the LPBs and the orientation regions defined by them. The topological map constructed from these elements is a very robust representation and can be used for powerful path planning. It is powerful, because it is less susceptible to errors, which tend to accumulate when building a metric map. Also the topological map seems to be a closer resemblance to the cognitive map of humans than the metric map.

2.3.2.2 Remarks

All four levels have now been discussed and a few interesting remarks can be made. As Kuipers and Levitt (1988) mention, robot navigation and guidance has traditionally been quantitative, relying on accurate knowledge of distances, directions and other metric data. Existing robot navigation techniques include triangulation, ranging sensors, stereo vision, dead reckoning, GPS, etc. These techniques are usually not very robust and tend to accumulate errors. Also, robot navigation algorithms usually try to focus on optimizing the metric information, while common sense or knowledge from human Spatial-Learning theory can be very useful in such algorithms. This is exactly what Qualnav claims to do.

Although the results of Qualnav seem impressive, it must not be forgotten that Qualnav uses a simulated environment while the critique of Kuipers and Levitt is directed at robots that navigate in a real environment. They do realize this, and therefore their conclusion is that the theory and implementation of Qualnav demonstrates that human Spatial-Learning theory can be very useful for robust robot navigation models, but much work still needs to be done.

One particular thing still needs to be done and that is the reliable object detection and classification by robots. The Qualnav model does not have this problem since it uses a simulation, but in the real world this is quite hard.

With regard to the Spatial-Learning theory the model captures some interesting and useful elements such as landmarks and sequences of landmarks and actions to navigate. However the algorithms used to plan a new route, for example A*, are not cognitively plausible. In the human mind there are other mechanisms responsible for finding a new route as described in the Spatial Learning section. The next model that will be discussed is a model that focuses on which information from the environment is encoded in the brain and the encoding processes themselves.

2.3.3 NAVIGATOR (Gopal et al., 1989)

NAVIGATOR (Gopal et al., 1989), is based on psychological research towards spatial learning. Although based on psychological theory the model is not an exact simulation of

spatial learning. The goal is not to fit data gathered through navigation experiments, but to gain insight into the acquisition of spatial learning by humans.

NAVIGATOR exists of two modules; the objective environment and the individual's subjective representation of that environment. The subjective representation is gained through the navigation system (NS) that travels through the objective environment. First the environment module will be discussed and then the NS module.

2.3.3.1 The Environment

The objective environment represents a city block with a grid-like structure of horizontal and vertical streets. On this grid plots and decision points, in the Spatial Learning section referred to as nodes, are located.

A plot consists of a location and associated objects, such as a lamppost or a car, which might be used as landmarks. Each object can have several properties, such as colour and shape. These properties are called *type* properties and there are also *relational* properties that describe spatial relations between objects. Examples of *relational* properties are: left-off, adjacent and near. To differentiate between noticeable and less noticeable objects a saliency value is given to each object. This way the NS can select more salient objects over less salient objects, just as humans would. All elements discussed in this paragraph are described using a predicate calculus-based language.

Decision points can be located at either corners or non-intersecting parts of the grid. Each decision point is associated with one plot and each plot with one or more decision points. From the decision points, objects from the associated plot are visible, but also objects at more distant plots. Given the grid-like structure of the environment, the NS can perceive information from four directions. All possible information perceived from one of these directions is called a scene.

These scenes are used to remember actions taken at decision points. In other words, in the NAVIGATOR model scenes are associated with navigation actions in accordance with the paired-associate learning theory previously discussed in the Route Learning section. There are two types of action, one is perceptual and the other is locomotor. There are four perceptual actions; that is perceiving information from one of four directions, front, leftward, rightward or behind. After perceiving one of the directions, a locomotor action could allow the NS to move towards that direction, if feasible. Such a locomotor action would take the NS to another decision point.

2.3.3.2 The Navigation System

The objective environment just described provides most of the elements used in spatial learning. The second module of the NAVIGATOR model, the Navigation System, represents an individual moving through that environment. The NS module consists of perceptual and memory structures and procedures that operate on these structures. Gopal, Klatzky en Smith (1989) unfortunately do not describe the procedures in detail. Therefore only the perceptual and memory structures will be discussed next.

The memory consist of 'long-term memory' (LTM) and 'working memory' (WM). The WM is seen as a transiently active subset of the LTM and therefore uses information from the LTM. Besides information from the LTM, the WM also receives information from the perception system. This information is obtained in two stages. First salient objects are selected and then salient scenes. A salient object and its properties are passed through an object filter that tries to match them with propositions in the LTM. If there is a match and if the total saliency of the object and its properties exceeds a certain threshold, the object and its properties pass the filter. Using the object filter output, the scene filter tries to match the objects and their properties with scenes stored in the LTM.

Again if there is a match and the saliency is above a threshold, the scene is passed. When a scene has passed the scene filter the objects and their properties in that scene are passed to the WM.

It is important to note that the initial saliency of objects depend on objects distance and direction from the observation point. The saliency becomes lower when an object is further away and also when an object is perceived sideways instead of directly ahead.

Besides information from the objective environment the perception system can also receive direct instructions. These instructions can for example guide the NS through the environment during a learning phase. If an instruction is received to go from A to B, a link is stored in the LTM between A and B. Another possibility for a link to be stored in the LTM is when information from two locations is present in the WM at the same time.

Information from the WM is stored in the LTM, but not always. The saliency of information in the WM decays exponentially over time. Also the capacity of the WM is limited. If new information enters the WM and the WM is full, the information with the lowest saliency is removed from the WM. If the current saliency of this removed information exceeds a threshold it is passed to the LTM. Once in the LTM, the saliency of the information again begins to decay, influencing its use in the future.

2.3.3.3 Experiments and Remarks

Using the NAVIGATOR model several experiments have been conducted that show some interesting results. The general setup of the experiment was to change one or more parameter settings, for example the decay rate of the WM or LTM or the threshold of the object filter, and observe the number of elements learned. Four types of learned elements were counted: the number of decision points, the number of objects, the number of actions, and the number of properties (of objects). The parameters that influence these counts could have two settings, high or low. High indicates a ‘low performance’ of the model and low indicates a ‘high performance’ of the model.

One of the results found was that the object filter and scene filter thresholds seem to influence the number of elements the most. However when the threshold of the object filter was set to ‘low performance’ the parameter for the scene filter had no longer an effect on the number of elements learned. This means that the mechanism represented by the object filter plays an important role in spatial cognition.

The influence of the object filter could however be compensated by a ‘high performance’ setting of the WM parameters. This indicates that people with a good WM can compensate for a bottleneck in the initial perception.

Another interesting result was that the saliency of an object always had an effect, whether the parameters were set to ‘low performance’ or ‘high performance’. This indicates that people who have poor recognition capabilities, for example mentally impaired people, can still learn to navigate in a highly salient environment, but would have trouble in a less salient environment.

Finally the model was run three times in a row to demonstrate that the saliency of objects and properties approached the saliency value they had in the environment. This indicates that objects and properties are better remembered after having seen them multiple times.

A lot of similarities between NAVIGATOR and the ACT-R architecture can be noted. The predicate calculus-based language is quite similar to the syntax used for ACT-R chunks. Also the idea of a WM and a LTM are present in both, as well as the decay of items in the LTM. An interesting difference however is that the initial saliency, called

activation in ACT-R, is determined by the environment in NAVIGATOR. Each object has a measure for its distinctiveness and is therefore easier or harder to be remembered.

Although NAVIGATOR incorporates many elements of general cognition, like memory and forgetting, it does simplify sensory processing. Just as in most models the perception of the environment is treated as an opaque subject. It seems that a lot of interesting work needs to be done in the field of perception before a complete model of navigation can be created. NAVIGATOR is of course far from complete as it also lacks mechanisms to create survey knowledge. Nonetheless, NAVIGATOR provides quite interesting insights into spatial cognition.

The next model that will be discussed is PLAN (Chown et al., 1995) which attempts to integrate the navigation process into general cognition. In that sense PLAN is similar to NAVIGATOR, but it is more elaborate.

2.3.4 PLAN (Chown et al., 1995)

PLAN (Chown et al., 1995) has combined several known theories to create a complete model of cognitive mapping. PLAN adheres to the developmental theory discussed in the Spatial Learning section and attempts to specify the mechanisms needed to acquire spatial knowledge. The mechanisms are discussed as four different problems: landmark identification, direction selection, path selection and environmental abstraction. Although discussed as four different problems Chown, Kaplan and Kortenkamp acknowledge that the problems are not separated, but would need to interact to create a complete model.

While in the Spatial Learning section, there were three sub-sections (Landmark, Route and Survey Knowledge), here there are four. The direction selection and path selection, however, both belong to the Route Knowledge sub-section. The distinction between the two is rather interesting and the two are analogue to the “what” and “where” system identified by research concerning the visual system of humans (for multiple references see: Chown et al., 1995).

PLAN uses the previous work of NAPS (Network Activity Processing Simulator) (Levenick, 1991) to build a complete model of cognitive mapping. The problem of path selection is completely solved using NAPS. The other three problems extend the work already done in NAPS and therefore NAPS will be discussed first.

2.3.4.1 Path Selection: NAPS

In NAPS nodes in a spreading activation network represent landmarks from an environment. In this network only neighbouring landmarks are connected, which means there must be a direct path between two connected landmarks and one landmark should be visible from the other. A connection between two landmarks therefore represents a path. Since only the sequence of landmarks is stored only topological information is available. This information is gained through the “what” part of the visual system.

A route can be found by activating the start node and the goal node. Activation will start spreading from both nodes and collide at some intermediate node that becomes a sub-goal. Next the start node and the node representing the sub-goal are activated and the process is repeated. This way a sequence of landmarks and the paths between them can be found, together representing a route.

The connections between the nodes do not all have the same strength. A certain sequence of nodes can therefore have a higher activation than another. Also nodes visited more frequently than others gain a higher activation. This means that the network will show a preference over familiar routes, which is consistent with the theory discussed in the Spatial Learning section.

NAPS also creates a hierarchy of nodes. Nodes of higher levels can for example represent complete routes that are represented by several nodes at a lower level. Searching these higher levels uses the same mechanism as the one described for the lower level. By using higher levels, it is easier and faster to find longer routes.

Although the hierarchy of nodes is not present in the model developed in this project, the idea of connections between two landmarks connected by a direct path is. Two landmarks are stored in a `route-element` chunk, and multiple of these route chunks together form a path or route. Since these chunks have an activation value as described in the ACT-R section, the `route-element` chunks together show similar characteristics as the NAPS network. For example, through activation the model also develops a preference for familiar, more frequently travelled routes. Also, the next point along a route is determined by the `route-element` chunk with the highest activation, just as a node with the highest activation would be selected in the NAPS network.

2.3.4.2 Landmark Identification

NAPS specifies how landmarks are connected, but not how a landmark is recognized and identified so that the corresponding node can be selected. By PLAN, the landmark identification problem is treated as a special case of categorization. The idea is that the sensor input containing a certain object can differ greatly. The sensory data depends for example on the angle the object is approached at and the lightning conditions it is observed in. Each of these sensor input variants represents a unique instance of a special category. This special category is the landmark to be identified.

To tackle the problem of categorization, the prototype theory was developed. This theory is discussed in more detail in (Chown et al., 1995) and will not be discussed here. Although the approach just discussed is quite an interesting approach to the landmark identification problem, it still does not describe what the mechanisms that are needed for the categorization problem, look like.

2.3.4.3 Direction Selection

Besides landmark identification, NAPS also does not specify how locational relations between landmarks are obtained. According to PLAN while travelling a route, locational relations between landmarks are learned automatically. This information is gained through the “where” part of the visual system mentioned briefly in the introduction to PLAN. The locational relations give the model an advantage, since it does not always have to search for the next landmark, but can start moving in the learned direction. As will be discussed, starting to move in the learned direction is exactly what the model developed in the current research does. For more detail see “The AIBO-Route model” chapter.

When the model travels a route, it stores the landmarks and the associations between them in NAPS. Similarly, since the next landmark is always visible, the relative spatial information can be stored. At the same time, the relative spatial information of other visible landmarks, up to 5 ± 2, can be stored. The relative spatial information is stored using local directional representations called *local maps*.

Local maps are created at points where the model would stop and head in a new direction. This means local maps are not always created at a landmark, but usually somewhere near it.

To acquire the relative spatial information a simple elegant method is used. The relative location of an observed object with respect to one’s body can be derived from the position of that object in one’s field of view and the angle between one’s head and body.

Since the body can still have several orientations, a reference point is needed. To solve that problem a robust local solution is used, instead of a global frame of reference.

The particular direction in which one faces a landmark depends on the landmark previously passed. Assuming one always uses the same path to reach a landmark the direction of facing that landmark is always the same (and so is the direction of the body). That direction can therefore be used as a reference to store the relative direction obtained from the field of view and the position of the head with respect to the body. Since it is not plausible to store the direction is exact degrees, intervals of 45 degrees are used. The local directional information can then be stored in a "local map".

Analogue to NAPS, which stores the topological relations between landmarks, the locational relations (i.e., local maps) are also stored in a spreading activation network called R-Net. Each time a new location is reached and a local map is generated a connection is created between that local map and the one from the previous location. In a similar way as NAPS, the R-Net can now be used as a second method to find and travel a route.

Just as the NAPS network is similar to a collection of `route-element` chunks, the collection of `relpos` chunks, which contain relative directional information, is quite similar to the R-Net network. Also the selection mechanism is similar since, obviously, `relpos` chunks have activations too. The `relpos` chunk is discussed in more detail in the "Interfacing AIBO and ACT-R: AIBO-R" and "The AIBO-Route model" chapters.

2.3.4.4 Environmental Abstraction

The environmental abstraction or survey maps have a similar structure as NAPS. The local maps just discussed, form the building blocks of the survey map. When one has travelled a route many times, the activation of local maps and the association between them have become very high. The associations between local maps have a predictive power that causes one to imagine the next point along the route when one pauses for a moment at a certain point. The pause is necessary because of the cognitive workload as is explained next.

In PLAN such pauses occur naturally at *gateways*. Gateways are points along a route that lead to a new perceivable environment that is relatively large. Examples of gateways are, doors, clearings in a forest or paths emerging from the edge of a forest, and a pass through the mountains. Such locations usually require a new decision on where to go and also present several new landmarks. Because a lot of information has to be taken in and processed, a natural pause occurs during which one can imagine the relations between several local maps. This process causes a new element, the regional map, which is a group of local maps.

Regional maps are conceptualized as abstractions of the group of local maps, thereby losing some information, which is quite efficient, since the lost information is still present in the local maps. The regional maps can then be linked together, representing an even larger area. This process can continue, each time conceptualizing a group of representations from a lower layer in the network.

This network with a natural hierarchy provides efficient and natural planning. For example when navigating through a city one first thinks in global terms moving to the desired area and navigating more precisely when approaching the goal.

2.3.4.5 Remarks

A few interesting remarks can be made regarding PLAN. It is adheres very closely to the Spatial-Learning theory discussed earlier and seems cognitively very plausible. It does not however use a description with the same level of detail as NAVIGATOR. Also PLAN

as just discussed was not implemented and NAVIGATOR was. PLAN has been implemented as R-PLAN on a robot, but the document discussing R-PLAN was listed as submitted and never published. R-PLAN therefore has not been discussed here.

An example of the adherence of PLAN to the Spatial-Learning theory is the segmentation discussed with respect to both route knowledge and survey knowledge. The hierarchical network representations of NAPS and regional maps seem to be an ideal implementation of the segmentation phenomenon, both in structure and in function.

With respect to landmarks, PLAN is somewhat brief. Chown, Kaplan and Kortenkamp (1995) do mention, however, that the landmark part of NAVIGATOR might be a useful way of representation. What all models seem to be lacking however, is the perception process by which objects in the environment and their saliency are coded. PLAN provides only a brief online on how to do that. This process remains a very difficult and opaque subject, but is essential to all representations of spatial knowledge.

The local maps are quite an interesting way to store spatial relations. Since they rely on local information only they are very robust. At each decision point a local map is created with respect to nearby objects, therefore errors cannot accumulate since there is no position to be kept with respect to a frame of reference. This is quite different than most models and especially robot navigation models that usually use GPS, dead reckoning or some other absolute navigation system to create absolute cartographic-like maps.

The two models, NAVIGATOR and PLAN, both stay very close to the cognitive theory. To show the contrast with other less cognitive plausible models, one last model will now briefly be discussed, called ARIADNE (Epstein, 1997). As we will see it has a more pragmatic approach to solve the navigation problem.

2.3.5 ARIADNE (Epstein, 1997)

The purpose of ARIADNE (Epstein, 1997) is not to simulate the cognitive processes of cognitive mapping but to simulate a robot that learns the features of an environment. Its goal is to provide a model with robust navigation that is resilient to changes. Also the performance has to increase over time. The environment navigated in is not known before hand, but its useful features have to be learned while navigating in it. These features include doors and extended walls.

The environment is a grid of squares, for example fourteen by fourteen squares. Of these squares thirty percent are randomly marked as an obstruction. Since the squares are marked randomly, all sorts of patterns can emerge. In this environment two kind of features are identified, facilitators and obstructers. The facilitators support efficient travel whereas the obstructers make it more difficult to travel.

There are three facilitators: gates, bases and corners. The gates provide a passage from one quadrant (the grid is divided in four equally sized quadrants) to another. The bases are locations that are frequently used in routes and could be seen as important nodes. The corners are squares at which a new direction has to be chosen. The bases and corners together create a hierarchy similar to the one that arises from the segmentation process discussed in the Route Knowledge sub-section.

There are four obstructers: corridors, chambers, bottles and barriers. A corridor is a path with the width of one square. The path may have a dead end or emerge in a new area. Chambers are small, irregular, almost confined spaces of several squares and have one access/exit point. Bottles are almost similar to chambers but are not identified during travel but afterwards. Finally barriers are estimations of walls based on an irregular pattern of obstructed squares. The idea is that such a linear approximation cannot be passed, except at one or both ends.

The facilitators and obstructers are used by reasoning processes divided into three tiers or layers, each having several advisors (i.e., functions). The three tiers are consulted in turns from the first to the third and work together to reach a decision. The first tier, however, can veto a move. This seems logical since the first tier represents simple actions that immediately reach the goal or prevent actions that make it almost impossible to reach the goal. The second tier provides advisors that can solve small immediate problems the model faces, such as obstacle avoidance. The third and last tier essentially provides advisors that facilitate commonsense path-finding.

ARIADNE was tested in a random environment of 20x20 squares. It first went through twenty learning runs and then through ten test runs. This was done for four difficulty levels. Of these problems ARIADNE was able to solve about ninety-five percent. ARIADNE was also tested in non-random environments that represent a warehouse, a furnished room and an office space. The warehouse and furnished room categories were all solved, but the office space category presented a challenge. Several problems in the office space were not solved (no percentage was reported in Epstein, 1997).

2.3.5.1 Remarks

It is obvious that ARIADNE is not a cognitive model, nor does it claim to be. It does not use landmarks, but does use the idea of nodes or decision points. Epstein (1997) also mentions that the facilitators and obstructers used in the model represent elements from the real world that humans also use to represent the environment. They are more abstract and general then direct perception. How they are linked to direct perception is however not discussed.

Another similarity can be noted between ARIADNE and humans: several of the advisors in the three tiers represent typical human behaviour. Examples of these advisors are: hurry, which simulates anxiety, adventure, which simulates curiosity, and plod, which simulates tentativeness.

It might be interesting to investigate how the algorithms used in ARIADNE can be used in a plausible way in the cognitive models discussed earlier, since Epstein (1997) claims that the mazes solved by ARIADNE are much harder than those solved by for example, TOUR (Kuipers, 1978), Qualnav (Kuipers & Levitt, 1988) and PLAN (Chown et al., 1995).

2.3.6 Remarks regarding Simulated Navigational Models

Several models have been discussed: TOUR (Kuipers, 1978), NAVIGATOR (Gopal et al., 1989), and PLAN (Chown et al., 1995) which are all cognitive models focused to gain more insight into human spatial cognition, and Qualnav (Kuipers & Levitt, 1988) and ARIADNE (Epstein, 1997), which are more focused on application and have solutions inspired by human spatial cognition. Since the model developed in the current research was implemented on a robot, the next section will discuss some other navigational algorithms that were implemented on a robot.

2.4 Mobile Robot Navigational Models

Some models of Spatial-Learning theory have been used for robots, for example the Qualnav model (Kuipers & Levitt, 1988) discussed in the previous section or the NX Robot (also discussed in: Kuipers & Levitt, 1988). However, these models are both simulations of robots that operate in a simulated environment. This section is about real robots that operate in the real world and deal with problems that come along with it.

Unfortunately, the spatial learning models were not implemented on real robots, but there are models implemented on real robots that show similarities with Spatial-Learning theory. A review of map-based navigation by Filliat and Meyer (2003) identifies two kinds of maps used by mobile robots: topological maps and metric maps. These kind of spatial representations were also identified by Kuipers and Levitt (1988) and are similar to respectively route maps and survey maps (Appleyard, 1970) as described in the Spatial Learning section.

Another distinction made by Filliat and Meyer (2003) is that usually robots are used to *generate* the maps or that the maps are *known* in advance and the robots have to navigate using them. There are also algorithms that do both and they are known as Simultaneous Localization and Mapping (SLAM) algorithms. These algorithms are usually very mathematical and therefore very different from the navigational models based on human spatial learning.

Besides the focus on topological or metric mapping and on map building or map usage there is also a category that focuses on the recognition of elements from the environment. This problem is very hard and usually provides enough of a challenge, without considering further use of the recognized elements.

Since combining perception, mapping and navigation is very hard it could be useful to build models inspired by humans or animals and indeed some models do (e.g., Franz, Schölkopf, Mallot, & Bülthoff, 1998; Smith & Husbands, 2002). The approach used in such models is known as the *behaviour based approach*. However, as mentioned several times in the Simulated Navigational Models section, cognitive models of spatial learning often treat recognition of landmarks as an opaque subject. This is impossible when building models for real robots. Since the recognition of landmarks is always the first step in a cognitive model of spatial learning and the rest of the model heavily depends on it, it is very hard to use these models as a basis for models intended for mobile robots. Robot models, therefore usually have a more pragmatic approach.

To illustrate the approach used in robotics, three models will be discussed (Madhavan, Fregene, & Parker, 2004; Mataric, 1992; Owen & Nehmzow, 1998). These models will also illustrate the problems of perception, map building and map usage. The first two models use topological maps and the third uses a metric map. In addition, the discussion of these mobile robot models will mention some similarities and differences with regard to the Spatial-Learning theory and the models already discussed.

2.4.1 Nomad 200 (Owen & Nehmzow, 1998)

Owen and Nehmzow (1998) have build a navigation system that builds a topological map based on a process of self-organization of the robot's sensory data. They argue for a topological map, since it results in a compact representation in which only distinct locations are stored. Also searching a topological map can be done using proven algorithms like A* and Best-First Search. Also Brooks (1985) argues that topological maps can be more robust than metric maps since they handle noise better, which is a common problem when dealing with robots. Since information is stored locally in topological maps, errors due to noise do not accumulate. Nevertheless there are several

reasons why metric maps are very useful and therefore should be used in combination with topological maps in an ideal scenario.

Topological maps do however have an important problem, which is the problem of *perceptual aliasing*. Perceptual aliasing is the problem that distinct locations have identical sensor readings. These locations can therefore not be distinguished from one another based on sensor readings alone. A solution is to increase the resolution of the used sensors or add a different sensor. Although this approach reduces the perceptual ambiguity, it does not solve it. The solution to this problem used by Owen and Nehmzow (1998) will be discussed later.

As mentioned before, the navigation system uses an approach of self-organization. This has two advantages. One is that the systems itself determines which landmarks it uses, which is useful since it is hard for humans to imagine how the world appears to the robot through its sensors. This is especially hard for humans, since the Nomad 200 only uses its sonar sensors and compass and no vision. Secondly, since the self-organization process uses a clustering technique, which enables generalization over perceptions, robust, noise tolerant landmark detection is possible. It is interesting to note that this approach seems to implement the idea to treat landmark identification as a problem of categorization as discussed in PLAN (Chown et al., 1995).

The clustering technique used is the Restricted Coulomb Energy (RCE) Classifier (Reilly et al., 1982 in Owen & Nehmzow, 1998). The classifier uses a representation vector (R-vector) to represent each class (i.e., landmark) and needs training to determine each R-vector. When an input pattern from the sensors of the robot is presented to the classifier it compares the pattern to each R-vector. The most similar R-vector is labelled "winner" if the similarity falls within a predetermined threshold. If the similarity does not fall within the threshold a new R-vector is created using the input pattern. This means that the nearest neighbour law determines the boundaries between classes.

Before the input pattern is compared to the R-vector it is transformed into an input vector. This transformation transforms the input of sixteen sonar sensors to a normalized input vector. The similarity between this input vector and an R-vector is then determined by the dot product.

Since the input pattern from the sonar sensors depend on the orientation of the robot, compensation is needed for different orientations. The sonar sensors are all located on the top of the robot on a "turret" that is able to turn. The solution is therefore quite simple; the turret is aligned with the compass north. As a result, the sensor input at a certain location is always the same regardless of the orientation of the robot.

Finally a solution must be found for the fluctuating readings of the sonar sensors. The solution to this problem is to use a pass filter. When the readings of the sensors change and the change is above a threshold, they have to be above the threshold for a certain number of time steps to be accepted as a new reading. This filters out small fluctuations over a longer period and large fluctuations over a very small period (i.e., spikes).

The landmarks represented by the R-vectors are stored in a vector map. This map is build in the learning phase in which the robot is led around by an operator. The operator can either let the robot move forward or make it turn. When the perception of the robot changes it takes note of the distance travelled since the previous perception. To decrease the chance a landmark is missed in subsequent visits, a new perception must be "visible" over a minimum distance before it is stored as a landmark in the vector map.

By keeping track of distances and angles, the robot is able to store links between the landmarks that contain the distance between them and the angle between the north

and a virtual line between two landmarks. It also stores the "size" of the landmark as it keeps track of the distance the landmark is "visible". A particular landmark therefore contains its size and a list of connected landmarks containing their identities, distances and angles.

As mentioned before, there is a problem with the use of topological maps, in this case a vector map, namely perceptual aliasing. When a landmark is encountered that has the same readings as a different landmark, no action is taken when it was expected. For example, when a landmark is followed to the next landmark, it is known which next landmark will be encountered. Therefore when an expected landmark is ambiguous, the ambiguity is ignored.

If however the robot encounters an ambiguous landmark that was not expected, it would explore the environment. Exploration in this instance would involve following each link to other landmarks that are known to have a connection to the ambiguous landmark. If all landmarks that are found were expected, the ambiguous landmark was already known and nothing happens. Otherwise the ambiguous landmark is added to the vector map. This solution to the perceptual aliasing problem leads to incremental map building, which is useful for obvious reasons.

Once the vector map is large enough, it can be used to plan new routes. This planning is done by "Best-First Search", which is used to determine the shortest path between current location and goal location. The found path can then be travelled by moving from one landmark to the next. If a landmark was not found, the robot travels back to the previous landmark and plans a new route from there.

2.4.1.1 Remarks

Experiments with the described model were conducted in a laboratory in which a few boxes were placed. The results show that the robot was always able to find its destination goal. In some runs however, the robot did not travel the shortest path because it failed to detect a landmark and had to plan an alternate route.

In some other environment, failing to detect a landmark could be a bigger problem. For example, when there is one chain of landmarks from one area to another in an environment, failing to detect a landmark could not result in an alternative route. In such an example the robot would not be able to complete its task.

A solution might be that the path following algorithm is adapted to switch to a more focused search when the robot fails to detect a landmark. Another possibility is to add vision to the robot. Vision could help guide the robot towards landmarks. Guiding the robot by vision also makes it possible to use a coarser representation for the angles stored in the vector map. The robot would have a rough indication in which direction to travel and could then be guided by vision. With respect to the Spatial-Learning theory, such a solution would seem more plausible.

Also vision, although a very complex modality, would enable the use of landmarks that seem natural to humans. Since humans design a large part of the world, recognizing landmarks, like tables, doors or crossroads, could potentially increase the navigating skills of the robot.

To conclude the discussion of the model implemented on the Nomad 200 robot, it worth mentioning that the algorithm uses some elements from the Spatial-Learning theory and the navigational models discussed. The model, for example, uses the idea of landmarks and the association between them represented in angles and identity. Besides that the model uses declarative knowledge as identified in the Spatial Learning section like, landmarks, nodes and paths. However, the acquisition and usage (i.e., procedural knowledge) of that knowledge is entirely pragmatic as is common in robotics. Also less

plausible with respect to Spatial-Learning theory is the lack of vision, which is something that is essential for a complete cognitive model of spatial learning.

2.4.2 Toto (Mataric, 1992)

The model for the robot Toto developed by Mataric (1992) is similar to the model just discussed, as it also uses landmarks and a topological map. Also, Toto uses sonar sensors and a compass. Despite these similarities there are several interesting differences, which will be discussed in this sub-section.

Toto uses the *subsumption architecture* (Brooks, 1991) as a basis for its model. The subsumption architecture is a hierarchical parallel-layered architecture. The lower layers represent processes of a low level like object avoidance and wall following, and higher layers represent more complex behaviour, like path planning. These layers influence each other, where the higher levels subsume the lower levels. For example, the decision to move to the goal takes into account the decision of the obstacle avoidance layer.

In the case of Toto, there are three layers: basic navigation, landmark detection and map-related computation (map construction, map update, and path planning). By using the distance readings from the sonar sensors, the basic navigation layer implements behaviours like obstacle avoidance and boundary tracing. These behaviours emerge from four basic behaviours:

- *Stroll*: this behaviour causes the robot to safely move forward as it moves the robot backwards when an object in front is approached to a distance considered dangerous.
- *Avoid*: the robot turns in the opposite direction from an obstacle that is too close, yet not so close that it is considered dangerous. If the obstacle is directly in front, the default direction, to the left, is chosen.
- *Align*: if an object is detected in the rear-lateral direction of the robot within a certain distance, the robot makes a small turn in that direction. This causes the direction of travel to align with the direction of wall-like structures.
- *Correct*: straight and convex boundaries can be followed by the using the above three behaviours. To prevent the robot from disorientating when a sharp corner is encountered, the behaviour *correct* is used. It detects these sharp corners by a large difference between two sensors on the each lateral side of the robot. When the front-most sensor of these two returns a large value and the rear-most of the two a relatively small value, a sharp corner is detected and the robot turns in the direction of the detected corner.

Now that the robot can safely wander around by following wall-like structures it can use the second layer, landmark detection, to detect landmarks. In total, there are four landmark types, which follow naturally from the boundary tracing behaviour: left walls, right walls, corridors and a default landmark. The default landmark corresponds to irregular boundaries that cannot be classified as one of the other three landmarks.

Left walls, right walls and corridors are detected by using two confidence counters, one for the left side of the robot and one for the right. Whenever the robot repeatedly detects short readings on a side and the compass reading is stable, the corresponding confidence counter is increased. When one of the confidence counters exceeds a certain threshold a left or right wall is detected and the counter is reset. If both confidence counters exceed the thresholds a corridor is detected. The thresholds represent

the minimum length of the landmarks to be detected and were set to the maximum length of non-landmark obstacles in the environment.

Since one of the four landmarks is always detected the robot detects a series of landmarks that together form a continuous string of boundaries. As a result, one landmark is always followed by another. When comparing this strategy to the Spatial-Learning theory, it seems that landmarks have *become* paths, instead of the landmarks being *connected* by paths. While this might work for indoor environments it is doubtful it would work for outside large-scale environments.

The landmarks are linked together in a topological map, in this case a graph, which is computed by the third layer of Toto's subsumption architecture. Each node of the graph contains four elements: the landmark type, the averaged compass bearing of the landmark, the length of the landmark and a pair of coordinates. The length of a landmark is derived from the times the threshold counter is reset during detection of the same landmark. The pair of coordinates is determined by summing the vectors representing landmarks (the angle and length of a landmark form a vector).

Each time a landmark is detected its four elements are compared to those of the landmarks stored in the graph. If the four elements are the same, it is a match, if not, the landmark is added and a link is formed between the new landmark and the previous one. A margin is used when comparing the distance, compass bearing and coordinates to solve the problem of noise.

Since Toto uses a topological map, it has to deal with the problem of perceptual aliasing. The problem is solved in two ways. First if a landmark was expected based on the topological map (i.e., the landmark corresponds to a landmark connected to the previous landmark) the new landmark is matched to the expected landmark. Second, through the coordinates, each landmark has a rough estimate of position. While two landmarks might have the same landmark type, average compass bearing and length, it is almost impossible for two landmarks to have the same coordinates. The coordinates, together with the expectation, therefore solve the problem of perceptual aliasing.

The third layer of Toto's subsumption architecture also performs path planning. The path planning is done by a variation of activation spreading. The goal node repeatedly sends out a *call* that reaches each node in the network. While the call is spreading through the nodes representing the landmarks, the length of these landmarks is summed up. This way when the call reaches the start node, a rough representation of the spatial distance to the goal is known. The call can reach the start node from all nodes that are connected to it. Based on the estimation of spatial distance, the shortest path is chosen and the robot moves to the selected node. That node again receives a call and the process is repeated until the goal is reached. Note that the spreading activation discussed here is very different from the spreading activation discussed in the ACT-R section.

The spreading activation also makes it possible for the robot to reach the goal if it were placed at some other location, since each node receives the call from the goal node. Also when the robot has tried to reach a node, but fails because, for example, the path has become blocked, the current node can receive the goal's call from another node. After a fixed time period the link between the current node and the one the robot tried to reach is removed and the other node is used to reach the goal.

2.4.2.1 Remarks

A few remarks can be made with respect to the Toto robot. The ability to detect failure and adapt the topological map accordingly makes it possible for Toto to navigate in a

changing world. Also, Toto's first layer makes sure it does not bump into anything and can wander around safely.

Although the map is topological it also contains metric data and in further research a metric map could be derived from the topological map. This metric map could then be used to calculate shortcuts. The metric map can also be better interpreted by humans and other robots, which makes reuse of the map possible.

An important drawback of the landmarks used by Toto is that they only work in indoor environments, since they represent wall-like structures. In an outside large-scale environment the map would probably consist primarily of the default landmark, which represents irregular boundaries. To extend the model to navigate outside, other landmarks would have to be added and also vision. Some benefits of vision have already been mentioned in the remarks regarding the robot of Owen and Nehmzow (1998). Thus, compared to a robot with a model that should be able to perform topological mapping in a real outside environment, Toto is still quite basic.

As mentioned briefly, the landmarks used by Toto differ somewhat from the landmarks described in the Spatial Learning section. The way they are used, however, is similar. Also, the architecture of Toto has some other cognitive plausible components. For example, the basic navigation layer represents the human's basic ability to wander around an environment without getting hurt. Also, planning is a "higher" cognitive process. This is represented by the fact that a higher layer in Toto represents the planning process.

The spreading activation algorithm implemented here is similar to the idea that the spreading activation of the goal node in combination with the start node will result in a recall of the next node along the path towards the goal. This process is similar to NAPS, which is described in the section that discusses PLAN (Chown et al., 1995).

In short, the architecture of Toto shows some similarities with the theory of spatial learning and the navigational models. By adding vision and adapting Toto's architecture, for example the landmarks that Toto uses, Toto could provide a start for a cognitive navigational model implemented on a robot.

2.4.3 Augustus and Theodosius (Madhavan et al., 2004)

While the previous two models discussed use a topological map and a single robot, Madhavan, Fregene and Parker developed an algorithm that builds a metric map, which will be discussed later in this section, by using two robots (Augustus and Theodosius) that cooperate. This model will illustrate the difference between a metric map and a topological map. Also it uses Extended Kalman Filtering (EKF), which is explained later. Although the approach used in this model is quite common in robotics, it differs greatly from the previous two models and the models discussed in the Simulated Navigational Models section.

The two robots used are identical and have several different sensor types: odometry, DGPS, vision, compass, inclinometer and laser rangefinders. These sensors are used to determine the position of the robots in an outdoor environment while mapping it. The goal of the robots is to create a three-dimensional map in which objects and the profile of the terrain are represented. This map is constructed by using a three-dimensional coordinate frame. To be able to build this map the robots must first be able to localize themselves with respect to the coordination frame.

The localization of the robots is done by using EKF. EKF employed for the localization of robots requires two models (Maybeck, 1979 in Madhavan et al., 2004): a kinematic model and a sensor model. The kinematic model uses the odometry sensors of

the robot to determine its forward and angular speed. These speeds are combined with an experimentally determined variance to get an estimate of the robot's location and orientation and a margin of error for these values. Basic laws of physics are used to derive the position and orientation from the forward and angular speed.

The sensor model uses DGPS and compass information to get a second estimate of the robot's location and orientation. The uncertainty of these values is again represented by variances. The variance of the compass is determined experimentally and the uncertainty of the DGPS data is inversely proportional to the number of satellites in view. The second estimate of position and orientation, together with their variances, is used to supplement the first estimate via the predict-observe-validate-update cycle.

In the predict phase of the cycle the kinematic model is used to predict a location and orientation, and the error covariance matrix. To determine the error covariance matrix the values of this matrix from the previous cycle are also used (i.e., it is recursive). In the observe-validate phase the covariance matrix together with the data from the sensor model is used to determine whether the data of the sensor model (i.e., the second estimate) should be used to update the estimates from the kinematic model (i.e., the first estimate). This means for example, that when the uncertainty of the DGPS data with respect to that of the kinematic model is too high, the data is not used to update the estimate of the kinematic model.

The final phase of the cycle, the update phase, calculates the new position and orientation of the robot based on the kinematic model and, if passed by the observe-validate phase, the sensor model. The error covariance matrix is updated too, so that it can be used by the next cycle.

Since Madhavan, Fregene and Parker (2004) use two robots, the relative position of one robot can help estimate the position and orientation of the other robot. By using the robot's camera or laser range finders it is possible to estimate the bearing and distance to the other robot. The other robot also sends it position and orientation by wireless communication. The information received and the bearing and distance to the other robot can then be used in the same way as the data from the sensor model to update the estimation from the kinematic model.

Now that the localization process has been discussed, the mapping process can be described. The mapping process takes place via four main processes. The first step is to determine the distances to several interesting features observed by the camera. The distances to the features are calculated by using the optical flow from the camera. Next, the most interesting feature is selected as the object to move to.

The second step involves moving towards the object. While moving toward the object, the vertical displacement values obtained by using the inclinometer and the DGPS are used to determine the vertical displacement of the robot. The data obtained from both sensors are weighted according to their certainty. For example, if the DGPS has many satellites in view, its measurement is weighted heavier than that of the inclinometer. By combining the vertical displacement of the robot and the data from the camera, a profile of the observed terrain can be determined. Once the robot has reached the object, it avoids it by turning away. The robot then searches for a new object.

During the third step the data obtained upon reaching the object is stored in a terrain matrix, which is the three-dimensional coordinate frame mentioned earlier. If there are small, unknown areas within the data, these areas are filled using cubic interpolation. Objects that form obstacles are represented by extremely high values for the vertical axis of the terrain matrix.

Since the obtained data is stored in a global frame of reference, the maps of the individual robots can be combined to create a larger map of the environment. This is done

in the fourth and final step at a central base station somewhere near the robots. It is possible that several areas have been visited and mapped by both robots and/or more than once by the same robot. When an area is visited multiple times the confidence of such an area in the representation of the mapped terrain is increased.

2.4.3.1 Remarks

Several differences between the first two models and this model can be noted. As mentioned this model uses an absolute metric map, which makes it possible to merge maps from multiple robots. Something that is a lot harder when relative maps or topological maps are used. In absolute metric maps the coordinates of an object indicate which object it is, but other solutions suffer from the symbol-grounding problem.

As we have seen the model uses landmarks to navigate the environment and while doing so, the robot maps it. However, the use of landmarks differs completely from the use described in the previous two models and the models discussed in the Simulated Navigational Models section. The difference is that landmarks are not used to represent a route, but are part of a larger representation of the environment.

Although the metric map created provides a lot of information, it is too much information for path planning. It is only relevant whether there is a path between two locations. It might be useful to know how much effort it costs to travel between the two locations, but there are simpler methods to represent the effort than describing the exact profile of the terrain.

The metric map is also very static, since the map is created at the base station, only when one of the robots has reached an object. As a result the environment might change while the robot is navigating toward a new object. This could be dangerous for the robot.

From a cognitive point of view, especially the detection of features in the environment using the camera is interesting, since that is the bottleneck of most of the models in the Simulated Navigational Models section. The EKF localization method is very powerful, but probably too powerful to resemble human localization capabilities. Other points that are lacking from a cognitive point of view are procedural knowledge and declarative knowledge. Their representations do not resemble those described in the ACT-R, Spatial Learning and Simulated Navigational Models sections at all.

In short, the model described has a very pragmatic low-level approach and similarities between Spatial-Learning theory and the model are based on coincidence. It might be the case that, for example, the image processing and the localization process represent low level processes in the human brain, but that is speculative. However, as already mentioned, the image processing might be interesting to implement in a cognitive model to facilitate reliable visual landmark detection.

2.5 Final Remarks regarding the Theoretical Background

The previous section about mobile robot navigation models is the second last section of this chapter. In this section the most important aspects of the Theoretical Background will be highlighted and some additional remarks will be made.

The chapter started with explaining the relevant components of ACT-R which is necessary for the reader to understand the model developed in the current research. ACT-R provides a framework in which several components of the mind have been combined. Therefore, by using ACT-R, the navigational model inherits important elements, and thereby constraints, of human cognition. The constraints are useful when developing a plausible cognitive model of human route learning. That is why ACT-R is used in the current research.

How humans exactly learn routes has been discussed in the Spatial Learning section following the ACT-R section. Spatial learning takes place in three interwoven stages in which landmark, route and survey knowledge is learned. The most important aspects of the Spatial Learning section are the different strategies of learning a route of which the most important is the paired-associate learning. Almost as important is the discussion about the need for active travel in the process of acquiring accurate spatial knowledge. The importance of active travel also forces researchers to examine the results of several experiments very carefully, since not all include active travel or try to simulate it. Somewhat less important, but also very interesting is the segmentation process that takes place on both the level of route knowledge as that of survey knowledge. Finally the discussion of landmarks can be very important when developing a plausible visual perception component of a navigational model.

The section following the Spatial-Learning theory discussed simulated models implementing that theory. Each model implemented different aspects of spatial learning and none is complete. The first model discussed, TOUR, shows some interesting similarities with ACT-R like chunks and the operations on them. Also NAVIGATOR has implemented aspects of cognition that are also present in ACT-R like activation, decay, and memory structures.

Qualnav demonstrates that common sense from Spatial-Learning theory could be beneficial for robots, but has proven this only in simulation. As with all the simulated navigational models the visual perception component of spatial learning is absent in Qualnav. This visual perception component is probably the most important aspect that the models lack. Once features (e.g., landmarks) from the environment are extracted NAVIGATOR provides interesting mechanisms on how to process these features through their saliency. Besides that, PLAN has specified in detail how directional information of these features could be obtained. In addition, to approach the problem of environmental feature identification as a problem of categorization, as proposed by the PLAN model, could also be very useful.

Storing the features and their properties and using them for learning is adequately modelled in most of the models discussed. Especially the mechanisms of the memory structure of NAVIGATOR together with the hierarchical way of storing information as in the PLAN model could be very useful when building a plausible navigational model, as they simulate learning and forgetting, and segmentation of knowledge respectively.

As a complete model of navigation should be able to move around in the real world, some mobile robot navigational models were discussed in the previous section. They illustrate how Spatial-Learning theory could be used with real robots in a real-world environment. The discussion of such models showed that the models become a lot more pragmatic and that specific mechanisms of cognition (e.g., memory, learning and

forgetting, procedural and declarative knowledge) fade away. They also showed that accurate and useful (visual) perception is hard when it comes to robots.

It is interesting to note that the just mentioned pragmatic shift is not only observed when moving from simulation to real-world robotics, but also when moving from theory to simulation. In the Spatial Learning section many aspects of spatial cognition have been discussed. However when the spatial-knowledge theory is implemented in simulated navigational models, some aspects of spatial learning again seem to fade and be replaced by more pragmatic approaches.

To conclude the theoretical background it is interesting to note that many aspects necessary to build a robot that navigates in the real world just as humans do, are known. Among these aspects is psychological knowledge about how humans acquire spatial knowledge, how this knowledge can be modelled in simulation, and what the capabilities are of real robots. The most important missing element among these aspects is the reliable, plausible visual perception of the real world.

In the next chapter the interface between ACT-R and the AIBO will be described. This interface will be used by a cognitive navigation model, which is described in the chapter following the description of the interface. That model is similar to the models described in the Simulated Navigational Models section, but is, unlike those models, also implemented on a robot.

3. Interfacing AIBO and ACT-R: AIBO-R

As mentioned before, the cognitive architecture used in this project is ACT-R (Anderson, 2005; Anderson et al., 2004). ACT-R by default does not have a way to interact with the AIBO. Therefore additional components are needed that interface the AIBO with ACT-R. One of the components is the Universal Real-time Behaviour Interface (URBI). URBI provides a scripting language that can be used to control complex systems, such as the AIBO. Although URBI shortens the gap between ACT-R and AIBO, it does not close it, because ACT-R cannot directly communicate with URBI. Therefore, ACT-R was expanded with two additional modules referred to as the roboperceptual module and robomotorical module. ACT-R with all the additional components that make it possible for ACT-R to interact with the AIBO will be referred to as the AIBO-R architecture.

Next, a short overview of the AIBO-R architecture will be given. The overview will start with URBI and work its way up to ACT-R. URBI consists of a server part and a client part. The URBI Server runs locally on the AIBO and communicates through wireless LAN with the URBI Client that runs on a PC. Although URBI provides a client, a new client was developed for the current research. The new URBI Client is written in Lisp and provides functions for any Lisp program to send information to and receive information from the URBI Server. A component called URBI Commands was developed to provide several lisp functions that use the URBI Client to interact with the AIBO. URBI Commands is used by the two modules that have been added to ACT-R: roboperceptual and robomotorical. Through these modules an ACT-R model can interact with the AIBO. An overview of the AIBO-R architecture is given in Figure 3.1. After having discussed the AIBO-R architecture in detail, a final section will discuss which processes should be implemented at what level of the AIBO-R architecture.

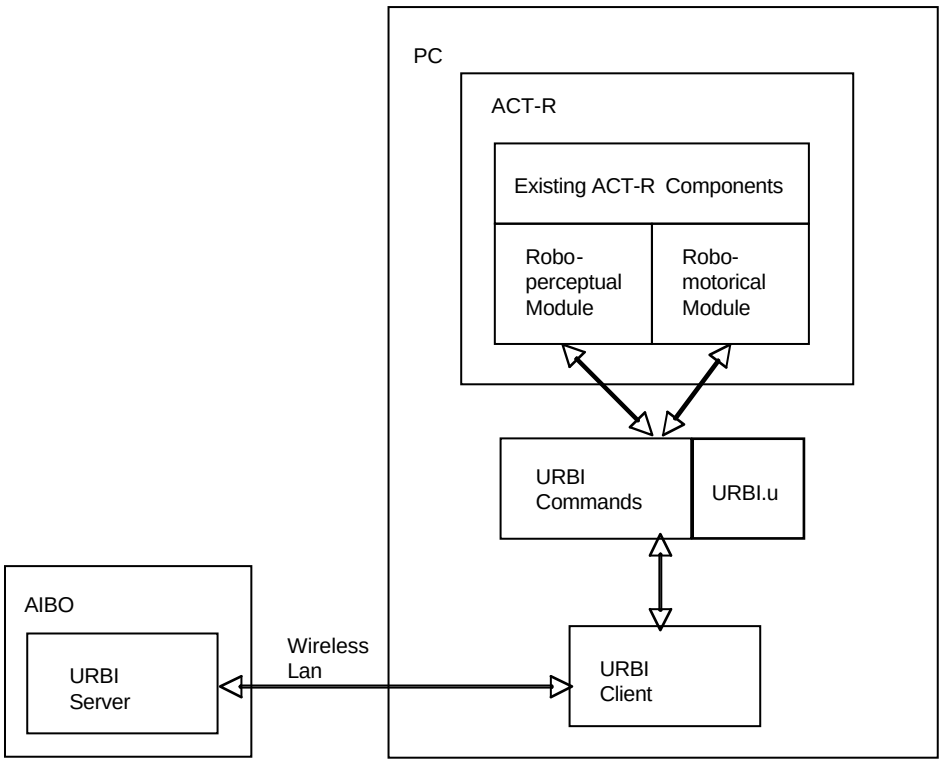

Figure 3.1:
An overview of the AIBO-R architecture.

3.1 URBI

The programming language provided by Sony to program the AIBO is called OPEN-R. In OPEN-R all the joints of the AIBO have to be programmed individually and therefore it requires some effort to program basic behaviours like walking and turning. URBI, however, creates a layer on top of OPEN-R that makes it relatively easy to program complex behaviours (e.g., walking, turning, tracking and searching). This is one of the reasons URBI was used in the current research. Another reason is the client/server architecture of URBI. Such architecture provides a good basis for an interface with a different program, such as ACT-R.

To communicate with the URBI server, the developers of URBI provided a client called URBIlab. However, since ACT-R is written in Lisp, it was decided to implement new URBI client in Lisp as a part of the interface between URBI and ACT-R.

3.2 URBI Client

URBI makes use of a socket connection between server and client. To send data from Lisp to the URBI server a few default Lisp functions are used. However, because of the time delay in the communication between the AIBO and the computer, receiving data proved difficult. Therefore a new function was written that waits until there is something to receive, thereby solving the time delay problem. This function can only be used is if one is sure that there will be something to receive, for example, after a request for information was sent.

The URBI client also provides functions to load the additional URBI functions onto the AIBO. To accomplish this, an external file (i.e., URBI.u) containing the additional URBI code is read and then sent to the URBI server.

In short, when using Lisp the URBI Client can be used to send data to and receive data from the URBI Server. Besides that, it makes it easy to load additional URBI code onto the AIBO.

3.3 URBI Commands

To keep a good overview of the AIBO-R architecture the URBI Commands component was developed. This component defines lisp functions that call specific URBI functions through the URBI Client. Some of these functions already existed in URBI, but a few additional functions were needed to add certain behaviours to the AIBO-R architecture. These additional URBI functions are located in a separate file, URBI.u, which contains all the additional URBI code needed.

URBI Commands therefore depends completely on the contents of URBI.u. If one wants to add additional functionality to the AIBO, one can write a new lisp function using the existing code in URBI.u. If one however wants to add a completely new behaviour that is not a combination of the existing URBI functions, one can add new code to the URBI.u file and use this in the new lisp function. The lisp function can then be used by the roboperceptual module and the robomotorical module.

The functions defined in URBI Commands can for example be used to make the AIBO look around for a certain colour, track a certain colour, walk towards a colour, request sensor information or just walk around. Also, for each behaviour, there is a stop function so that a behaviour can be stopped easily when needed. These stopping functions are needed because ACT-R can start certain behaviours that last for a while. After ACT-R has started such a behaviour, it might receive information that indicates that this behaviour should be stopped. An ACT-R model can then call the stopping functions through the roboperceptual module and/or the robomotorical module.

Besides that, URBI Commands keeps track whether the legs of the AIBO are busy performing any action or that they are free to execute a command. The same applies to the head of the AIBO. URBI Commands also keeps track of angle the AIBO has turned. As a result, it is always known which direction the AIBO is facing.

Now there are several Lisp functions available that can be used by the roboperceptual module and the robomotorical module. The advantage of using URBI Commands, instead of integrating the functions into the modules, is that all functions can easily be tested outside the context of ACT-R. In fact any Lisp program can use the components described so far to interact with the AIBO. Testing at this level ensures that a higher level, for example ACT-R, is not causing a potential problem, keeping debugging manageable.

3.4 Expanding ACT-R

As discussed earlier in the ACT-R section of the theoretical background, ACT-R uses several modules that interact with the central production system through their buffers. The existing ACT-R modules obviously do not provide the possibility to interact with the AIBO. Therefore additional modules and associated buffers are needed. Since the central production system interacts with the modules through chunks in their buffers, the additional modules also need to define new chunk-types.

Two additional modules are created analogously to the visual module and the manual module respectively: the roboperceptual module and the robomotorical module. These new modules inherit the rules for production compilation from the visual and manual module. As a result the production rules using the roboperceptual or robomotorical module compile in the same way as the rules using the visual or manual module respectively. In general that means that those rules do not compile.

The robomotorical module has one buffer, just as the manual module, and just as the visual module (Anderson et al., 2004), the roboperceptual module has two buffers. As mentioned in the ACT-R section the two buffers of the visual module represent the “what” and “where” pathways. The same applies to the buffers of the robovisual module where the robovisual buffer represents the “what” pathway and robovisual-location buffer the “where” pathway. As explained in the ACT-R section, using two buffers makes parallel processing possible. As a result, the robovisual and robovisual-location buffers can operate in parallel, just as the “what” and “where” pathways do.

The idea of using two systems for perception also resembles the strategy used by the PLAN model (Chown et al., 1995) discussed in the Simulated Navigational Models section. More or less analogue to PLAN, the “what” system, represented by the robovisual buffer, enables the acquisition of topological knowledge and the “where” system, represented by the robovisual-location buffer, that of metric knowledge.

Since any number of modules can be added to ACT-R, one might wonder why two modules have been added and not more or just one. Sticking as close as possible to the existing ACT-R architecture, as mentioned before, is one good reason. The other reason is that there are two processes: perceiving the environment and moving around in it. Both processes can operate in parallel, since perceiving is done using the AIBO’s head and walking around by using AIBO’s legs. Processing of production rules in the central production system of ACT-R is a serial process, but a single production rule can match multiple buffers as described in the ACT-R section. Thus by using two modules with their associated buffers it is possible to perceive the environment and at the same time move around in it.

It is interesting to note that the NAVIGATOR (Gopal et al., 1989) model uses two actions types, perception and locomotor, which are used respectively to perceive the

environment and move around in it. This is a similar distinction as the distinction between the roboperceptual and robomotorical modules.

3.4.1 Roboperceptual Module

As just mentioned, there are two roboperceptual buffers that can be used by an ACT-R model. To keep track of the environment around the AIBO, the robovisual-location buffer can be used to request information about the environment. To do this, one can use the following lines in the right-hand side of a production rule:

```
+robovisual-location>
      ISA         observation
      colour      nil / =specificcolour
```

This request fills the buffer with an `observation` chunk that can be used by a subsequent production rule. The `observation` chunk contains several slots that contain information about the environment:

- The distance of an object in front of the AIBO. Two sensors are used for this purpose. One is for close range, 5.7 – 50 centimetres and one is for further away 20 – 150 centimetres. The values of these sensors will be placed in slots called `distance-near` and `distance`, respectively.
- Whether an object is visible or not. If the request for information contained a value for the `colour` slot only objects of that colour are considered. The request would translate as "look for something of colour X". If the `colour` slot does not contain a value (i.e., the value of slot `colour` is `nil`) all objects are considered. Such a request would translate as "look for something". When an object is visible the slot `object-visible` will get the value `t`, otherwise `nil`.
- The colour of the visible object. If the request contained a value for the `colour` slot, this will not change. If `nil`, the colour of the visible object is used for the `colour` slot and if multiple objects are visible the colour of a randomly selected object is used.[1]
- Whether an object of a specific colour is near or not. A combination of the distance sensor and number of visible pixels of the specific colour is used to determine a Boolean value, `t` or `nil`. This value is then stored in the slot called `object-near`.
- The number of visible pixels of the colour of the object that was returned. This is stored in a slot called `pixels`. In the current research this slot is not used. It can, however, be used by other models that use the AIBO-R architecture.
- The x and y degrees of the object of interests centre relative to the centre of the camera image. These values are stored in slots called `x-degree` and `y-degree` respectively. Just as the `pixels` slot, these slots are not used in the current research.

Not all colour values can be used. Only the colours defined in the URBI.u file are available. These colours are defined by creating a colourmap, which is a subspace of the three dimensional YUV colour space. If one adds a different colour by creating a new

[1] Although selecting an object at random is not plausible from a cognitive point of view, it is sufficient for the current research. Which object should be selected if multiple objects are visible is discussed further in the Future Work chapter.

colourmap, the URBI Commands component needs to be expanded to incorporate the new colour, otherwise when searching for any object, objects of the new colour are disregarded.

The visual-location buffer can also be used to determine the relative direction of a perceived object with respect to a previous location and the current location (Figure 3.2). Since humans are not able to determine angles to a one-degree precision, the eight main points of a compass are used, that is: N, NE, E, SE, S, SW, W and NW. This is also in agreement with the bias people have towards straight, forty-five and ninety degree angles as proposed in the PLAN model (Chown et al., 1995).

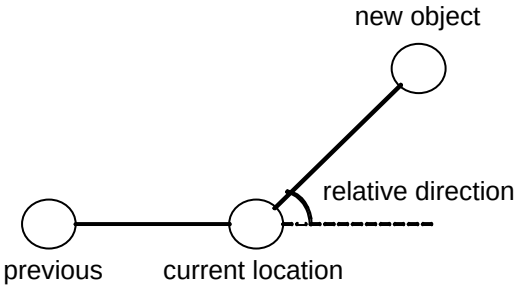

Figure 3.2:
The relative direction of a new object.

The way the direction of objects is obtained is also similar to the way described when discussing PLAN. In AIBO-R, the direction of objects is also derived from the position of the head relative to the body and the position of the body itself. There was, however, no need to analyse the position of the object in the visual field, since the robot is programmed to focus on objects that are encoded. As a result these objects are always in the centre of the field of view.

To determine the direction of a new object and create a `relpos` (relative position) chunk, which represents this information, the following request can be used in the right-hand side of a production rule:

```
+robovisual-location>
      ISA         relpos
      next        =newobjectcolour
      previous    =previous
      current     =currentlocation
```

The robovisual-location buffer is then filled with a new chunk containing the information from the request and one extra slot called `angle`. This slot contains the relative direction of the new object. The new object is stored in the `next` slot, the current object in the `current` slot and the previously visited object in the `previous` slot.

It is important to note that the angle represented, represents the true angle only when the robot is standing in line with the previous location and the current location. This is however not necessary for the `relpos` chunk to fulfil its function. As long as each time the AIBO walks between two specific locations, the second is approached from the same directions as the times before, the `relpos` chunk correctly indicates the direction of the next location. As a result, the `relpos` chunk can even be used in a static world where objects need to be avoided. In a more dynamic world, however, the idea of the `relpos` chunk might fail. Nevertheless, for the model developed in the current research, which is discussed in the next chapter, the `relpos` representation is satisfactory.

Up to now the robovisual-location buffer has been discussed, which can only be used for spatial information. However, if an object of a certain colour is found and its information is located in the robovisual-location buffer, one can use the robovisual buffer to classify the object (i.e., request the “what” information). For example when a red thing is

observed the robovisual buffer can be used to classify this red thing as a waypoint. To do this one can make the following request through the robovisual buffer:

```
+robovisual>
      ISA         object
      colour      =newobjectcolour
```

After the request, the object that is being attended by the AIBO is classified. Ideally a complex pattern recognition algorithm would classify the object and return the class. This class information could then be used to form a chunk representing the object. However such a pattern recognition algorithm is beyond the scope of the current project. Therefore a simplification was made: any object attended must be a waypoint and which waypoint it is, depends only on the colour.

A chunk, which is of the `object` chunk-type, containing colour and class information, is placed in the robovisual buffer. More about the classification of objects can be read in the chapter Future Work.

A few more requests can be made through the robovisual buffer. These requests can be used to make the AIBO look around or track an object. Whether these requests should be handled by the robovisual, robovisual-location or even the robomotorical buffer is open for discussion. On the one hand, moving the head are motor commands, on the other, head movements are closely related to perception. The existing visual module handles all perception, including where to look and where to focus the attention of the model, through the visual buffer. Therefore it seems like a good idea to let the roboperceptual module also control the searching and tracking through the robovisual buffer. Another, more pragmatic reason, is that it is easier to implement and maintain control of searching and tracking behaviour from the roboperceptual module. Finally, within the roboperceptual module the choice has been made for the robovisual buffer instead of the robovisual-location buffer, because it should be possible to make a request for information about the environment through the robovisual-location buffer in parallel to the request of starting or stopping the searching or tracking behaviour.

To start searching, tracking a specific colour or stop either, one can make the following requests through the robovisual buffer respectively:

```
+robovisual>
      ISA         search
```

```
+robovisual>
      ISA         track
      colour      =specificcolour
```

```
+robovisual>
      ISA         stop
```

A conflict does not arise when humans switch between two tasks using the same body part. This is not so trivial when working with robots. If robots are instructed to continuously perform a certain action, they need a specific command to stop. Therefore roboperceptual module automatically stops the previous behaviour when switching between searching and tracking if necessary. For example, when a model is searching for a certain object and the object is perceived upon which the model decides to track it, the model does not have to stop the searching behaviour. Vice versa, when the model is

tracking an object and loses sight of it, it also does not have to stop the tracking behaviour before starting the searching behaviour.

The AIBO searches for objects by slowly moving its head continuously from left to right and back. While the head moves, a model can use the robovisual-location buffer to request information about the environment through the `observation` chunk. The tracking behaviour is a colour blob tracking behaviour. It simply moves the head of the AIBO, and thereby the focus of the camera, towards the centre of a colour blob. Which pixels belong to the colour blob is defined by the colour map discussed earlier.

Finally a few queries can be made to the roboperceptual module through the robovisual buffer. One can check whether the AIBO is currently searching or tracking or is doing both or neither. To do this use the following lines in the left-hand side of a production rule:

```
?robovisual>
      state       searching / tracking / busy / free
```

These queries are useful because the states can be used as an indicator in what phase a model is. This helps building models that rely on the possible values of the `goal` chunk as little as possible. The details of this convention were discussed in the ACT-R section when discussing the "minimal control principle" (Taatgen, 2007).

3.4.2 Robomotorical Module

The robomotorical module can be used to make the AIBO walk around. This is done using the buffer also called robomotorical. The most important request that can be made to this buffer is:

```
+robomotorical>
      ISA         move-to-object
      colour      =colourofobject
```

This request makes the AIBO walk towards an object of the specified colour. The AIBO walks to the object in segments of seventy centimetres and corrects its direction at the beginning of each segment. The AIBO will only start moving if the object is visible. If the AIBO were to lose sight of the object it will stop when it has finished the current segment. On a lower level a fail-safe could be implemented to prevent the AIBO from bumping into things. However since the AIBO moves rather slowly this implementation was omitted. To stop the AIBO moving towards an object through the model, use the request:

```
+robomotorical>
      ISA         stop-move-to-object
```

It is also possible to move the AIBO to a specific location instead of an object. Three slots can be used to determine the destination. These slots are `forward`, `sideways` and `turn`. The first two must be given in meters and the third in degrees. For `turn` it is also possible to use one of the eight main points of a compass as a value. The AIBO moves the amount of meters provided in the `forward` and `sideways` slots. At the same time it will make a turn of the amount of degrees given, relative to the original orientation. A move request would look like this:

```
+robomotorical>
      ISA         walk
      forward     =forwardvalue
      sideways    =sidewaysvalue
      turn        =turnvalue
```

To stop such a move use:

```
+robomotorical>
      ISA         stop-walk
```

If one simply wants to move the AIBO seventy centimetres in a certain direction and also change the orientation towards that direction, the request using `walk-to-angle` can be used:

```
+robomotorical>
      ISA         walk-to-angle
      angle       =anglevalue
```

For the `angle` slot the same values as for the `turn` slot can be used. The `move-to-angle` behaviour can be stopped in the same way as the `walk` behaviour, but `stop-move-to-angle` should be used.

The robomotorical module also keeps track of the state of the legs of the AIBO. One can use a query to check whether the AIBO is using its legs or not:

```
?robomotorical>
      state       busy / free
```

3.5 A discussion of levels

While building an interface between ACT-R and AIBO one has to make choices which component will process what information and execute what actions. One of the obvious choices is that the filtering of camera images is done at a low level by URBI and not by ACT-R. In this section "low level" basically indicates all levels other than ACT-R or its additional modules while "high level" indicates the level of ACT-R.

A more complicated choice is the tracking of an object. Should ACT-R get information of the objects location and then move the head of the AIBO towards the object? Or should this be a low level automated process which ACT-R only needs to start and stop? The choice was made for the latter since there are no conscious choices made about tracking, while busy tracking, except for starting or stopping this behaviour. It is almost like moving an arm from left to right, there is a lot of motor control from the brain, but there is barely any higher cognitive control.

On the other hand if one has to track an object while computing a complex multiplication one would expect a drop in the performance of both tasks. This would mean that tracking an object does imply higher cognitive control. Therefore the question arises "how does such a simple behaviour as tracking an object have an impact on central cognition?" For now, this remains an open question.

There is however also a pragmatic reason to let the tracking of an object be a low level automated process: the time it takes to send information from the AIBO to ACT-R and then for ACT-R to respond with a motor command which is then executed by URBI,

takes too long to let the tracking process be handled by ACT-R. This would result in a very shaky and unreliable tracking behaviour.

Another example of a complicated choice is searching. The behaviour is very similar to tracking when the amount of lower and higher cognitive control is compared. The searching behaviour is currently implemented as a periodic movement of AIBO's head from left to right. However when considering humans searching a certain space, they keep track of where they have searched and where they have not. Also when searching, humans use their experience and knowledge to choose places to search. This would be impossible if searching would be a low level process.

Therefore not only starting and stopping the AIBO's search behaviour should be controlled by ACT-R, as it is now, but also the places where to look. The existing ACT-R visual module already has this possibility, but the roboperceptual module would have to be expanded to include the higher level of control over the searching behaviour. This has not yet been done, as it is beyond the scope of the current.

As mentioned in the Robomotorical Module section, AIBO moves in segments of seventy centimetres. If AIBO loses sight of the object it is walking towards, it stops at the end of the current segment. This is done at a low level by URBI. One could think of a scenario where an object is temporarily hidden because something is passing between the AIBO and the object. ACT-R could reason that this is temporarily and decide to keep moving. In the current situation, where URBI handles this decision, AIBO stops and then moves on as soon as the object becomes visible again. This is however not the way one would expect a human to react. To make it possible for the AIBO to move on in such a scenario, ACT-R should have more control over the walking behaviour than it has now.

In short, when expanding ACT-R to interact with the real world using a robot, many hard choices have to be made. Also there is a strong connection between motor control and information control at a low level and at a high level. If there is one thing the current project has shown, it is that ACT-R is a long way from moving around in the real world as humans do. However, there is no reason to become pessimistic, as for now it certainly seems possible.

4. The AIBO-Route model

Now that the interface between AIBO and ACT-R has been discussed the model that uses the interface can be discussed. As mentioned, the model is a model of the human route-learning process. First a general description of the AIBO-Route model will be given. In the subsequent sections the components mentioned in the general description will be discussed in more detail.

4.1 General Description

In Figure 4.1 an overview of the AIBO-Route model is given. As can be seen from the figure, the model is divided into three horizontal layers. The first layer is called the *Decision Making* layer. In this layer two competing strategies determine how the model continues to the second layer, which is called the *Searching and Processing* layer. The Searching and Processing layer contains processes that search and classify objects. These processes are divided into three groups as indicated by the regions labelled General Search, Processing, and Specific Search. Once an object is classified and determined to be interesting, the model progresses to the third and last layer: *Tracking and Moving*. This layer enables the robot to track the classified object and move to it. Once the object is reached the model returns to the first layer. The process repeats until the goal object (i.e., the destination) is reached.

Of the default modules of ACT-R the AIBO-Route model uses the goal, declarative and imaginal module. Obviously, besides these modules, the model also uses the roboperceptual and robomotorical module. The goal module is used to keep track of the general goal, which is the final destination, and the last two visited locations. The imaginal module is used to hold on to a sub-goal, which is an intermediate destination. The declarative module is used to hold declarative knowledge of the route. The most important chunks stored in the declarative module are of chunk-type `object`, `route-element` and `relpos`. Together these chunks form the building blocks of a route.

A chunk of type `object` has two slots, which hold an object's colour and the class it belongs to. Although the model could reason with any number of classes, only one class is necessary for the AIBO-Route model. More classes could be implemented, but since that would require complex recognition processes, it is beyond the scope of the current project. Therefore, in the current research only the "waypoint" class is implemented. The possibility of multiple classes is discussed in the Future Work chapter. The usage of the `object` chunk-type is discussed in the Processing part of the second layer.

The `route-element` chunk-type represents a route segment as described in the Spatial Learning section. It contains two "route" objects and a goal object, which is the destination of the route. The two "route" objects in the `route-element` chunk represent landmarks between which exists a direct path, that is, it is possible to travel from the first landmark to the second. Similarities between the representation using `route-element` chunks and the NAPS network of PLAN have already been mentioned when discussing PLAN in the Theoretical Background chapter.

Since the classification of objects is simplified and there is only one class (i.e., waypoint), the objects can be represented by their colour. The slots of the `route-element` chunk-type therefore do not contain the objects themselves, but only their colours. Given that the AIBO-R architecture only uses the colour property of an object to search and track it, representing objects by their colour is sufficient.

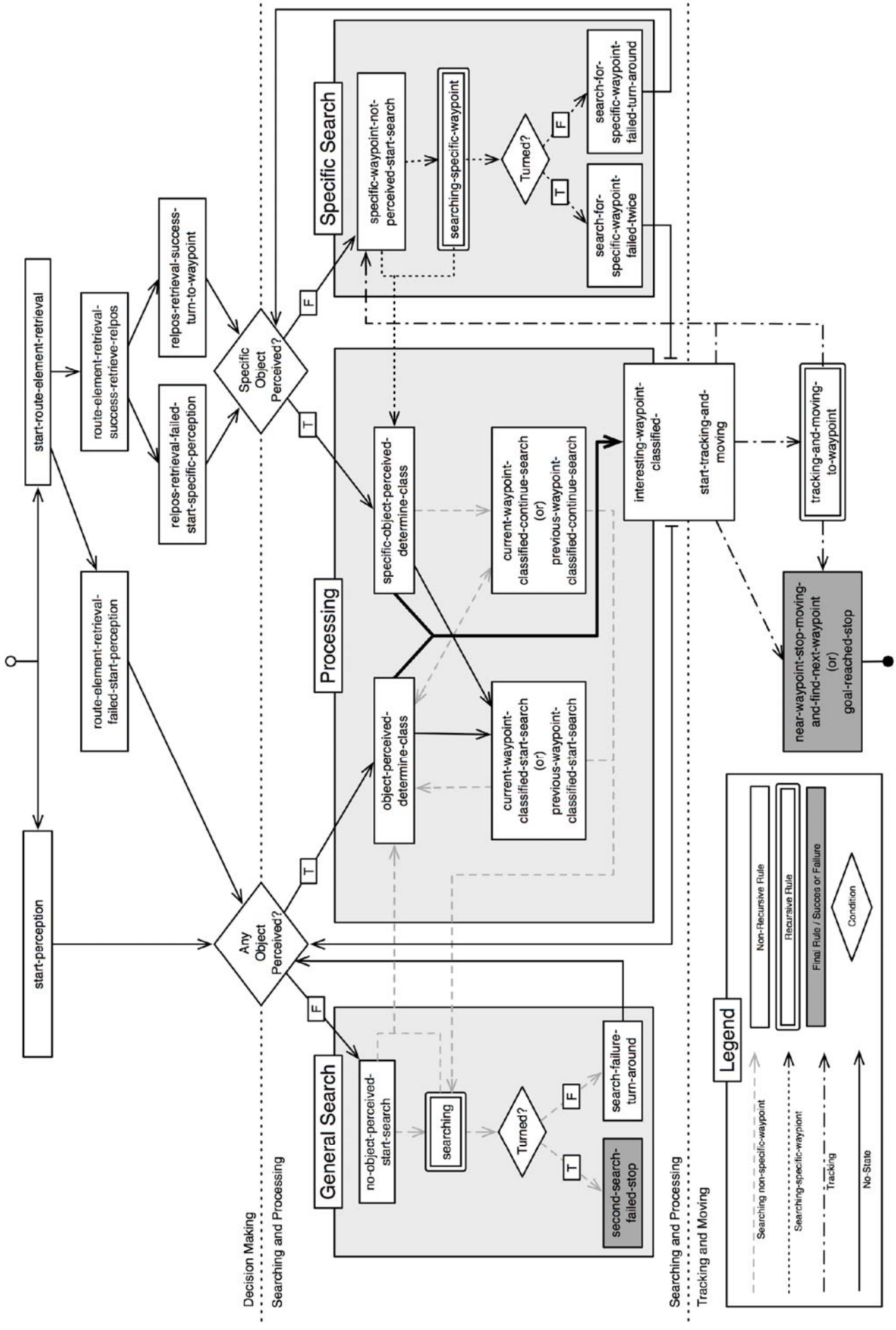

Figure 4.1:
The AIBO-Route model.

Chunks of type `relpos` have already been discussed in the chapter "Interfacing AIBO and ACT-R: AIBO-R" and contain local relative directional information. Also in the Theoretical Background chapter similarities between `relpos` chunks and the R-Net representation of PLAN have already been mentioned. However, at this point an additional remark can be made.

As has been discussed, the `relpos` chunk contains the previous and current location. It also contains a next location and its direction relative to the previous and current location. The previous and current locations together form a route segment. The `relpos` chunk therefore represents an association between a *route segment* and the direction of the next location. This is similar to the association between *landmark* and change of direction described in the Spatial Learning section (paired-associate learning). However, since the perception of a landmark depends on the angle of approach and therefore on one's previous location, the AIBO-Route model uses an association between *route segment* and change of direction.

The `route-element` and `relpos` chunks together form a route. When several routes have been learned the sum of these chunks can be seen as a topological map, which has been discussed in the Theoretical Background chapter. Also it is interesting to note that the `route-element` and `relpos` chunks are very similar to, respectively, the GO-TO and TURN description of the TOUR model (Kuipers, 1978).

The `route-element` and `relpos` chunks are learned in the second and third layer of the model and when learned can be used in the first layer. The three layers will now be discussed in more detail, starting with the first layer.

4.2 Decision Making Layer

In the theoretical background several experiments using slide shows or video presentations have been discussed. In these and also some other experiments, the participant of the experiment is led around an environment. By observing the environment the participant learns the route. Then, after learning the route, when the participant has to navigate it, he or she might not always know what the next landmark along the route is. In such a scenario a backup strategy is needed. This is one reason why the AIBO-Route model is designed with two strategies. The other reason is that the two strategies make it possible to learn a route without leading the model around as is done with the participants of the experiments described.

The first strategy is to simply look for what is out there in the world. This strategy starts with the production rule `start-perception` (see Figure 4.1):

IF the goal is a route and
all other buffers are empty and free
THEN request an `observation` chunk through the robovisual-location buffer

As a result of this rule the robovisual-location buffer will be filled with an `observation` chunk containing information about the environment. The exact content is described in the previous chapter. The content of this chunk will determine whether the Searching and Processing layer continues with searching the environment or classifying an object. If the observation chunk indicates that no object is visible, the model continues searching and if the observation chunk indicates that an object is visible, the model continues classifying the object. The content of the chunk can therefore be seen as a condition on how to proceed and is represented by the condition diamond "Any Object Perceived?" which is shown in Figure 4.1 directly below the `start-perception` rule.

The second strategy is more complex and uses information from declarative memory. The idea behind this strategy is to try to remember the next landmark along a route given the current location. If the model remembers the next landmark it tries to remember its direction and starts moving in that direction. The strategy starts with the rule `start-route-element-retrieval`:

IF the goal is a route and the current location is known and
all other buffers are empty and free
THEN request a route-element chunk of which the next object is not the current object and the goal is the goal of the route in the goal buffer

If the model fails to remember the next landmark, the declarative module will returns an error. In that case the model switches to the first strategy through the rule `route-element-retrieval-failed-start-perception` (see Figure 4.1). If, however, the request succeeds, a `route-element` chunk will be placed in the retrieval buffer and the rule `route-element-retrieval-success-retrieve-relpos` will match:

IF a `route-element` chunk is retrieved
THEN hold the `route-element` as the sub-goal in the imaginal buffer and
request a `relpos` chunk

Since the model knows what specific object is next on the route it can use this information by keeping it in the imaginal buffer to specifically search for this object when it is not immediately perceived. This will be discussed in more detail in the discussion of the second layer.

Just as the retrieval of the `route-element` chunk, the retrieval of the `relpos` chunk either fails or succeeds. If the retrieval fails the model will request an `observation` chunk in the same way as the `start-perception` rule, but this time the request will be limited to observations containing the colour of the retrieved object (i.e., the next object on the route). Depending on the returned information the model continues classifying the next object or starts searching for it. These possibilities are represented by the "Specific Object Perceived?" condition diamond.

If the retrieval of the `relpos` chunk is successful the model can use the information contained in it to move and turn the AIBO in the direction of the next landmark. This is done by the rule `relpos-retrieval-success-turn-to-waypoint`:

IF a `relpos` chunk is retrieved and
the imaginal buffer holds a `route-element` chunk
THEN hold the `route-element` as the sub-goal in the imaginal buffer and
use the `walk-to-angle` command through the robomotorical buffer to turn and move towards the next object and
request an `observation` chunk limited to those holding the colour of the next object

After this rule has fired the model continues in a similar way as after the failure of retrieving a `relpos` chunk, that is, it again depends on the specific object being perceived or not, whether the model continues searching or classifying it. At this point it might be helpful for the reader to look at Figure 4.1 to fully grasp the flow of the model in the first layer.

Since the condition (i.e., *if* part) of the rules `start-perception` and `start-route-element-retrieval` are identical, they are in competition. Both are rules that represent relative general behaviour that is quite common to humans. Therefore, they both have a history of experience, which is simulated by setting the initial successes, failures and

efforts of the production rules to certain values. By doing so, the utilities of these production rules are more or less fixed.

In the ACT-R section the process of product compilation has been discussed. This is the process by which two rules merge into one new rule that has the effect of both. Since some of the production rules belonging to the second strategy can merge into new rules, these new rules add to the already existing competition between `start-perception` and `start-route-element-retrieval`. Which rules can merge in what way is illustrated in Figure 4.2.

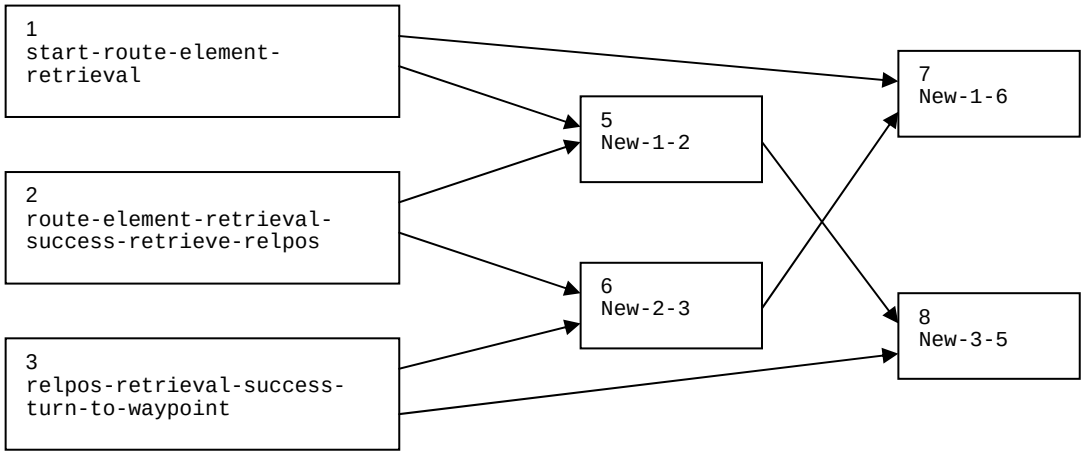

Figure 4.2:
Production compilation in the AIBO-Route model.

Rule one, `start-route-element-retrieval`, is merged into rules five and seven and indirectly, through rule five, into rule eight. Rule five, seven and eight therefore have the same condition (i.e., *if* part) as the `start-route-element-retrieval` rule. Since they have the same condition they, together with the `start-perception` rule, also compete with the `start-route-element-retrieval` rule. Rule six only competes with rule two.

By merging two original rules the chunk that they requested and retrieved is incorporated into a new rule. The compiled rules in the centre, five and six, respectively eliminate the retrieval of a `route-element` chunk and that of a `relpos` chunk. The rules to the right, seven and eight, eliminate the retrieval of both chunk-types and therefore are identical. Since there may be several landmarks along a route, there are also several `route-element` and `relpos` chunks. These chunks are merged into the new production rules and therefore there may be multiple instances of each of the new production rules.

As described in the ACT-R section, of the rules that compete with each other, the rule with the highest utility is chosen. The utility of the original rules, `start-perception` and `start-route-element-retrieval`, is more or less fixed as mentioned earlier. Over time the new production rules can win from the old rules thereby speeding up the deliberation process of the model after having reached a landmark and before moving on to the next. More on the creation and development of the new rules is discussed in the Experiment and Results chapter. The Searching and Processing layer that will be discussed next determines what happens when the Decision Making layer is done.

4.3 Searching and Processing Layer

The Searching and Processing Layer consists of three parts, General Search, Processing, and Specific Search. From the first strategy discussed in the previous section, the model usually progresses to the part in the second layer labelled General Search and incidentally progresses directly to the Processing part. The second strategy causes the model to

continue to either the Processing or Specific Search part. The General Search part will now be discussed first.

4.3.1 General Search

This part of the model consists of four production rules that together are able to search for any object. The searching starts with the rule `no-object-perceived-start-search`:

IF no object is perceived and
the model is not searching
THEN start searching and
request an `observation` chunk holding the colour of any object and
start counting using the temporal module

The temporal module has not yet been discussed as it is not part of the default ACT-R architecture. It has been developed by Taatgen, Van Rijn, and Anderson (2004) to add temporal reasoning to the existing ACT-R architecture. By adding a temporal module and its associated temporal buffer to ACT-R, models using ACT-R are given the possibility to reason about time.

The temporal module is used by the AIBO-Route model because when searching in front of the AIBO, it is possible that no object is found because the object is behind the AIBO. Therefore when a certain amount of time has been spent searching, the AIBO turns around. The searching behaviour is implemented by the `searching` production rule:

IF no object is perceived and
the model is searching
THEN request an `observation` chunk holding the colour of any object and
continue searching

Since the searching behaviour that pans the head of the AIBO from left to right and back, is implemented at a low level in URBI, the model only needs to continuously request an `observation` chunk to gain knowledge about the environment from different directions. The `searching` production rule therefore will keep firing[2], until the counter of the temporal module has reached a certain threshold or an object is perceived. If the threshold is reached, the `search-failure-turn-around` production rule will fire:

IF no object is perceived and
the model is searching and
a certain amount of time has passed
THEN turn around using the `walk` command through the robomotorical buffer and
set the `turned-around` slot of the route chunk in the goal buffer to `t` (true) and
request an `observation` chunk holding the colour of any object and
stop searching

How exactly humans keep track of where they have searched or whether they have turned around, remains an open question. In the AIBO-Route model the pragmatic solution of adding a slot to the goal chunk is chosen, since it does not really increases or decreases the model's plausibility. The models plausibility is not really altered, because for a human it is trivial to remember whether one has turned around to search behind him. Since it is so trivial it does not affect the cognitive workload or mental processing speed. The model

[2] Since the production rule repeatedly fires, it is marked as recursive in Figure 4.1. Instead of continuously fire a production rule, buffer stuffing could also be a solution. In that case the robovisual-location buffer would be filled bottom-up in the event the observation changes. However due to technical constraints (i.e., time delay in the wireless communication) this has not been implemented.

has to keep track of whether it has turned around or not, because otherwise it would keep on turning around. After the model has turned around the rule `no-object-perceived-start-search` will fire again unless an object is perceived.

It is possible that the model still does not find an object after having turned around. In that case a final production rule called `second-search-failed-stop`, which is indicated as a failure to reach the goal, fires:

IF no object is perceived and
the model is searching and
a certain amount of time has passed and
the `turned-around` slot contains `t`
THEN stop searching and
clear the goal buffer

After this rule has fired the model will stop, since the model is lost. It is lost because the model cannot find any objects to which it can travel. Usually, however, the model will find an object in front or behind the AIBO. If an object is found the model switches from the General Search part to the Processing part.

An object might be perceived directly from the Decision Making layer or after the `search-failure-turn-around` rule has fired. In that case, the model was not searching which is indicated by a solid arrow from the "Any Object Perceived?" condition diamond to the `object-perceived-determine-class` rule of the Processing part. An object might also be perceived directly after the rule `no-object-perceived-start-search` has fired or after the `searching` rule fired. If that happens the model also switches to the `object-perceived-determine-class` rule, but keeps on searching too, which is indicated by the dashed arrow pointing from the `searching` and `no-object-perceived-start-search` rules to the `object-perceived-determine-class` rule (see Figure 4.1). The Processing part of the second Layer will now be discussed.

4.3.2 Processing

Basically the Processing part of the second layer classifies objects. When an object has been perceived, its colour is known, but to what class it belongs is still unknown. The class has first to be determined so that it is known exactly at which object the model is focussing. The model can then compare the classified object to the landmark it is currently at and the landmark it has passed before that. These landmarks will be referred to as the current object and the previous object respectively. If the classified object is neither of those, it must be a new landmark and therefore interesting to navigate to.

The classification of an object perceived by the General Search part is started by the `object-perceived-determine-class` rule:

IF an object is perceived and
it is unknown which landmark should be next (i.e., the imaginal buffer is empty) and
the model is not tracking the object
THEN request a classification of the object through the robovisual buffer and
store the relative direction of the object in a `relpos` chunk through the robovisual-location buffer

The roboperceptual module then classifies the perceived object as a waypoint and places an `object` chunk, holding the colour and class of the object, in the visual buffer. The `object` chunk can then be used by subsequent production rules that determine whether the object is the same as the current or previous object or that it is an interesting object.

In the previous sub-section it was described that the model could enter the Processing part either while searching or not searching. When a production rule determines that the classified object is either the current or the previous object and the model is not searching, it should start searching for any object. This is exactly what the rules `current-waypoint-classified-start-search` and `previous-waypoint-classified-start-search` do. The `current-waypoint-classified-start-search rule` looks like:

IF if the classified object is the same as the current object and
the model is not searching
THEN start searching and
request an `observation` chunk holding the colour of any object (general search) and
start counting using the temporal module

The `previous-waypoint-classified-start-search` rule is the same except for that the *if* part compares the classified object to the previous object.

It is also possible that the classified object is matched while the model was still searching. In that case the model should simply continue searching. This is done by the `current-waypoint-classified-continue-search` and `previous-waypoint-classified-continue-search` rules. These rules are similar to the rules that start the search in the current context except that they do not start searching and counting, but only request an `observation` chunk holding the colour of any object.

When the model enters the Processing part through the first strategy of the Decision Making layer or the General Search part of the second layer, it starts with the `object-perceived-determine-class` rule. However if the model enters the Processing part through the second strategy or the Specific Search part, it starts with the `specific-object-perceived-determine-class` rule:

IF an specific object is perceived and
it is known which landmark should be next (i.e., the imaginal buffer contains a `route-element` chunk) and
the model is not tracking the object
THEN request a classification of the object through the robovisual buffer and
store the relative direction of the object in a `relpos` chunk through the robovisual-location buffer and
start counting using the temporal module and
set the `turned-around` slot to `nil`

The *if* part of this rule is different from the `object-perceived-determine-class` rule so that it can match a specific search instead of a general search. Upon a match the *then* part of the rule can then reset the counter of the temporal module and set the `turned-around` slot to `nil`. This is necessary because in the Processing part it is possible to switch from a specific search to a general search when the current or previous landmark is perceived. As a result the search routine, that is whether the model has turned around and the time spent searching, has to be reset before continuing with a general search.

One might wonder how it is possible that the specific object perceived is not an interesting object. The answer is that it is possible due to noise in the perception. At some point the model might mistakenly have perceived an object as another object due to noise.

Finally the classified object could also be determined as interesting in which case the model can stop searching (if it was searching), start tracking the object and navigate towards it. All this is done by the `interesting-waypoint-classified-start-tracking-and-moving` rule:

IF if the classified object is not the same as either the current or the previous object and
the model is not yet tracking
THEN start tracking and
start moving towards the classified object and
place a `route-element` chunk in the imaginal buffer containing the current, the classified and the goal object and
stop counting and reset the temporal module and
set the `turned-around` slot to `nil`

In Figure 4.1 this rule is positioned in the Processing part and halfway between the Searching and Processing, and the Tracking and Moving layer since the *if* part of the rule belongs to the second layer and the *then* part belongs to the third layer. The third layer will be discussed after the Specific Search part of the second layer has been discussed.

4.3.3 Specific Search

The Specific Search part is very similar to the General Search part, something that also can be derived from the symmetrical appearance of these two parts in Figure 4.1. However, obviously, the Specific Search part searches for a specific object. The representation of the specific object (i.e., the colour) is part of the `route-element` chunk present in the imaginal buffer as a result of the second strategy of the Decision Making layer as has been mentioned before. The advantage of searching for a specific object is that other objects are ignored and therefore do not need to be classified. The model "filters out" the other objects, which is more efficient than considering each perceived object. One might consider this top-down control on the search process.

Just as the General Search, the Specific Search also first searches in front of the AIBO and after a while behind the AIBO. However, when neither in front of the AIBO nor behind the AIBO the specific object is perceived, the model should switch to the General Search part to find another object to navigate to. This is done by the rule `search-for-specific-waypoint-failed-twice`:

IF the specific object is not perceived and
the model is searching and
a certain amount of time has passed and
the `turned-around` slot contains `t`
THEN stop searching and
request an `observation` chunk holding the colour of any object and
set the `turned-around` slot to `nil`

After this rule has fired the model either perceives any object or it does not and starts searching for any object. This transition is indicated by the relatively long arrow from the `search-for-specific-waypoint-failed-twice` rule to the "Any Object Perceived?" condition diamond shown in Figure 4.1.

The last difference between the General Search and the Specific Search is that when an object is found it progresses to the `specific-object-perceived-determine-class` rule instead of the `object-perceived-determine-class` rule. The reason for this difference has already been discussed in the Processing sub-section.

The Specific Search was the last part of the second layer to be discussed. The third and final layer will be discussed next.

4.4 Tracking and Moving Layer

Through the `interesting-waypoint-classified-start-tracking-and-moving` rule, the model progresses from the second layer to the third layer. The rule already started tracking the classified object and moving towards it. During tracking the AIBO-Route model has to keep track of the object, but also observe whether it has reached the object or not. Therefore there is a rule called `tracking-and-moving-to-waypoint` that constantly observes the environment and therefore the object. This rule is similar to the `searching-specific-waypoint`, but instead of continuously firing while searching until the object is found, the `tracking-and-moving-to-waypoint` continuously fires[3] until the model loses sight of the object or has reached it. The rule looks like this:

IF the specific object is perceived and
the model is tracking and
the object is not near (i.e., the `object-near` slot is `nil`)
THEN request an `observation` chunk holding the specific colour of the object and
keep track of the specific object through a `route-element` chunk in the imaginal buffer

If the AIBO loses sight of the object in the third layer of the model it switches back to the Specific Search part of the second layer. The model then attempts to find the lost object again, and if found continues to the Processing Part and then again starts tracking the object and moving towards it. It is also possible, mainly due to noise, that the model immediately loses sight of the object after the `interesting-waypoint-classified-start-tracking-and-moving` rule has fired. In both cases, what happens next is indicated by a dot-dashed arrow from the `interesting-waypoint-classified-start-tracking-and-moving` and `tracking-and-moving-to-waypoint` rule to the `specific-waypoint-not-perceived-start-search` rule shown in Figure 4.1.

Whether the AIBO-Route model has reached the object or not is determined by the `object-near` slot of the observation chunk described in the previous chapter. If the `object-near` slot is `t`, it means that the AIBO has reached the object, otherwise the slot is `nil`. When the AIBO has reached the object there are two possible rules that can fire. One rule fires when the sub-goal, that is an intermediate object along the route, has been reached. The other rule fires if the reached object is the goal of the route (i.e., the final object along the route). The rule that fires when a sub-goal has been reached is called `near-waypoint-stop-moving-and-find-next-waypoint` and is defined as:

IF the specific object is perceived and
the specific object is not the goal object of the route and
the model is tracking and
the object is near (i.e., the `object-near` slot is t)
THEN stop tracking and
stop moving and
set the current object in the goal chunk as the previous object in the goal chunk and
set the reached object as the current object in the goal chunk

Since the reached object and the last object passed are set as the current object and the previous object respectively, the model "knows" where it just was. This information can be used in the Processing part of the second layer to determine whether a new interesting object is classified or the current or previous object.

[3] See footnote 2 earlier on page 62.

After the `near-waypoint-stop-moving-and-find-next-waypoint` rule the model progresses from the third layer back to the first layer to start a new cycle to find and move to the next object. This cycle starts as described in the Decision Making layer with one of two strategies.

If the model has reached the goal of the route, the rule `goal-reached-stop` fires:

IF the specific object is perceived and
the specific object is the goal object of the route and
the model is tracking and
the object is near (i.e., the `object-near` slot is t)
THEN stop tracking and
stop moving and
clear the goal buffer

This rule is marked as a success, which indicates that the model has successfully completed its task.

4.5 Summary and Description of Running the Model

Now that the AIBO-Route model has been discussed in detail, a brief summary describing how the model works can be given. The model starts with no declarative knowledge except for the goal object of a route it has to learn. Therefore in the first runs of the model it usually uses the first strategy, which is to search around for something to move to. When doing so, the model learns connections between objects and stores these in declarative memory as `route-element` chunks. Also, when observing a new object, it stores the relative direction of that object in a `relpos` chunk.

When searching around the model might turn the robot around when no object is perceived in front of the robot. Once an object is perceived the robot classifies the object. If the object is classified as interesting it starts moving towards it, otherwise the object is ignored and the model continues searching for another object. The process of searching for an interesting object and moving towards it repeats until the interesting object is the goal object of the route. In that case the model has finished its run.

After a few runs the model has learned sufficient route knowledge. This means that there are `route-element` and `relpos` chunks which are above the retrieval threshold (with or without noise). Since the production rules representing the two strategies have the same fixed utility there is a fifty percent chance due to noise that the second strategy is chosen. If that happens, the model remembers which object it should move to and possibly in which direction that object is. If the direction is remembered the models starts moving and searching for the specific object, regardless of whether it was perceived or not. Otherwise the model does not move and just start searching for the specific object.

Once the object has been perceived it is again classified and then the model either starts moving, in case it did not already move, or corrects its direction of movement towards the perceived object. It might be the case that the model cannot find the specific object even when having turned around. If that happens the model starts searching for any object it can find.

For each route segment either the first or the second strategy is used. Eventually the goal object is reached and the run is complete. As mentioned in the first few runs the first strategy is primarily used. However when the second strategy is used, its production rules compile into new production rules, which do not have a fixed utility. After a certain number of runs those utilities will become higher than those of the original production rules. This causes the second strategy to be used more often, since the compiled production rules represent the second strategy. When the route is walked using primarily

rules of type seven and eight from Figure 4.2, the route is thoroughly learned as procedural knowledge.

The learning of chunks and compilation of production rules described in this section will be discussed in detail in the Experiment and Results chapter.

5. Experiment and Results

To find out whether the AIBO-Route model meets the two sub-goals of this project, the model was tested in a robot lab. The two sub-goals are repeated below:

1. Given a setup of several landmarks the AIBO-Route model should be able to learn a route to a predefined goal.
2. When having learned such a route and the environment changes in such a way that a shorter route is possible, AIBO-Route should be able to learn the new shorter route.

The experiment performed in the robot lab consisted of two phases. In Phase 1 the model was tested for the first sub-goal and in Phase 2 for the second. To test the model for the first goal, the experimental setup illustrated in Figure 5.1 was used. After the model had learned the route of that setup, the setup was changed to test the model for the second goal in Phase 2. The changed setup is illustrated in Figure 5.2. For the model to learn, several runs were needed, where in each run the AIBO moves from START to the goal landmark via a certain path. Next, the experiment will be discussed in detail, followed by its results.

5.1 The Experiment

As mentioned above, the experiment was performed in a robot lab. Since the camera of the AIBO is sensitive to changes in lighting conditions, the robot lab was illuminated using only fluorescent lightning thereby minimizing variations in visual perception.

5.1.1 Phase 1

The setup used in Phase 1 consisted of a START point and four landmarks (see Figure 5.1). The landmarks are constructed of four differently coloured cardboard cylinders with a height of 50 cm and a diameter of 20 cm. The landmarks each have a different colour, which enables the model to uniquely identify each landmark. The pink landmark, landmark D, is set as the goal of the route the model has to learn.

A wall of cardboard boxes, indicated by the rectangles in Figure 5.1, is used to limit the number of landmarks the model can perceive. As a result, AIBO can perceive exactly one landmark it has not yet visited from each landmark or the START point. Note that although landmark C might be visible from landmark A, the AIBO does not stop exactly at landmark A, but a little bit earlier. Also, the AIBO is facing away from landmark C upon reaching landmark A. Therefore AIBO does not perceive landmark C from landmark A. The same line of reasoning applies to landmark B and D.

A different setup could have been used such that multiple landmarks are visible from one of the nodes, but the AIBO-Route model was not designed to handle multiple visible landmarks other than to choose one at random. Furthermore, the model only remembers the last two landmarks visited. As a result, if two landmarks are visible at the same time, the model might travel back to a landmark it has already visited. If the model is able to learn a route in such a specific setup, it could be modified in the future to work with different, more complex setups. Therefore to test whether the model is able to learn a route, the setup illustrated in Figure 5.1 where at most one unvisited landmark is visible was used in the experiment.

The idea behind the setup is that the AIBO is forced to always walk the route in the same order, START-A-B-C-D, as illustrated by the arrows in Figure 5.1. While

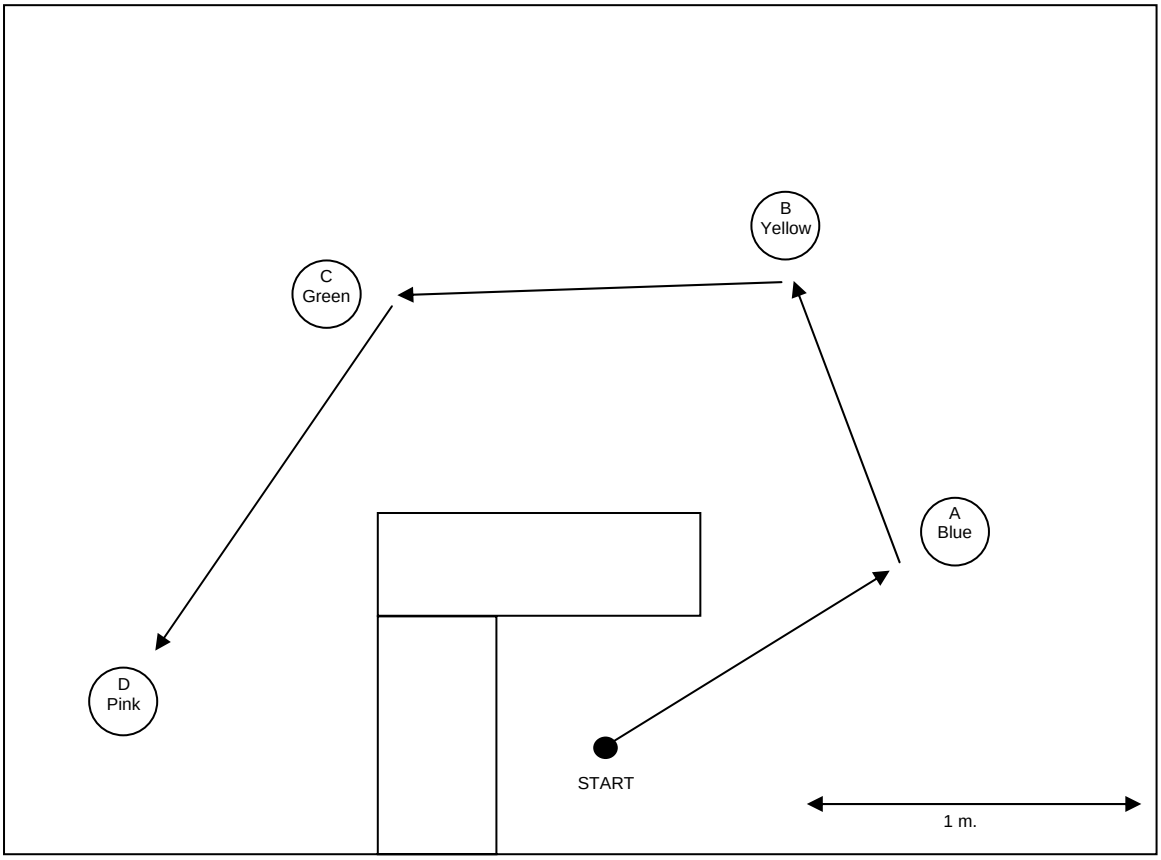

Figure 5.1:
The setup in Phase 1 of the experiment.

walking that route, the model should be able to learn the order of and relative directions between the landmarks. Since the model was developed to use those directions to start walking towards a landmark before having perceived it, the model saves time because it does not have to search before moving towards that landmark. As a result, the performance of the model should improve over time, because less time is needed to reach the goal. Whether the AIBO-Route model learns the route is therefore not perceived by the order in which the model visits the landmarks, but by the change in behaviour. Of course, learning can also be determined by examining the declarative and procedural memory that cause the changing behaviour.

5.1.2 Phase 2

After having learned the route, which is the end of Phase 1, the setup of the environment was changed as illustrated in Figure 5.2. As shown in the figure, the first two landmarks have been removed as well as part of the wall. As a result, the AIBO is able to directly perceive the third landmark, landmark C, from its start position. The setup should cause the model to forget the route learned in Phase 1 and start learning a new shorter route START-C-D.

Since the model should have learned the route START-A-B-C-D in Phase 1 and therefore knows the relative directions of the landmarks, it should start moving from START to A without first searching for landmark A. While moving, the model will try to find the landmark, but it will fail because landmark A is no longer there. As a result the model stops moving, which will be near point X shown in Figure 5.2, because the model moves in segments of seventy centimetres. After having stopped, the model continues searching for landmark A and after a while starts searching for any other landmark. The

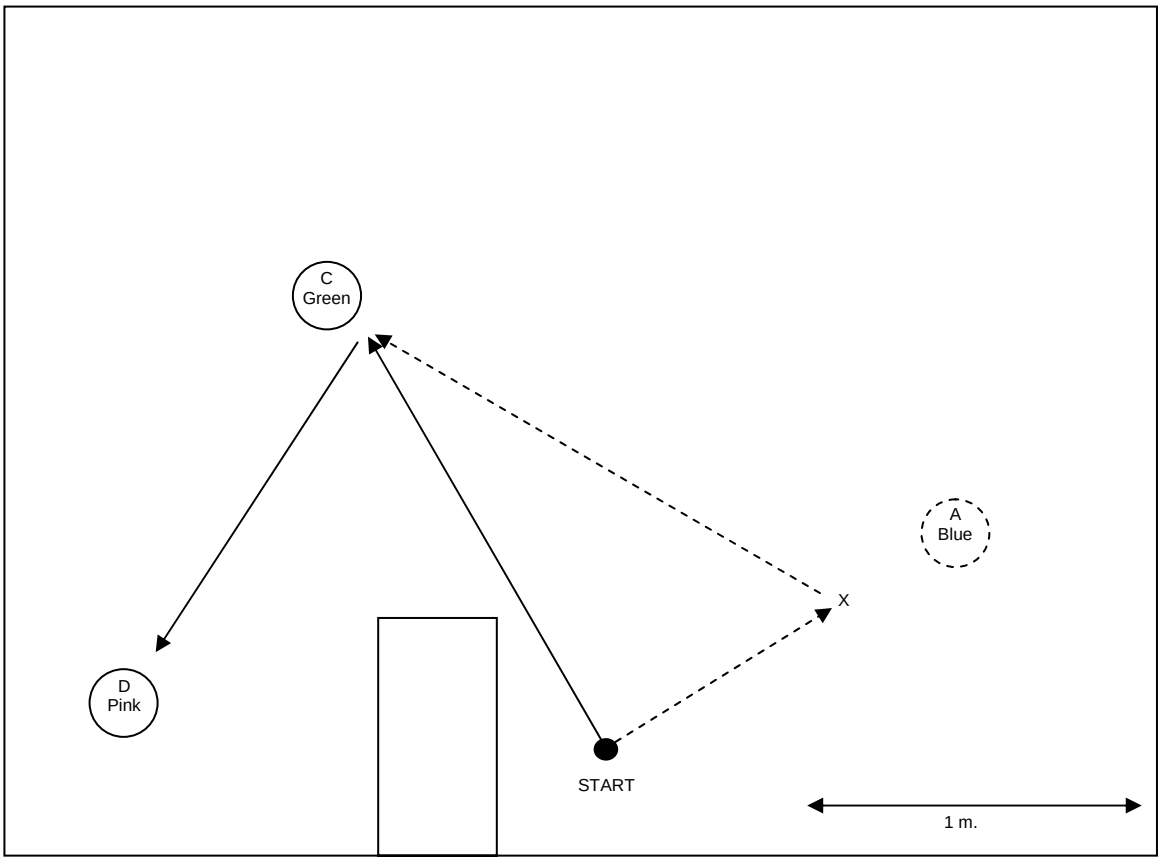

Figure 5.2:
The setup in Phase 2 of the experiment.

model should find and then move to landmark C, because that is the only one visible from point X. Finally, the AIBO-Route model will reach landmark D via landmark C. The path just described is marked with the dashed arrows followed by a solid arrow from landmark C to D.

The AIBO-Route model will only start moving towards landmark A if it uses the second strategy described in the previous chapter. If the second strategy is used, the model tries to remember the next node and its direction and starts moving towards it. However, there is a chance that the AIBO-Route model will use the first strategy, which is to look around for any landmark to move to. If the first strategy is used, the model should perceive landmark C from START and start moving towards it, which results in the path illustrated by the solid arrows in Figure 5.2. Just as the AIBO-Route model learned route START-A-B-C-D in Phase 1 of the experiment, it should eventually learn START-C-D in Phase 2 of the experiment.

In the next section the results of running the AIBO-Route model in the two described setups will be discussed. Since the model was developed to meet the expectations just discussed, it is not surprising that the results match those expectations very closely.

5.2 Results

In this section the results of the experiment will be discussed. First the results of Phase 1 will be discussed and then those of Phase 2. The description of both phases will start with a general description of the observed behaviour, followed by a detailed description of the learned declarative and procedural knowledge.

5.2.1 Phase 1

By examining the data, that is the activations of chunks and utilities of original and new production rules, the learning of Phase 1 was determined to be complete after run 25. The figures referring to Phase 1 will therefore only show the first 25 runs of the total of 85 runs.

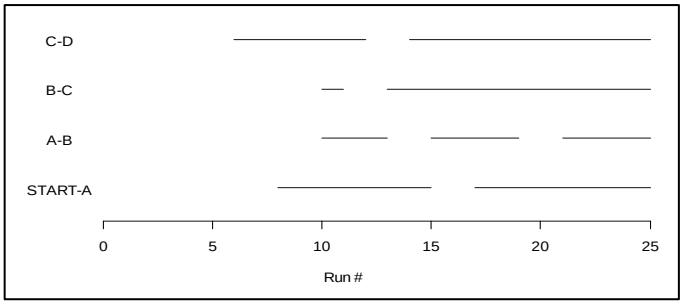

Figure 5.3:
The remembered directions for each run in Phase 1.

As expected, the AIBO-Route model walked the route START-A-B-C-D indicated by the solid arrows in Figure 5.1. In the first few runs the model had to search at each node for the next landmark. In run 5, however, the model remembered the direction of a landmark for the first time. As a result, the model started moving from landmark B to C, without first searching for landmark C. It is important to note that remembering the directions of landmarks in this context can be because of declarative memory, but also because of procedural memory. In the next few runs the model started to remember the directions of the other landmarks as well and in run 8 the model remembered the directions of all landmarks in a single run for the first time. In the subsequent runs, the model remembers the direction of at least three landmarks and in the final five runs of Phase 1, the model remembers the directions of all landmarks in each run. The results just described are illustrated in Figure 5.3. The lines indicate whether the direction of a landmark was remembered. For example, at run 10 the line START-A is present, which indicates that the model remembered the direction of landmark A at START in run 10.

For the model to remember the direction of a landmark it needs to apply the second strategy. However, by default, there is a fifty percent chance that the model will apply the second strategy, because either the `start-perception` rule (first strategy) or the `start-route-element-retrieval` rule (second strategy) will fire and their utilities are fixed at the same level because they both have a similar history of experience (see section 4.2). Since the results shown in Figure 5.3 indicate that chance for the model to remember the direction of landmark becomes higher than fifty percent, additional production rules must have been learned that compete with the original production rules.

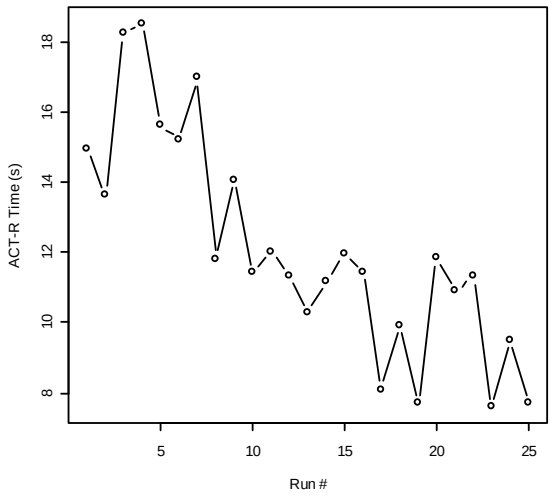

Figure 5.4:
The runtimes in ACT-R Time for Phase 1.

These new production rules, as described in the previous chapter, eliminate a retrieval (i.e., the need to remember a certain fact), which speeds up the model. This, in combination with remembering the direction of a landmark, causes the duration of each run (i.e., runtime) to decrease. The runtimes in ACT-R time for each run of Phase 1 are illustrated in Figure 5.4.

The ACT-R time is based on the internal clock of ACT-R, which it uses to calculate retrieval times and the durations of other cognitive processes. However, the

temporal aspects of the cognitive processes handled by the modules developed in the current research (i.e., roboperceptual and robomotorical) are not implemented. Therefore, the temporal data obtained by the experiment is meaningless from a cognitive point of view. However, the duration of a run in ACT-R time is a reliable measure for the duration of a run in real time and is therefore illustrated in Figure 5.4 to illustrate the increase of the model's performance.

One might ask why there is a relatively high variation in the runtimes. The answer to that question is: noise. The noise is primarily due to simultaneously moving and perceiving the environment. The movement of the AIBO results in an unsteady video feed that sometimes causes the model to poorly judge a situation. As a result, for example, the AIBO not always stops at the same point upon reaching a landmark. Stopping too close to a landmark results in a longer path and also causes the AIBO to bump into the landmark, both causing the runtime to increase.

Now that some general observations about Phase 1 of the experiment have been discussed, the learned declarative knowledge can be discussed.

5.2.1.1 Declarative Knowledge

As mentioned in The Experiment section, the order of landmarks that can be learned is fixed as a result of the setup illustrated in Figure 5.1. Therefore it is not surprising that the model has learned four `route-element` chunks that represent the four route segments of the route START-A-B-C-D. Remember that a `route-element` chunk has three slots that represent the goal of the route and two connected landmarks. As values for the `previous` and `next` slot, the four chunks respectively have the values: `START-A`, `A-B`, `B-C` and `C-D`. All four chunks have landmark D as a value for the `goal` slot. The activations of the four chunks are illustrated in Figure 5.5.

Since the activations illustrated are the activations values after a run has been completed and chunks used earlier in the route have more time to decay before the goal is

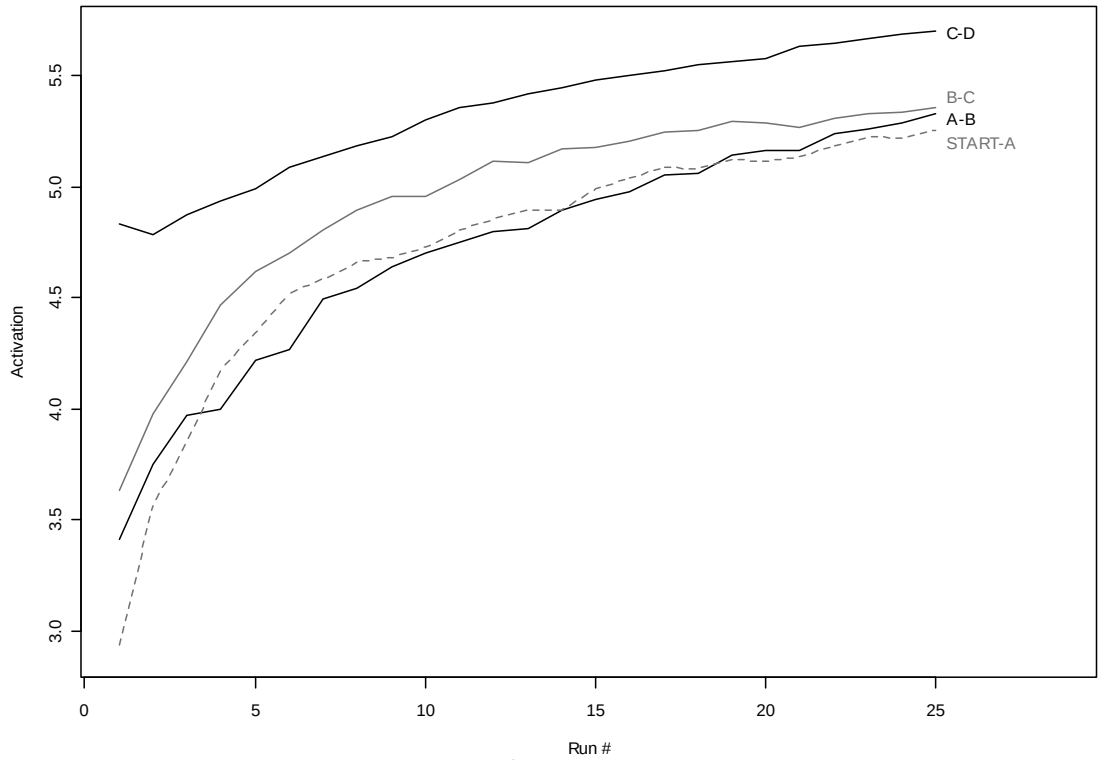

Figure 5.5:
The activations of the `route-element` chunks in Phase 1.

reached than those used later in the route, there is a difference in the activation levels of the four chunks. For example, chunk `A-B` is not used after landmark B and therefore decays more before the end of a run than chunk `C-D`, which is used until the goal is reached. As a result, the activation of chunk `A-B` measured after a run has been completed is lower than the chunk `C-D`.

Besides the order of the landmarks, the model has also learned their relative directions which are stored in `relpos` chunks. The activations of these chunks are illustrated in Figure 5.6. At the end of each curve there is label that represents the information stored in the chunk belonging to that curve. For example, the label "`START-A < B: W`" represents a chunk holding the start position in its `previous` slot, landmark A in its `current` slot, landmark B in its `next` slot, and the relative direction west in its `angle` slot. This information indicates that landmark B is to the west with respect to line connecting START and landmark A. In other words: if the line START-A indicates north, then landmark B is to the west.

Note that not all `relpos` chunks are illustrated in Figure 5.6. First, the figure only contains `relpos` chunks learned during Phase 1 of the experiment. Second, only relevant `relpos` chunks are illustrated. For example, the model also learns the relative direction of landmark B with respect to the line that connects landmark A and B. Since the model ignores landmarks that are the same as the previous or current landmark, only `relpos` chunks representing relative directions of landmarks that are not ignored are illustrated in Figure 5.6.

In Figure 5.7 a graphical representation of the `relpos` chunks illustrated in Figure 5.6 is given. The arrows originating from a landmark indicate the relative direction of the next landmark. Thus the arrows originating from landmark A indicate the direction of landmark B. The three types of arrows used in the figure indicate an activation level. This activation level is the activation of the `relpos` chunks at the end of Phase 1. A thick arrow represents the `relpos` chunk that has the highest activation with respect to the `relpos` chunks that represent a direction to the same landmark. The solid arrows represent `relpos` chunks that are above the retrieval threshold and the dashed arrows represent those that are below the retrieval threshold.

The relative direction is derived from the direction of AIBO's body. This direction is maintained via dead reckoning, which is not very robust. As a result there are multiple `relpos` chunks, which contain the same landmarks, but different angles. As described in the ACT-R section, of chunks that match a request, the chunk with the highest activation is retrieved. Therefore the chunk with the highest activation should represent the most accurate relative direction. As can be seen from Figure 5.7, the `relpos` chunks with a higher activation are indeed more accurate. It is interesting to note that the gradual build up of correct relative directional knowledge seems to fit a theory of skill acquisition referred to as *instance theory* or *instance learning* (Logan, 1988).

The arrows in Figure 5.7 originate from the centre of the landmarks, but the AIBO never perceives the next landmark from exactly that position. Therefore, the arrows should originate from the position of AIBO's head at the moment it sees the next interesting landmark. Also the direction of the arrows should be drawn with respect to the orientation of the AIBO's body at that moment. However, since the position and orientation varies slightly with each run, the centre of a landmark is used. Although Figure 5.7 is not entirely accurate, it is still an useful illustration of the directions learned at the end of Phase 1.

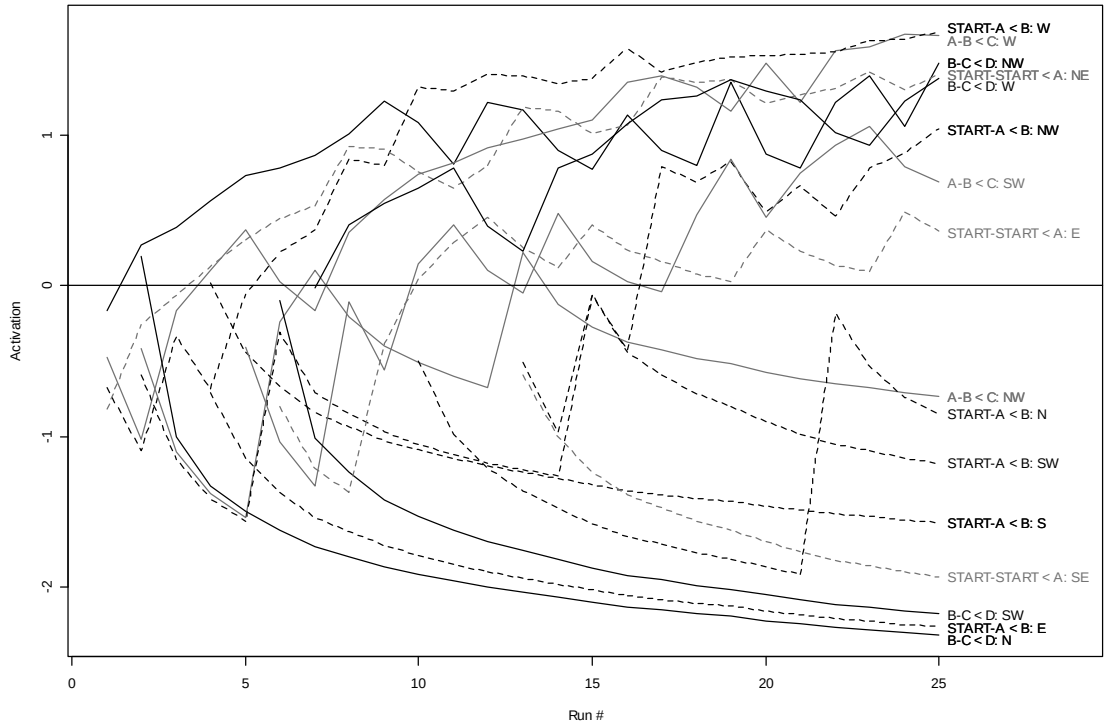

Figure 5.6:
Activations of the `relpos` chunks in Phase 1.

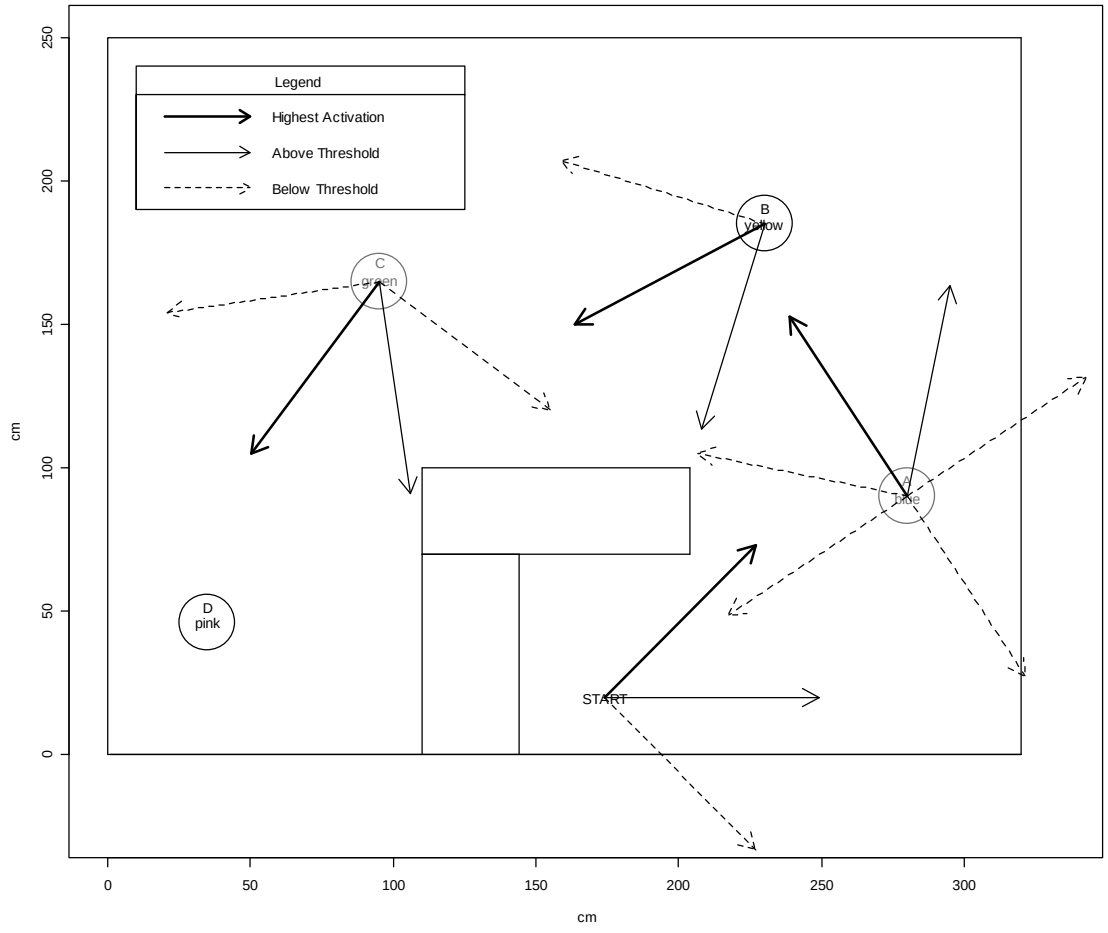

Figure 5.7:
Graphical representation of the activations of the `relpos` chunks after run 25.

The learned declarative knowledge indicates that the route was learned correctly: the model has learned the correct order of the landmarks and also correctly learned their relative directions. Besides this declarative knowledge the model was also designed to learn procedural knowledge, which is discussed next.

5.2.1.2 Procedural Knowledge

As explained in the previous chapter, there are three possible types for new production rules. One type eliminates the retrieval of a route-element chunk and will be called type (1), a second eliminates the retrieval of a relpos chunk and will be called type (2) a third and final type eliminates the retrieval of both chunks and will be called type (3). In Figure 4.2, which illustrates the production compilation process, these types correspond to respectively rules number five, six, and seven or eight.

The utilities of the through production compilation learned production rules are illustrated in Figure 5.8. In the figure there are five different line types. Except for the thick line, each line type indicates a different situation in which the *if* part of the production rule represented by that line, matches that situation. For example, the dashed black lines represent production rules that match the situation where the AIBO has reached landmark A. The thick lines represent the utility of the production rules: `route-element-retrieval-success-retrieve-relpos` (RERSRR), and `start-perception` and `start-route-element-retrieval`. Since these rules are in competition with the new learned rules, the thick lines more or less act as a threshold for those new rules, as will be explained below.

On the right of Figure 5.8 a list of labels is given. Each label is a short representation of what the corresponding production rule stands for. Also each label starts with a number that matches one of the three production rule types just mentioned. Below, examples of these labels are explained for each rule type:

- A label like "`(1) B -> C`" represents a production rule of type (1):

  IF the previous landmark was a landmark X and
  the current landmark is landmark B and
  the goal is landmark D
  THEN the next landmark is landmark C and
  request a `relpos` chunk matching "`current`: B, `next`: C, `previous`: X"

  The "X" in the description represents a variable that can hold any landmark. The `relpos` chunk needs to be requested because the direction of landmark C is still unknown.

- A label like "`(3) B-C < D: NW`" represents a production rule of type (3):

  IF the previous landmark was landmark B and
  the current landmark is landmark C
  THEN the next landmark is landmark D and
  the AIBO moves to the north-west (NW) with respect to its current direction

- A label like "`(2) B-C < D: NW`" represents a rule of type (2):

  IF the previous landmark was landmark B and
  the current landmark is landmark C and
  the `route-element` chunk representing "current: C, next: D, goal: D" is retrieved
  THEN the next landmark is landmark D and
  the AIBO moves to the north-west (NW) with respect to its current direction

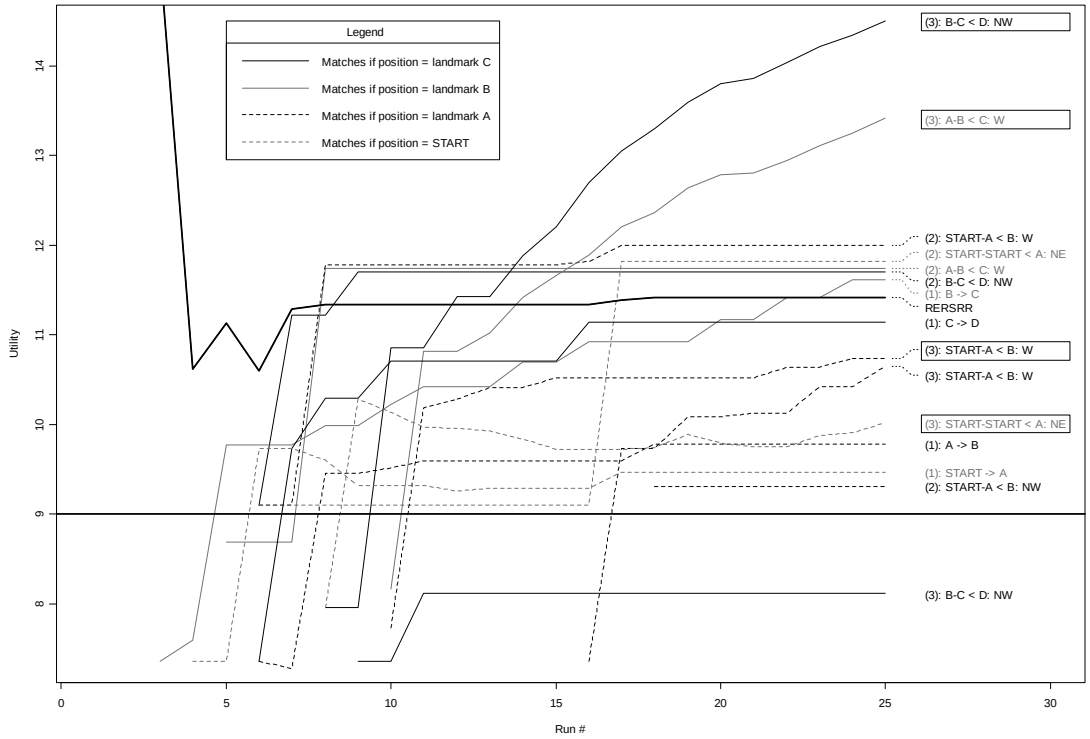

Figure 5.8:
The utilities of the learned production rules in Phase 1.

Note that although a label of type (2) is similar to a label of type (3), a rule of type (2) requires a retrieved `route-element` chunk whereas a rule of type (3) does not. Also it interesting to note that of some rules of type 3 there are two (e.g., `(3) START-A < B: W`), because three production rules can merge into one production rule in two different ways. For example rules A, B and C can merge into AB and BC, which in turn can merge with respectively C and A, both resulting in the rule AC. This is explained in more detail in the previous chapter and also illustrated by Figure 4.2.

Production rules of type (1) and (3) compete with `start-perception` and `start-route-element-retrieval` and therefore must have a higher activation (with added noise) than the latter two if they are to be used. The thick horizontal line that represents the utility level of the last two rules, therefore acts as a threshold for rules of type (1) and (3). Similarly, the thick line representing the utility of the rule `route-element-retrieval-success-retrieve-relpos` (RERSRR) acts like a threshold for rules of type (2).

From Figure 5.8 it is possible to derive the production rule with the highest utility of the production rules that compete at the beginning of a route segment. Since there are four route segments there are also four corresponding rules which are: "`(3) START-START < A: NE`", "`(3) START-A < B: W`", "`(3) A-B < C: W`" and "`(3) B-C < D: NW`". These four rules are marked with a rectangle.

The four production rules have incorporated the four route-element chunks, and the relpos chunks represented by the thick arrows illustrated in Figure 5.7. Since the relpos chunks represented by the thick arrows already had relatively high activations during the first 10 runs, they are usually retrieved instead of other matching chunks during those runs. Because most of the production rules learned in Phase 1 are created before the 10^{th} run, it is not surprising that the four production rules just mentioned have incorporated the chunks represented by the thick arrows. In fact, those chunks are the

only chunks that were used in the production compilation process, with the exception of the chunk "START-A < B: NW", which resulted in the rule "(2): START-A < B: NW". That rule was created in run 18, but was never used again.

The fact that the four production rules represent the correct order and direction of landmarks indicates that the model has correctly learned the route as procedural knowledge. Also taking into consideration the results the declarative memory, Phase 1 of the experiment demonstrates that the AIBO-Route model has correctly learned a route. The results of Phase 1 therefore satisfy its goal.

5.2.2 Phase 2

After Phase 1 of the experiment was complete, the environmental setup was changed to the setup illustrated in Figure 5.2. The AIBO had learned to move from START to landmark A, and therefore kept moving towards the position where landmark A was in Phase 1, even though the setup had changed. Since landmark A was no longer present in Phase 2, the model could not find it and turned around near the position marked with "X" in Figure 5.2. Of course, the model still could not find landmark A and as a result switched to a general search thereby perceiving landmark C from X. After perceiving landmark C, the model started moving towards it and then continued towards landmark D.

The model repeatedly showed the behaviour just described until the utilities of the production rules causing the AIBO to move towards X had decreased sufficiently to allow the rule `start-perception` to fire. The utilities of the production rules causing the AIBO to move to X decrease, because reaching the goal through those rules takes longer in Phase 2 than in Phase 1. When the `start-perception` rule fired, the model immediately started searching for any object from START without first moving towards landmark A. As a result, the AIBO moved from START to landmark C without the detour.

The `start-route-element-retrieval` rule always has the same fixed utility as `start-perception`, because it shares the same history of experience as explained in section 4.2. Since `start-perception` is able to fire and `start-route-element-retrieval` has the same utility, the latter is also able to fire thereby requesting a `route-element` chunk. If the right `route-element` chunk (i.e., `START-C`) was retrieved the model would also directly move to landmark C. However, since the `route-element` chunk `START-A` still had a high activation, it had a higher chance of being retrieved than `START-C`. Therefore, if the `start-route-element-retrieval` rule were to retrieve the route-element chunk representing `START-C`, that chunk first had to gain a higher activation.

Since in Phase 2 landmark C is the first landmark the model can perceive from START, the activation of the chunk `START-C` is increased each time the model perceives landmark C. Also the activation of `START-A` decays each time the model starts with the rule `start-perception`, because in that case `START-A` is not used. After a few more runs, when the rule `start-route-element-retrieval` fires, the model is able to retrieve the chunk `START-C`, because its activation has come close to that of `START-A` and with added noise can surpass the activation of `START-A`. This occurred for the first time in run 42.

At that point, by merging `start-route-element-retrieval`, `route-element-retrieval-success-retrieve-relpos` and chunk `START-C`, a new production rule of type (1) is created which represents that after node START, landmark C should follow (see "(1): START -> C" in Figure 5.13). In addition to that rule, two rules of type (2) are created. One rule represents that landmark C with respect to START is to the north-west (i.e., "(2): START-START < C: NW") and the other that landmark D is to the west with

respect to START-C (i.e., "`(2): START-C < D: W`"). More importantly, in run 56 the rule "`(1): START -> C`" fires for the first time and is followed by the rule `relpos-retrieval-success-turn-to-waypoint`. As a result, those two rules can be merged with the chunk "`START-START < C: NW`", which results in the rule "`(3): START-START < C: NW`". That rule and the rule "`(1): START -> C`" show an upward trend which clearly illustrates the model learned to move from START to C. The fact that the model has learned to move from START to C instead of from START to A can also be derived from the activations of chunks `START-A` and `START-C`, since in the final runs `START-C` surpasses `START-A` (see Figure 5.10).

The runtimes of Phase 2 will now be discussed, followed by the learned declarative and procedural knowledge.

The trend in the runtimes of the model can be explained by the development of the model just described. In Figure 5.9, one can see the decrease in runtimes during Phase 1, which is explained in the previous sub-section. The first half of Phase 2 is characterized by a high variation in the runtimes. This variation is the result of the fact that the model sometimes uses the rule `start-perception`, which results in a fast run, and sometimes uses rules that make the AIBO walk towards where landmark A was in Phase 1, which results in a relatively slow run. The runtimes in the second half of Phase 2 show a downward trend, which illustrates that the model has learned to move directly from START to C. The runtimes of the second half are also lower than the lowest runtime in Phase 1; the mean of the last 25 runs is 4.87 seconds while the fastest runtime in Phase 1 is 7.6 seconds. This is not surprising, since the route learned in Phase 2 is much shorter than the route in phase1, but it does show that the model is able to adapt and improve its performance.

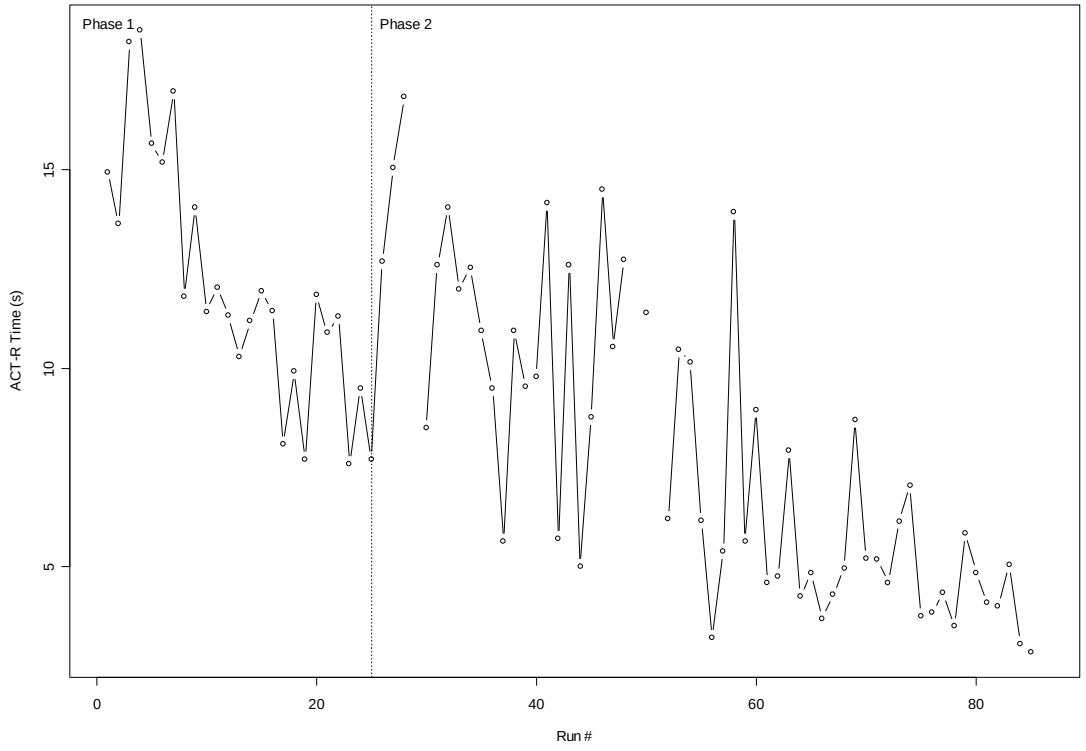

Figure 5.9:
The runtimes in ACT-R Time for Phase 1 and 2.

Note that there are three runtimes missing (run numbers 29, 49 and 51). These runs were relatively high and respectively have the values 37.8, 27.2 and 15.3. The high runtimes are due to a programming bug and to noise. In run 29 the model kept searching and did not turn around, because the temporal module stopped functioning for unknown reasons. As a result, the run had to be aborted, but only after some time, which resulted in a high runtime. In runs 49 and 51 the model mistakenly perceived a wall as the yellow landmark (i.e. landmark B). Since, there was no real yellow landmark in Phase 2 and no other object in Phase 2 that could satisfy the threshold that indicates that the yellow object is near, the robot wandered a long time before switching to a general search. After having switched to a general search and perceiving the green landmark (i.e., landmark C), the AIBO moved to that landmark and then to the pink landmark (i.e., landmark D). In short, the misperceptions led to a detour, which resulted in high runtimes.

5.2.2.1 Declarative Knowledge

Figure 5.10 shows the `route-element` chunks learned in Phase 1 and the new `route-element` chunks learned in Phase 2. Of course, the most interesting chunk learned is `START-C`. One can see in the figure that its activation increases with each run, with a few exceptions, in the end surpassing the activation of `START-A`. Also clear in the figure is the decay of activations of the chunks representing the first part of the route learned in Phase 1 (i.e., `A-B` and `B-C`).

As explained before, the production rules that make the AIBO move from START to A were still used in the beginning of Phase 2. As a result, the chunk `START-A` was also still used and the activation of that chunk did not decay as for example the activation of chunk `A-B`. However, one can see that the line representing START-A is not straight. The variations (i.e., small decays) are because of the production rule `start-perception`. When that rule was used, the chunk `START-A` was not used and only `START-C` gained a

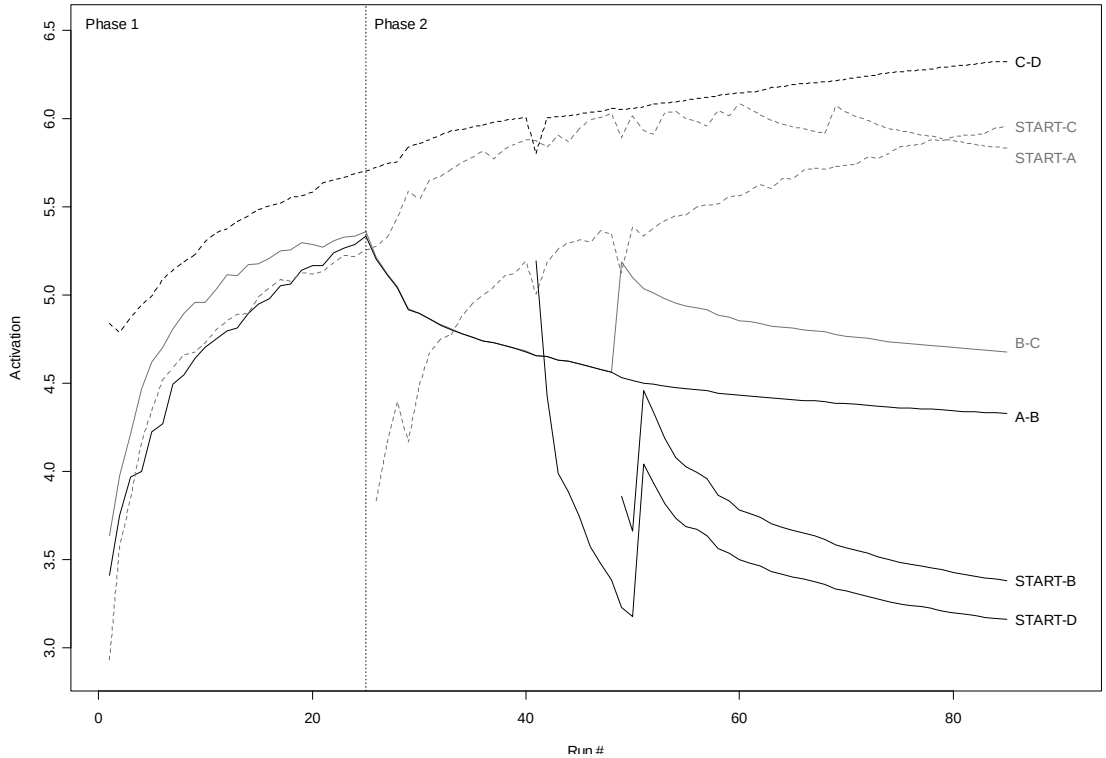

Figure 5.10:
The activations of the `route-element` chunks in Phase 1 and 2.

higher activation.

The other variations present in Figure 5.10 are mainly due to the problem with the temporal module and the misperception of the yellow landmark (i.e., landmark B). For example the relatively large decay in the activation of `START-C` at run 29, is because `START-C` was never used in that run, since the run had to be aborted. Another example is the creation of the chunk `START-B` and `START-D` at respectively run numbers 49 and 42. Because those chunks were created, `START-C` and `C-D` were not used, resulting in a sudden decay in their activations at those run numbers.

Finally, it is interesting to note that, since the model has learned to go from START to C, the chunk `START-A` is not used anymore in the final few runs. As a result the activation of chunk `START-A` continuously decays from run 69. The chunks `START-C` and `C-D` represent the two route segments of the new route START-C-D.

Besides new `route-element` chunk the model also learned new `relpos` chunks in Phase 2. The activations of the `relpos` chunks are illustrated in Figure 5.11. Note that just as in Phase 1 only the activations of relevant `relpos` chunks (i.e., `relpos` chunks that are not ignored by the model) are illustrated. Also, since most of the relpos chunks from Phase 1 only decay after run 25, the activation of those chunks is not illustrated after run 25.

In Figure 5.12 the graphical representation of the activations of the `relpos` chunks after run 85 is given. Figure 5.12 is similar to Figure 5.7, which belongs to Phase 1, but there is a new type of arrow: the dotted arrow. All the arrow types, except for the dotted arrow, are part of the route START-C-D. Thus arrows originating from START that are not dotted indicate the direction of C with respect to START. Similarly, non-dotted arrows originating from C indicate the direction of D with respect to the line START-C.

The dotted arrows represent `relpos` chunks, which are the result of noise and/or were already learned in Phase 1 and the fact that they are dotted has nothing to do with activation levels. A clear example of a `relpos` chunk learned due to noise is the arrow originating from where landmark B was, because the model mistakenly observed something as the yellow landmark (i.e., landmark B). Note that not all `relpos` chunks illustrated in Figure 5.11 are visible in Figure 5.12, because some dotted arrows are concealed by an arrow of a different type.

An important dotted arrow originates from START and points to the north-east. That arrow represents the chunk "`START-START < A: NE`" and is frequently used in the first half of Phase 2. However, just as the chunk `START-A`, the activation of "`START-START < A: NE`" continuously decays in the final runs, in this case from run 60 (see Figure 5.11).

From Figure 5.11 and Figure 5.12, one can see that the model has again learned the correct local relative directions of the landmarks and again the activation levels of the chunks have a high correspondence to their accuracy. The relpos chunks with the highest activation, represented by the thick arrows in Figure 5.12, represent the most accurate directions within the limited set of eight possible directions.

Since the directions, but also the order of the landmarks, are correctly learned, just as in Phase 1, one can conclude that the model can acquire correct and accurate declarative knowledge about a new possible route. Also, just as in Phase 1, if wrong or inaccurate knowledge is gained, the decay mechanism of ACT-R causes that knowledge to be forgotten. Only the chunks above the threshold represent accurate knowledge, where chunks with the highest activation represent the most accurate knowledge.

The procedural knowledge gained in Phase 2 is discussed next after which all the results of Phase 2 have been discussed.

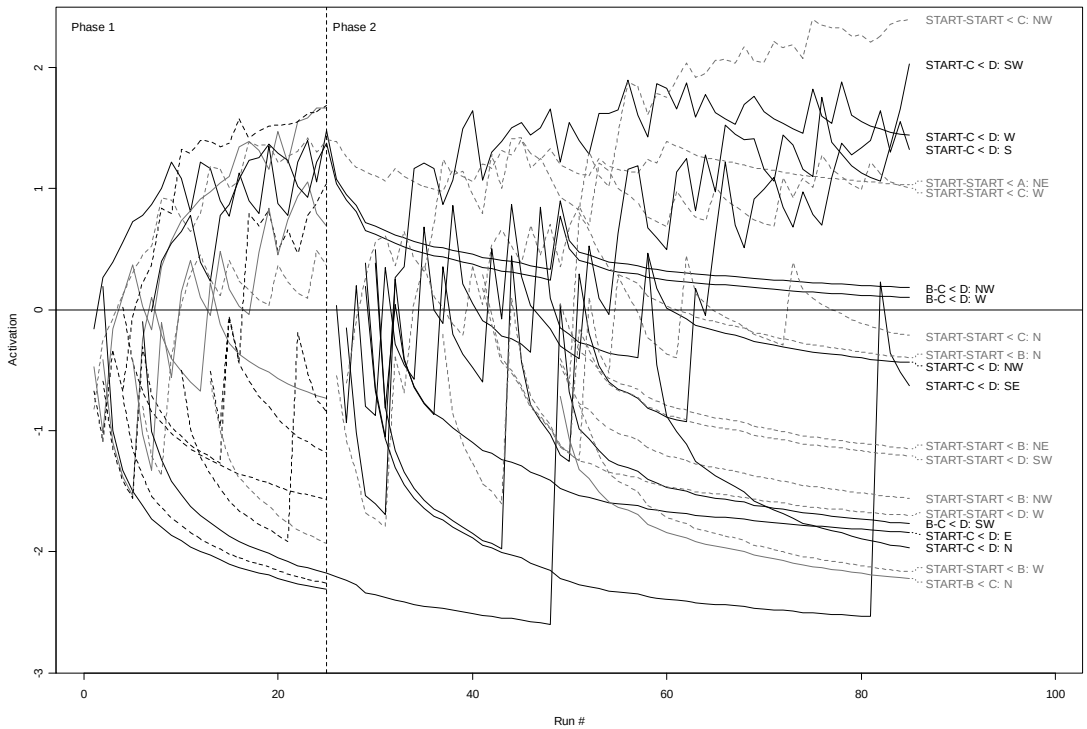

Figure 5.11:
The activations of the relpos chunks in Phase 1 and 2.

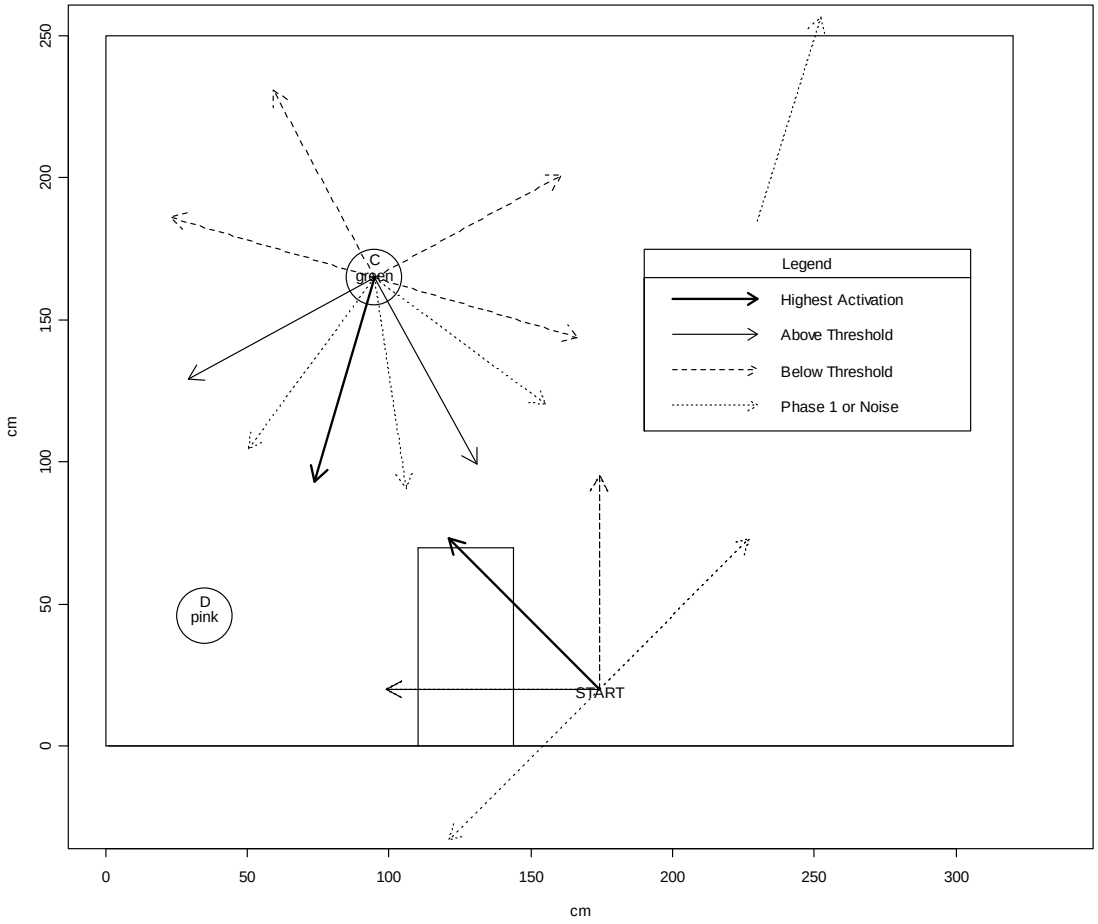

Figure 5.12:
The activations of the relpos chunks after run 85.

5.2.2.2 Procedural Knowledge

The production rules learned in Phase 2 are illustrated in Figure 5.13. Note that of some rules, the utility is not illustrated after run 25, because these rules are not used in Phase 2 of the experiment. Since the rules are not used, their utility level remains constant after run 25.

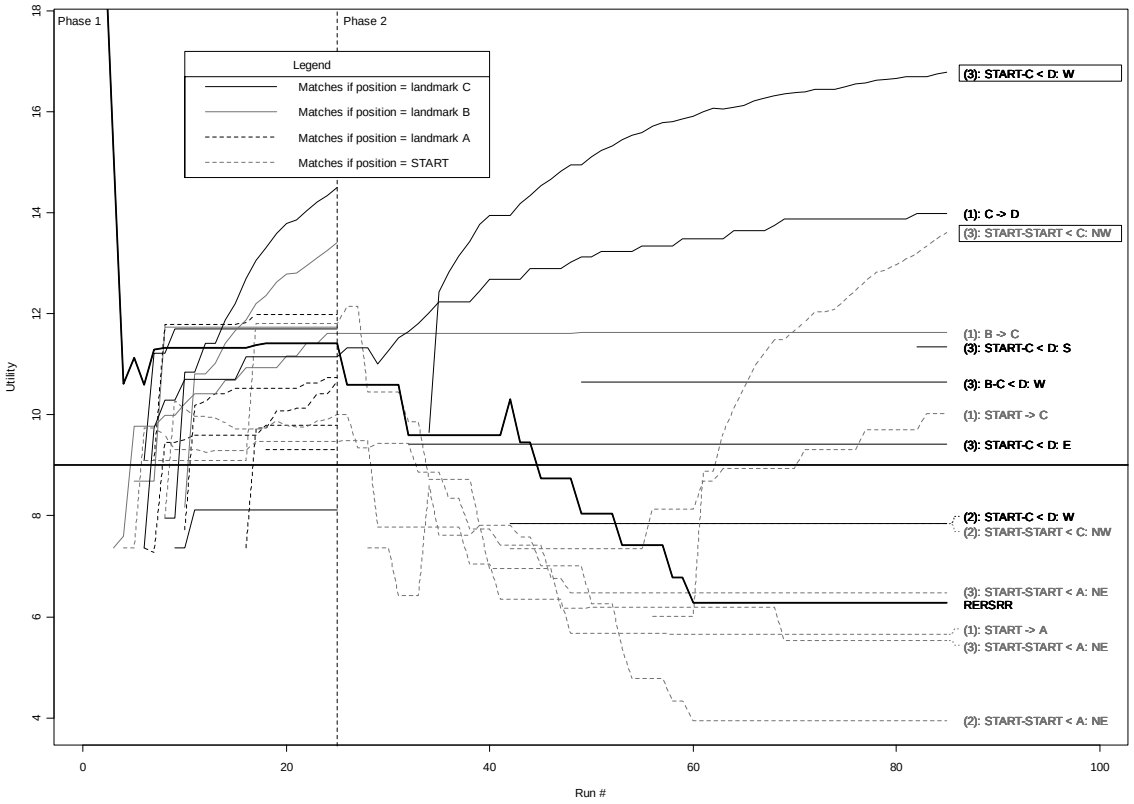

Figure 5.13:
The utilities of the learned production rules in Phase 1 and 2.

Globally, the figure is a good illustration of learning the first route in Phase 1, forgetting that route in the first half of Phase 2 and learning the new route in the second half of Phase 2. The difference between the first route and the second route is where the AIBO has to go from start. All the rules that indicate what to do from START are illustrated as grey-dashed lines. The general trend these lines show correspond to the learning development just described: first an upward trend until run 25, which represents learning the first route, then a downward trend until run 60, which represents forgetting that route, and then a upward trend, which represents learning the second route.

As mentioned before, in Phase 2 the model first moves towards the location of where landmark A was in Phase 1. However, because landmark A is no longer present, the model cannot find it and has to search for a different landmark. Eventually, landmark C is found, but first searching for landmark A costs a lot of time. As a result, the utility of the rules representing the behaviour to go to landmark A from START (i.e., "`(1): START-A`", "`(2) START-START < A: NE`", "`(3) START-START < A: NE`" and "`(3) START-START < A: NE`") decrease. Around run 35 the utility of those rules are all below the utility of `start-perception` and `start-route-element-retrieval`[4], giving those two rules a higher chance to fire.

[4] Remember that the horizontal line at a utility level of 9 represents the utility of `start-perception` and `start-route-element-retrieval`.

The rule `start-perception` causes the chunk `START-C` to gain on `START-A`, as explained before. As a result, `START-C` can be retrieved by `start-route-element-retrieval` at which point, the new rules "`(1): START -> C`", "`(2): START-START < C: NW`" and "`(2): START-C < D: W`" are created. These rules respectively represent that after START, landmark C should follow, that C is towards the north-west from start, and that D is towards the west from the line START-C. The latter two rules are not used in later runs, because at run 56 the rule "`(1): START -> C`" fires and soon thereafter gains a higher utility as `start-route-element-retrieval`. The rule `start-route-element-retrieval` needs to precede the rules "`(2): START-START < C: NW`" and "`(2): START-C < D: W`", because they are of type (2) (for a more detailed explanation see section 5.2.1.2 and Figure 4.2).

Also, because "`(1): START -> C`" fires the rule "`(3): START-START < C: NW`" can be created, which brings the model much closer to achieving the goal of Phase 2. At run 61 the rule "`(3): START-START < C: NW`" fires for the first time. The rule is very successful and immediately surpasses "`(1): START -> C`" and almost `start-perception`. In the runs following run 61 "`(3): START-START < C: NW`" is almost the only rule to fire when the AIBO is at START. Of the 24 runs that remain of Phase 2 after run 61, "`(3): START-START < C: NW`" is used 17 times. In the remaining 8 runs "`(1): START -> C`" is used 4 times and `start-perception` twice. These numbers together with the upward trend of both "`(3): START-START < C: NW`" and "`(1): START -> C`" indicate that the model has learned the segment START-C as procedural knowledge.

Finally, the model has also correctly learned the segment C-D as procedural knowledge. One can derive this from the utility of the rules "`(1): C-D`" and "`(3): START-C < D: W`", which have the highest utilities at run 85. The rules "`(3): START-START < C: NW`" and "`(3): START-C < D: W`" together form the procedural representation of the route START-C-D and are marked with the rectangles in Figure 5.13. The gained declarative knowledge and the fact that the model has learned the two marked rules, indicate that the model has achieved its second goal, which is to learn a shorter route after having learned a different route.

5.3 Remarks

In this chapter the experiments and results have been discussed. The results are as expected. They show that the model is able to learn a route and, when the environment changes, is able to adapt and learn a second route. The model is able to learn both routes as declarative knowledge and procedural knowledge. Also the durations of the runs indicate that the model is able to improve its performance, that is, it is able to decrease the amount of time needed to reach its goal.

In the next chapter the model and its results will be compared to the other models and Spatial-Learning theory, which both have been discussed in the Theoretical Background chapter.

6. Discussion and Conclusions

The goal of this project is to examine which mutual benefits arise from integrating the research areas cognitive modelling and robotics. To accomplish that goal, an embodied cognitive model of route learning was developed with a number of constraints. In short, the constraints are: the technical limitations of the AIBO, operation in a real-world environment, cognitive constraints due to the ACT-R architecture (Anderson, 2005; Anderson et al., 2004) and the minimal control design principle (Taatgen, 2007). Furthermore, the model has to be a plausible model of route learning and has to be able to perform two tasks, which are the sub-goals of the current research. First, the model must be able to learn a route and second, it must be able to adapt to changes in the environment to learn a different route.

The route to be learned existed of a start point and four landmarks and after a change in the environment two landmarks remained and a shorter route was possible. During an experiment the model had to learn the route and after the change had to adapt to learn the shorter route. This chapter will first discuss the AIBO-Route model and the results of the experiment with respect to the Spatial-Learning theory and existing models discussed in the Theoretical Background chapter. The chapter will continue with the discussion of the advantages of combining cognitive modelling and robotics and will finish with the conclusion, which summarizes the results with respect to the research goals.

6.1 Discussion of the AIBO-Route model

Through the use of ACT-R, the AIBO-Route model incorporates many cognitive aspects. Of these aspects, the declarative and procedural memory with respectively their activation and utility mechanisms are the most important. These mechanisms enable the model to gain declarative as well as procedural knowledge about a route. Procedural knowledge, as indicated by Gale et al. (1990) is one of the most important aspects of route learning, since, as described in the theoretical background, people who travel a route passively by, for example, observing a video, often lack procedural knowledge with respect to people who actively travel the same route in the real world.

Also an important feature of ACT-R's procedural memory is the production compilation mechanism. In the AIBO-Route model, the production compilation mechanism might explain the event in which people travel a route, but upon reaching their destination do not remember some parts of that route. It is as if they automatically, more or less unconsciously, made the choices needed to get to their destination. Since the production compilation mechanism causes declarative knowledge to be merged into procedural knowledge, the need to retrieve certain facts is eliminated. Because those facts are not retrieved, they also do not enter the buffers. However, the content of the buffers is often seen as the things one is aware of. If that is true, the production compilation mechanism prevents the model to become "aware" of certain facts. With respect to the task of travelling a route, those facts could be changes in direction or the order of nodes to visit. If these facts never enter the buffers, as is the case when only procedural memory is used, one would be less aware of the decisions made during travel. However, one is not completely unaware, because, for example, there is always visual and motor feedback of one's actions. This feedback causes some facts to enter the buffers, thereby making one aware of those facts.

Although the AIBO-Route model is able to explain "unconscious" travel, it lacks some reasoning processes at a conscious level. The routes are learned completely through

sub-symbolic or unconscious processes of activations and utilities. As a result, a relatively high number of runs is needed to forget a route and learn a new route. It is hard to determine a plausible number of runs, since the route used in the experiment is relatively easy for humans. Therefore, even if experimental data of subjects were to be available, that data would be hard to compare to the results of this project. However, more than forty runs to forget a route and learn a new route as procedural knowledge certainly seems too much. On the other hand, the number of runs to learn declarative knowledge of the route is not unreasonable. After one run the model has learned the order of the landmarks, which is plausible given that there are only four landmarks in the first route and two in the second. In addition, the relative directions are learned in about six runs, which might be a bit high, but spatial information is harder to learn as is explained in the theoretical background. Furthermore, Chown, Kaplan and Kortenkamp (p. 21 1995) state that first people learn the order of landmarks and after a while learn their relative directions.

Since the learning speed of declarative knowledge is in the right order of magnitude, the procedural knowledge that uses that knowledge must be incomplete. What is missing is a higher reasoning level. One might have noticed by now that the model seems a bit unintelligent with respect to the part where it walks towards landmark A when it is no longer there. It keeps doing that for a large number of runs while it could have reasoned that it makes no sense doing so, because landmark A is simply not there. If a higher reasoning level would have been present a thought process like: “I am at START and should go to landmark A, but landmark A, I remember, is no longer there thus I should do something else” might have been possible.

To further illustrate the higher reasoning level that is lacking an example is given below. Imagine one has travelled a route from home to work many times and one has complete procedural knowledge of this route and can travel it “unconsciously” as mentioned before. Then at some time, a new bridge is build and a much shorter route has become possible. As long as one consciously decides on where to go, one can reason that the new route via the bridge is best. However, if the one were to travel to work while not paying enough attention to the route, one might accidentally take the old longer route. In such a scenario, it is the higher reasoning level that is not used. The AIBO-Route model therefore is a model of the lower processes regarding route learning. For example, the activations of the chunks representing knowledge of the first part of the route learned in Phase 1 decrease after the model has learned the new route in Phase 2 (e.g., `START-A` in Figure 5.10). The decrease in those activations, indicate that the model is slowly starting to forget the old route. However, although the declarative part of the model functions sufficiently with respect to the current research, it lacks metric knowledge or another indication of distance/effort that is necessary for the higher reasoning level.

The fact that the model needs many runs before it has learned to travel the new route is a perfect example of bounded rationality (Simon, 1957): within the limitations of the model a near optimal solution is found. The model is limited or “bounded”, because it lacks the higher reasoning level and therefore needs relatively many runs to find a near optimal solution. Also, the fact that the model repeats previous learned behaviour, even though the setup of the environment has changed such that a different more efficient behaviour is possible, is a good example of the Einstellung effect (Luchins & Luchins, 1959) mentioned in the Introduction chapter.

In the introduction, it was also stated that because of the Einstellung effect a second similar task, such as a second route, could be solved quicker because of knowledge gained in the first task. Unfortunately, the results do not immediately show that possibility. However, the model needs a lot of time to forget the first route and learn

the second route because it lacks the higher reasoning level. If that level had been present and if the route was longer, a positive effect of the Einstellung effect might have been present. As a result of the longer route, there would have been a bigger overlap between the first and second route. The model could then have used its knowledge from the latter part of the first route to travel a part of the second route. If this knowledge included the relative directions of landmarks part of the second route, the model could start walking towards those landmarks without first searching for them. As a result, the model would be faster than it would be without that knowledge. Therefore, the knowledge acquired when travelling the first route could be beneficial when travelling the second route.

A longer route does create a situation in which the Einstellung effect could be beneficial, but also creates a problem. In the AIBO-Route model only the last two landmarks visited are remembered, which is sufficient for a setup with four landmarks. However given a different setup with more landmarks, where also more landmarks might be visible at once, a different strategy is needed to keep track of the visited landmarks. The current solution, to keep the two last visited landmarks in two slots of the goal chunk is not a plausible solution. How exactly people keep track of where they have been, remains an open question, but one that could be examined to improve the AIBO-Route model.

What representation should be used to keep track of visited locations is one of many problems of representation. Closely related is, for example, how one knows one has turned around to search behind oneself. Another example is how the segmentation process should be represented. Although the model uses `route-element` chunks to represent the smallest possible segment of a route, it is known (e.g. Lynch, 1960) that people also group those segments to summarize larger parts of a route. The AIBO-Route model does not provide that possibility, but it might be possible by using a different chunk-type that does not represent a direct path between two nodes. A similar representation as used in NAPS (Levenick, 1991), which is part of the PLAN model (Chown et al., 1995), could be used to facilitate the segmentation process. How exactly, is explained in the Future Work chapter.

The segmentation process is also present when it comes to metric knowledge. However, the `relpos` chunk in the AIBO-Route model only provides directional information and no information about distance. The model therefore has only a rudimentary understanding of metric relations. To include plausible metric knowledge, not only the AIBO-Route model would have to be expanded, but also the AIBO-R architecture, because, currently, it cannot provide any additional metric information besides directional information. To expand the AIBO-Route model, again, inspiration from the PLAN model might be used, as PLAN's (Chown et al., 1995) R-NET provides a theory on how metric knowledge could be represented. PLAN, however, is not very specific about how that metric knowledge is acquired. To that end, the approach used in TOUR (Kuipers, 1977, 1978) might be useful, also because the representations used in TOUR closely resemble those used in ACT-R. Besides PLAN and TOUR, Qualnav (Kuipers & Levitt, 1988) also has interesting theories on the representation and acquisition of metric and topological knowledge that could be used as inspiration to expand the AIBO-Route model. However, as just mentioned, the AIBO-R architecture would first have to be expanded such that it can provide additional metric knowledge, before the AIBO-Route model could be modified to fully reason with metric knowledge.

Although, the AIBO-Route model is lacking many aspects of metric knowledge, the local directional information represented by the `relpos` chunks is sufficient to help improve the performance of the model. Currently the `relpos` chunk can contain eight possible directions, since these directions were also used in PLAN, but how relative

directions between objects are represented exactly, remains an open question. For the current model, the limited eight directions work, because after the model starts moving in the direction of an object, it also starts searching for it. As a result, the model can find the object and use visual information to guide itself towards the object. Since the model moves in segments, the direction can only be corrected at the beginning of each segment, which results in behaviour where the model makes corrections that become smaller as the model approaches the object. This trend of smaller corrections closely resembles Fitt's law (Fitt, 1954) and provides robust behaviour, because an initial direction does not have to be very accurate. In short, the `relpos` chunk representation does not have to be very accurate, which is a step towards navigating in more dynamic environments, while it also helps to improve the model's performance.

Another analogy between the model and Fitt's law can be found when looking at the utility levels of the production rules. Figure 5.8 and Figure 5.13 show that the utilities of productions rules are higher, if those production rules are used when the AIBO is closer to the goal. For example, in Figure 5.8, the utility of the rule "`(3) A-B < C: W`", which is used at landmark B, is lower than "`(3) B-C < D: NW`", which is used at landmark C. As a result, it is more likely that a general rule from the first or second strategy fires when the model is at landmark B than when it is at landmark C. Since the general rules are more prone to errors than the through production compilation learned rules, the model is less likely to make mistakes as it approaches the goal. It is interesting to examine whether this is true for humans as well.

As mentioned before in the previous chapters, there are three production rules that compile. The first rule requests a chunk that holds the next object given the previous and current location, the second rule requests an direction given the next object, and the third rule moves in that direction if a direction was retrieved. These three rules are the most important rules of the model as through these rules the model makes it primary decisions. The sequence of the three rules can be seen as one of the variations proposed by Kuipers (1983) and was represented as (1) $V \rightarrow V'$ and (2) $(V\ V') \rightarrow A$ in the theoretical background. V is the current view, which in the AIBO-Route model is represented by the previous and current location, V' is the next view and is represented by the next object, and A is the action, which belongs to the combination of the two views and is represented by moving in a certain direction.

It is interesting to note, that the AIBO-Route model could be modified to fit another variation: (1) $V \rightarrow A$ and (2) $(V\ A) \rightarrow V'$. After the modification, the first rule would request a direction given the previous and current location, the second rule would start moving towards that direction (if retrieved) and try to retrieve the next object given that direction, and the third rule would start searching for the next object if it was retrieved. Kuipers (1983) predicted that when the first representation was used, one would be able to recall the sequence of landmarks encountered during a specific route, but not always which actions are needed to get from one landmark to the next. This prediction is confirmed by the results of the experiment of the current research, as the activations of the `route-element` chunks are higher than those of the `relpos` chunks. Also, the first few runs of the model show that often the model is able to recall a `route-element` chunk, but not a `relpos` chunk. It would be interesting to see if Kuipers' predictions about the latter representation also hold.

Although the utilities of the three production rules and compiled versions of these rules are discussed in detail in the previous chapter, most of the utilities of the non-compiled production rules are left out of the results as they have no effect on the behaviour of the model. They have no effect, because they are not in competition with

any other rule, since, in accordance with the minimal control principle, each rule matches a specific set of conditions. The rules might have had competition if new rules would have been created through the production compilation process, but since the robomotorical and roboperceptual modules inherit the product compilation rules from the motorical and visual modules, respectively, there are very few situations where rules using one or both of the robo-modules can compile. Whether the inheritance of product compilation rules by the robo-modules is correct or whether different rules apply, is open to debate.

Besides the compilation rules, also the production compilation process itself might be debated. As mentioned in the previous two chapters similar production rules might be created in different ways. For example, when there are three rules, A, B and C, the first two and the latter two might compile to respectively AB and BC. Next, the new rule AB can compile with C and the new rule BC can compile with A, both resulting in AC. In ACT-R, as shown in the Experiment and Results chapter, this would lead to two identical rules with their own utilities. However, they represent the same knowledge and therefore it might be possible that they must be fused into one production rule with one utility value. Since learning and forgetting with two identical production rules instead of just one would lead to different results, an experiment could be conducted to examine whether two identical production rules should be fused or not.

A general problem of ACT-R, closely related to the production compilation mechanism, is how one gains procedural knowledge in the first place. Through the defined production rules, a model is able to create new procedural knowledge, but the initial production rules have to be defined manually. Whereas declarative knowledge can be gained through visual and auditive perception of the environment in combination with reasoning processes, the acquisition of procedural memory (i.e., production rules) is still missing in ACT-R. Perhaps all procedural knowledge can be derived from some basic rules already present at the moment of birth, just like, for example, many laws in physics can be derived from a limited set of basic formulas (i.e., Grand Unification Theory).

Before the advantages and disadvantages of combining cognitive modelling and robotics are discussed, one final subject regarding the model itself will be discussed. As has been mentioned several times in the previous chapters, the visual perception, and object classification in particular, is simplified in the AIBO-Route model. Currently those processes are treated as a black box and, of course, a complete model should contain those processes as well. However, as already explained in the Theoretical Background chapter, the perception component is very hard to model. The human brain performs exceptionally well when it comes to visual perception, which has been studied for many years by researchers from several research areas. One component of perception is particularly hard to model, which is the top down influence. It is well known that the mental image created in one's brain is very different from what one's eyes perceive. Besides that, the bottom up mechanisms that process several features like edges, movement and colour are computationally demanding. The combination of the complexity of visual processing and high computational demand make it hard to implement many aspects, let alone all aspects, of visual perception. Therefore, it is hard to create a complete model of route learning. Thus, for future embodied cognitive models it is a challenge to also include the perception and object classification processes.

6.2 Advantages of combining Cognitive Modelling and Robotics

Above, the AIBO-Route model and AIBO-R architecture have been discussed with respect to the Spatial-Learning theory and existing models. Next, the advantages and disadvantages of combining cognitive modelling with robotics will be discussed.

One of the advantages of combining a cognitive model with a robot is the perception problem. This probably sounds contradictory, but implementing a cognitive model on a robot forces one to deal with problems that might not have presented themselves at all when creating conventional models. Therefore, implementing a cognitive model on a robot leads to more complete and plausible models. For example, many of the simulated navigational models discussed simply ignore the perception and object classification problems, something which is impossible when implementing such a model on a robot. Admittedly, in the AIBO-Route model the perception and object classification are simplified, but at least they are included in the model. Embodied cognitive modelling therefore forces one to take aspects into consideration, which otherwise might have been missed.

Another example of an aspect that might have been missed in conventional cognitive models is the fact that the AIBO turns around when no interesting object is perceived in front of it. Qualnav, for example, uses a 360 degree view which makes turning unnecessary. However, the turn included in the AIBO-Route model has proven to be an important aspect, because turning around takes relatively long, thereby influencing the utility learning process. For example, when the AIBO has learned the direction of an object that lies behind it at a certain point, the behaviour of first searching in front and then turning around are replaced trough utility learning and production compilation by the behaviour of immediately turning towards the object. Since first searching in front of the AIBO and turning around are time consuming processes, the learned behaviour of immediately turning towards the object, has a large advantage with respect to the old behaviour. Because of the large advantage (i.e., faster solution), the production rules that represent the behaviour of immediately turning around, gain a higher utility than the production rules representing the old behaviour. However, if the model would have had a 360 degree view, the advantage would not have been that large and therefore might have prevented the use of new production rules. The turn behaviour therefore is important for the learning of new production rules.

The time consuming process of turning around also helps to forget the behaviour of moving from START to A in Phase 2. In the first few runs of Phase 2, the AIBO moves to the location of where landmark A was in Phase 1. After having moved, the AIBO has to turn around to find landmark C. Since turning around is time consuming, the utilities of the production rules that cause the AIBO to move into the wrong direction, decrease. Because of the decrease in the utilities of those production rules, other production rules, for example start-perception and start-route-element-retrieval, have a higher chance of being selected. This example demonstrates again that the behaviour of turning around influences the learning process. Since the turning around behaviour is important and might have been absent in a conventional model, implementing a cognitive model on a robot has proven to be beneficial.

Another advantage of implementing a cognitive model on a robot is the fact that the durations of certain behaviours and the variations in those durations due to noise do not have to be simulated. For example, in the real world slight variations in friction during movement and lightning conditions are always present. The presence of these variations is important, because they might influence the development of a model. For example, lightning conditions might cause misperceptions or a certain path might be faster because a robot has more grip. In the experiment of this project, misperceptions were present, for example, the model mistakenly perceived something as the yellow landmark. Despite the misperception the model was able to complete it task and satisfy the goals of the current project. Since, the mentioned variations are not present when testing conventional models, it is unknown whether they would have been able to cope with them. Therefore,

if a cognitive model implemented on a robot is able to perform a task in the real world, the model is more likely to be a correct representation of the cognitive processes belonging to that task. As a result, data generated by embodied cognitive models is more reliable than data generated by conventional models.

In short, it is important to include all possible aspects of a task, since they all might have an influence on the development of a model. Since, implementing a cognitive model on a robot helps to find those aspects, embodied cognitive models are likely to be more complete than conventional cognitive models. Also, embodied cognitive models implement all three levels proposed by Marr (1982). Of course, it is harder to build an embodied cognitive model, but, just as in the AIBO-Route model, some components could be simplified for pragmatic reasons and in future work be replaced by more detailed components.

So far only benefits for using robots in the area of cognitive modelling have been discussed, but robotics can profit from the area of cognitive modelling as well. Several advantages of using cognitive modelling in the specific area of human robot interaction have already been mentioned in the introduction and were examined by Trafton et al. (2006). However, there are more advantages when using cognitive modelling in the area of robotics, especially when using a cognitive architecture such as ACT-R.

One of the advantages of using a cognitive architecture to control a robot is that the learning mechanisms of the architecture can be used. In the area of robotics several learning algorithms have been used, like, for example, Bayesian learning and neural networks. However, these are usually designed for a specific task and have to be created from scratch, while, if using a cognitive architecture, one would get a learning mechanism for free. In case of the AIBO-Route model, the learning mechanism was used to learn a route and adapt to a different route. Closely related to such an advantage are the unified representations used in cognitive architectures. By using similar representations for information from different sources (e.g., vision, sound and memory), this information can be compared more easily than when each source has its own representation. For example, in the AIBO-Route model information from an `observation` chunk is compared with that from a `route-element` chunk. In addition, the logical rules, which are common in cognitive architectures, can reason with information from any source, since all information is represented using a similar representation. In short, a cognitive architecture provides unified representations and learning mechanisms that are able to work with those representations.

Besides the general ability to learn, the learning mechanism and unified representation also makes it possible to reuse knowledge gained during a certain task in another similar task. Unfortunately, as explained before, this advantage was not demonstrated by the AIBO-Route model. However, as also explained, the AIBO-Route model could have demonstrated the advantage if the route had been longer. Fortunately, the AIBO-Route model did show the ability to solve two similar tasks using the same unmodified model, which is also an advantage of using a cognitive model to control a robot.

Another advantage of the use of cognitive models in the area of robotics is that existing models could be used as inspiration. If a robot is to perform a task of which such a model exists, that model could be used as a basis for the model that is to control the robot. As a result, many problems that accompany a certain task have already been solved, and, as a bonus, the cognitive model is expanded to work in the real world, which could provide new insights into human cognition with respect to the task modelled.

Using a cognitive model (a cognitive architecture in particular) as inspiration also creates the possibility of combining models of different tasks into one model. Since cognitive models, using the same architecture, have the components of the architecture itself in common, they can be combined more easily than two random conventional algorithms used in robotics. Such cognitive models share the same unified representation for declarative and procedural memory, which is especially interesting when combining two closely related tasks. For example, it would be interesting to see how declarative and procedural knowledge from both tasks merge and complement each other. However, most importantly, a robot which is able to perform two tasks using a single architecture is an important step towards human cognition, since humans are able to perform all tasks using the same architecture.

Another thing humans are capable of is using top-down knowledge in perception and classification. Most of the time this happens unconsciously, but sometimes when perception conditions are poor humans use top-down knowledge more consciously. For example, when observing three lights in the distance, one might reason that one's house is where the middle light is, although one cannot see the house. An example of using top-down knowledge in normal perception conditions is when there are several identical drinking glasses on the table and one knows which glass is his or hers because of the location of the glass on the table. When using a cognitive model to control a robot, it might be possible to take advantage of such top-down reasoning processes. In the current research an experiment was attempted to show that advantage, but unfortunately the AIBO-Route model lacked sufficient high level reasoning skills.

The idea behind the experiment was to use four landmarks of which two landmarks would have similar colours. Trough calibration, one of the landmarks could be perceived as either of the two landmarks and therefore would be ambiguous. In the first few runs the AIBO-Route model should perceive the ambiguous landmark as either of the two. If the model classifies the ambiguous landmark as the other, the model is stuck, because it should not go somewhere it has already been. At this point, it is important to remember that at most one landmark is visible at each node. However, if the ambiguous landmark was correctly classified, there would be no problem. As a result of the activation, utility and production compilation mechanisms the model should learn to search for an object that matches the correct classification of the ambiguous landmark, thereby eliminating the perception problem of that landmark.

Unfortunately, as mentioned, this experiment failed because higher reasoning skills are lacking. The AIBO-Route model continuously processes the camera image and therefore classifies an object not only once, but each time it requests an `observation` chunk. As a result, the ambiguous object is classified many times in a short time interval. Among those classifications both possible classifications of the ambiguous landmark are always present. Thus the scenario in which the ambiguous object is correctly classified and the AIBO moves on has a very low probability. That would happen only if each observation request would lead to the correct classification, which in practice is almost impossible. Therefore, for the experiment to work, the AIBO-Route model should, for example, be able to reason that the object just perceived could not suddenly be another object. Another possibility is that the model reasons that the next object cannot be the same as the one just perceived, because its location is different (and the object cannot move). One might think the experiment would have been possible using a different order of the landmarks, but all possible orders, using four landmarks positioned as in the original experiment during Phase 1, resulted in failure. Either the model would be lost, since there was no landmark to go to, or it would be stuck in an infinite loop, because the

model could repeatedly travel between two landmarks if one of those two landmarks could be perceived as two possibilities[5].

Although a second experiment demonstrating the advantage of using top-down knowledge failed, it is still plausible that such an advantage is possible using a cognitive model to control a robot. However, such a model would have to include the higher reasoning processes (i.e., procedural knowledge) required to use that top-down knowledge. In future work the AIBO-Route model could be modified to include those reasoning processes and hopefully would then be able to demonstrate the advantage of the usage of top-down knowledge in perception.

6.3 Conclusions

The primary goal of the current research is to explore what insights can be gained by combining cognitive modelling and robotics. Above, these insights have been discussed and it is certain that cognitive modelling and robotics can benefit from each other. It is, however, not an easy task to utilize the advantages, as can be read in the previous two sections and section 3.5 in which the interface between AIBO and ACT-R was discussed. It is also difficult to include all aspects of a task such that an advantage is utilized, but working towards that goal does help to build more complete cognitive models. On the other hand, controlling a robot with a cognitive model is also a challenge, because one has to deal with many more constraints. However, again, the advantages are worth the effort, because robots controlled by cognitive models inherit the learning capabilities provided by a cognitive architecture. Furthermore, a robot controlled by a cognitive model resembles a human more closely than a conventional robot algorithm and therefore is a step closer towards explaining and applying human intelligence.

Besides the primary goal there are two sub-goals which are:

1. Given a setup of several landmarks the AIBO-Route model should be able to learn a route to a predefined goal.
2. When having learned such a route and the environment changes in such a way that a shorter route is possible, AIBO-Route should be able to learn the new shorter route.

Regarding the first sub-goal, the results of Phase 1 clearly indicate that the AIBO-Route model is able to learn a route. This is supported by the acquired declarative knowledge as well as the acquired procedural knowledge. In addition, the decrease in time necessary to complete a run demonstrates an improvement in performance, which is also a clear indication of learning.

Just as the results of Phase 1 satisfy the first sub-goal, the results of Phase 2 satisfy the second sub-goal. In Phase 1 the model has learned a route, which in the first half of Phase 2 was de-learned and in the second half of Phase 2 the model successfully learned the new shorter route. It is important to note however, that forgetting the first route took relatively long, because higher reasoning levels were lacking in the AIBO-Route model. The AIBO-Route model must therefore be seen as a model of the lower cognitive processes involved with route learning.

[5] Imagine a landmark X, which can be classified as A or B and a second landmark Y, which is classified as C. The AIBO-Route model remembers only the last two landmarks visited, therefore the next orders are possible: in the case of X-Y: (A-C-B)-(A-C-B)-etc or (B-C-A)-(B-C-A)-etc and in the case of Y-X: (C-B-A)-(C-B-A)-etc or (C-A-B)-(C-A-B)-etc. As a result the model is stuck in such loops. Using similar analyses it can be proven that all orders of the four landmarks lead to failure.

Furthermore, it is important to note that although existing models implement some aspects of route learning in more detail and some existing models have aspects which the AIBO-Route model lacks, the AIBO-Route is a model that contains aspects from the lowest to the highest level. In contrast, most of the existing models have focussed on only one or two aspects of spatial cognition. For example, mobile robot models primarily focus on perception and functionality and simulated models focus on higher reasoning processes, whereas AIBO-Route combines aspects from all levels into one functional, plausible model of route learning.

In short, combining a cognitive model and a robot is a challenging task, because there is a strong connection between higher and lower levels of cognition. Also, if there is one thing the current project has shown, it is that cognitive models are a long way from moving around in the real world as humans do. However, there is no reason to become pessimistic, as it certainly seems possible. A lot of future research has to be done, but it is certain that the work of combining cognitive models and robotics will be very rewarding.

7. Future Work

It is mentioned several times throughout this document that a more detailed model of object classification could be added to the AIBO-Route model. Besides that, there are a few more possible additions and experiments, which will be discussed in this chapter. This chapter does not describe in full detail what the additions and expansions should look like, but it gives a global overview of work that that could be done in the future to improve the AIBO model and AIBO-R architecture. Next, the detailed object classification will be discussed.

Currently, the model classifies any object as a waypoint object and the only property such an object has, is its colour. This classification is done by the roboperceptual module after a request through the robovisual buffer. At that point there is some simple function that returns an `object` chunk, which is always of class "waypoint". However, that code could be replaced by a different function, which performs a more complex object recognition process. To create such a function, inspiration from the area of computer vision could be used. The function could then, just as the current function return an `object` chunk with multiple properties like, for example, the number of legs, colour and height. With the addition of a more complex object classification system, the model could, for example, distinguish a red chair from a red table, which is currently impossible, but obviously necessary for a complete model of route learning.

A more complex object classification system provides more possibilities and thereby creates a new problem. In the current model any object is a waypoint, and thus a possible navigation point. However, if an object is not immediately classified as a waypoint, the model does not know whether it is a navigation point belonging to the route or not. Therefore, after an object has been classified it should be matched against declarative memory to determine whether it is an object on the route or not. If it is, the model could move towards it, and if it is not, the model could continue searching.

The object classification system could also implement an algorithm to determine an object's saliency. In ACT-R the activation of a chunk only depends on the previous encounters and association with other chunks, but saliency could have an effect too. For example, very striking objects might be easier remembered than ordinary objects. Recently a theory, referred to as RACE (Van Maanen & Van Rijn, in press), has been developed to model what happens between a retrieval request and the actual retrieval. Saliency could be incorporated into an implementation of that theory such that striking objects are easier remembered than ordinary objects. Of course, striking objects might also be easier remembered, because one divides more attention to those objects. Whether striking objects are easier remembered because of their saliency, or whether they are easier remembered because of the attention they receive, must therefore be examined first.

Although it is uncertain if saliency directly influences the remembering of an object, it is certain that saliency, among other things, determines which objects are attended. A detailed description of how saliency influences the selection of objects can be found in the work of Gopal et al. (1989). In contrast to choosing an object at random, adding a mechanism to select objects based on their saliency might make it possible to create a model that handles multiple visible objects in a plausible way. As a result, the environment would be searched in a more natural way. However, adding a mechanism that takes saliency into account is not sufficient. The AIBO-Route model searches the environment from right to left and back, but that is not how humans search an environment. A future modified version of the AIBO-Route model should therefore include several different searching mechanisms.

Another, but completely different, addition that would make the AIBO-Route model more complete is the addition of a segmentation mechanism. AIBO-Route can only learn the lowest possible segments of a route, that is, two landmarks, which are directly linked. To make a fully segmented hierarchical representation of a route possible, chunks of a different chunk-type than `route-element` could be added. The `route-element` chunk would only be used to determine if it is possible to directly move from one point to another and the new chunk-type could be used to represent nodes that are connected through several `route-element` chunks. The example below illustrates how these new chunks could be created.

Imagine the model first learns a route as declarative knowledge through `relpos` and `route-element` chunks. However as the model travels the route more often, it gains procedural knowledge about the route. As a result, some declarative knowledge has become obsolete. The model can now travel parts of the route using procedural knowledge. However, at some point procedural knowledge may still be lacking and the model would need declarative knowledge. The nodes at the beginning and end of the part travelled using procedural knowledge could be stored in a chunk of the new chunk-type. The nodes are not directly connected, but it is known that there exists a route between them. As the model gains more procedural knowledge through production compilation, the nodes that are stored in the new chunk will be further apart. The chunks containing such nodes are therefore a summary of parts of the route and together form a hierarchical topological representation of the route.

To reason with those segmentation chunks the model would need additional procedural knowledge, which represents a higher reasoning level. At that level the model should also be able to reason with distance such that shortcuts and new paths can be found. However, in order to gain information about distance, as mentioned before, the AIBO-R architecture would have to be expanded, because now it can only provide directional information to the AIBO-Route model.

Finding shortcuts and different paths through reasoning, creates scenarios in which the model travels much larger distances. However, as of now, the model is only able to function in the robot lab. Therefore before any higher level reasoning processes can be fully utilized, the AIBO-Route model should be able to function outdoors. For the AIBO-Route model to be able to function outside, many more expansions are needed, like, for example, object avoidance, a robot with better movement capabilities and a better localization algorithm. Thus, a complete embodied cognitive model of route learning would need a better link between low level (perception, movement and localization) and high level (reasoning about distances, objects, shortcuts, etc.) processes.

Eventually, after many of the additions and expansions have been added to the AIBO-Route model and AIBO-R architecture, several interesting experiments could be conducted. For example, in the theoretical background it was discussed that humans first use global landmarks to navigate and later, when they are more familiar with the environment, rely more on local landmarks. It would be interesting to see if, for example, the saliency mechanism in combination with different search processes would also show such behaviour. Finally, an ultimate experiment would be an experiment where the model learns a route and through those routes forms some kind of mental map. The information gained, could then be used to draw maps, which in turn could be compared to maps drawn by humans that have also learned those routes. Such experiments will probably be near to impossible for some time, but it is interesting to work towards such a goal.

8. References

Aginsky, V., Harris, C., Rensink, R., & Beusmans, J. (1997). Two strategies for learning a route in a driving simulator. *Journal of Environmental Psychology, 17*(4), 317-331.

Allen, G. L. (1981). A developmental perspective on the effects of "subdividing" macrospatial experience. *Journal of Experimental Psychology: Human Learning and Memory, 7*(2), 120-132.

Anderson, J. R. (2005). Human symbol manipulation within an integrated cognitive architecture. *Cognitive Science, 29*(3), 313-341.

Anderson, J. R., Bothell, D., Byrne, M. D., Douglass, S., Lebiere, C., & Qin, Y. L. (2004). An integrated theory of the mind. *Psychological Review, 111*(4), 1036-1060.

Appleyard, D. (1970). Styles and methods of structuring a city. *Environment and Behavior, 2*, 100-117.

Brooks, R. (1985). Visual map making for a mobile robot. *Robotics and Automation. Proceedings. 1985 IEEE International Conference on, 2*, 824-829.

Brooks, R. (1991). New approaches to robotics. *Science, 253*(5025), 1227-1232.

Chown, E., Kaplan, S., & Kortenkamp, D. (1995). Prototypes, Location, and Associative Networks (Plan) - Towards a Unified Theory of Cognitive Mapping. *Cognitive Science, 19*(1), 1-51.

Cohen, R., & Schuepfer, T. (1980). The Representation of Landmarks and Routes. *Child Development, 51*(4), 1065-1071.

Colle, H. A., & Reid, G. B. (1998). The room effect: Metric spatial knowledge of local and separated regions. *Presence-Teleoperators and Virtual Environments, 7*(2), 116-128.

Cornell, E. H., & Hay, D. H. (1984). Children's Acquisition of a Route Via Different Media. *Environment and Behavior, 16*(5), 627-641.

Cornell, E. H., Heth, C. D., & Alberts, D. M. (1994). Place Recognition and Way Finding by Children and Adults. *Memory & Cognition, 22*(6), 633-643.

Darken, R. P., & Peterson, B. (2001). Spatial Orientation, Wayfinding, and Representation. In K. Stanney (Ed.), *Handbook of Virtual Environment Technology*. New Jersey: Laurence Erlbaum Associates.

Epstein, S. (1997). *Spatial Representation for Pragmatic Navigation*. Paper presented at the Spatial Information Theory - A Theoretical Basis for GIS, International Conference COSIT'97.

Filliat, D., & Meyer, J. A. (2003). Map-based navigation in mobile robots: I. A review of localization strategies. *Cognitive Systems Research, 4*(4), 243-282.

Fitt, P. M. (1954). The information capacity of the human motor system in controlling the amplitude of movement. *Journal of Experimental Psychology, 47*(6), 381-391.

Franz, M. O., & Mallot, H. A. (2000). Biomimetic robot navigation. *Robotics and Autonomous Systems, 30*(1), 133-153.

Franz, M. O., Schölkopf, B., Mallot, H. A., & Bülthoff, H. H. (1998). Where did I take that snapshot? Scene-based homing by image matching. *Biological Cybernetics, 79*(3), 191-202.

Gale, N. D., Golledge, R. G., Pellegrino, J. W., & Doherty, S. (1990). The Acquisition and Integration of Route Knowledge in an Unfamiliar Neighborhood. *Journal of Environmental Psychology, 10*(1), 3-25.

Gibson, J. J. (1979). *The ecological approach to visual perception*. Boston: Houghton Mifflin.

Goldin, S. E., & Thorndyke, P. W. (1982). Simulating Navigation for Spatial Knowledge Acquisition. *Human Factors, 24*(4), 457-471.

Golledge, R. G., Gale, N. D., Pellegrino, J. W., & Doherty, S. (1992). Spatial Knowledge Acquisition by Children - Route Learning and Relational Distances. *Annals of the Association of American Geographers, 82*(2), 223-244.

Golledge, R. G., Ruggles, A. J., Pellegrino, J. W., & Gale, N. D. (1993). Integrating route knowledge in an unfamiliar neighborhood: along and across route experiments. *Journal of Environmental Psychology, 13*(4), 293-307.

Gopal, S., Klatzky, R. L., & Smith, T. R. (1989). Navigator: A psychologically based model of environmental learning through navigation. *Journal of Environmental Psychology, 9*(4), 309-331.

Heft, H. (1979). Role of Environmental Features in Route-Learning - 2 Exploratory Studies of Way-Finding. *Environmental Psychology and Nonverbal Behavior, 3*(3), 172-185.

Kuipers, B. J. (1977). *Representing Knowledge of Large-Scale Space* (No. TR-418). Cambridge, MA: Massachusetts Institute of Technology.

Kuipers, B. J. (1978). Modeling spatial knowledge. *Cognitive Science, 2*(2), 129-153.

Kuipers, B. J. (1983). *Modeling human knowledge of routes: Partial knowledge and individual variation.* Paper presented at the Proceedings of the National Conference on Artificial Intelligence (AAAI-83), Washington, DC.

Kuipers, B. J., & Levitt, T. S. (1988). Navigation and Mapping in Large-Scale Space. *Ai Magazine, 9*(2), 25-43.

Levenick, J. R. (1991). NAPS: a connectionist implementation of cognitive maps. *Connection science, 3*(2), 107-126.

Logan, G. D. (1988). Toward an instance theory of automatization. *Psychological Review, 95*(4), 492-527.

Lovett, M. C., & Anderson, J. R. (1996). History of success and current context in problem solving. *Cognitive Psychology, 31*(2), 168-217.

Luchins, A. S., & Luchins, E. H. (1959). *Rigidity of Behavior: A Variational Approach to the Effect of Einstellung*. Eugene, Oregon: University of Oregon Books.

Lynch, K. (1960). *The image of the city*. Cambridge, Massachusetts: The M.I.T. Press.

Madhavan, R., Fregene, K., & Parker, L. E. (2004). Distributed Cooperative Outdoor Multirobot Localization and Mapping. *Autonomous Robots, 17*(1), 23-39.

Marr, D. (1982). *Vision*. San Francisco, Ca: W. H. Freeman.

Mataric, M. J. (1992). Integration of representation into goal-driven behavior-based robots. *Robotics and Automation, IEEE Transactions on, 8*(3), 304-312.

Newell, A., & Simon, H. A. (1972). *Human problem solving*. Englewood Cliffs, NJ: Prentice-Hall.

Owen, C., & Nehmzow, U. (1998). *Landmark-based navigation for a mobile robot.* Paper presented at the From Animals to Animats, Proceedings of SAB'98, Zurich, Switzerland.

Schweizer, K., Herrmann, T., Janzen, G., & Katz, S. (1998). The Route Direction Effect and its Constraints. In *Spatial Cognition, An Interdisciplinary Approach to Representing and Processing Spatial Knowledge* (pp. 19-38). Berlin: Springer-Verlag.

Siegel, A. W., & White, S. H. (1975). The development of spatial representations of large-scale environments. *Adv Child Dev Behav, 10*, 9-55.

Simon, H. A. (1957). *Models of man: Social and Rational.* New York, NY: Wiley.

Smith, L., & Husbands, P. (2002). *Visual landmark navigation through large-scale environments*. Paper presented at the EPSRC/BBSRC International Workshop on Biologically-Inspired Robotics: The Legacy of W. Grey Walter.

Steck, S. D., & Mallot, H. A. (2000). The role of global and local landmarks in virtual environment navigation. *Presence-Teleoperators and Virtual Environments, 9*(1), 69-83.

Taatgen, N. (2007). The minimal control principle. In W. D. Gray (Ed.), *Integrated models of cognitive systems*. New York: Oxford University Press.

Taatgen, N., van Rijn, H., & Anderson, J. R. (2004). Time perception: Beyond simple interval estimation. *Proceedings of the Sixth International Conference on Cognitive Modeling*, 296-301.

Tlauka, M., & Wilson, P. N. (1994). The Effect of Landmarks on Route-Learning in a Computer-Simulated Environment. *Journal of Environmental Psychology, 14*(4), 305-313.

Trafton, J. G., Schultz, A. C., Perznowski, D., Bugajska, M. D., Adams, W., Cassimatis, N. L., et al. (2006). Children and robots learning to play hide and seek. *ACM SIGCHI/SIGART Human-Robot Interaction*, 242-249.

Van Maanen, L., & Van Rijn, H. (in press). An Accumulator Model of Semantic Interference. *Cognitive Systems Research*.