TECHNISCHE
UNIVERSITÄT
WIEN

VIENNA
UNIVERSITY OF
TECHNOLOGY

Diplomarbeit

Episodic Memory Based Self Localization and Orientation for Autonomous Agents

ausgeführt zum Zwecke der Erlangung des akademischen Grades eines Diplom-Ingenieurs unter der Leitung von

o.Univ.Prof. Dipl.-Ing. Dr.techn. Dietmar Dietrich
sowie
Dipl.-Ing(FH) Heimo Zeilinger
als verantwortlich mitwirkendem Assistenten am Institut:
384 Institut für Computertechnik

eingereicht an der technischen Universität Wien
Fakultät für Elektrotechnik und Informationstechnik

von
Robert Borer
Matr.Nr.: 0350967

Vienna, October 2009

_____

Abstract

The work is related to a self localization and orientation system for autonomous agents inspired by theories about the human mind. The aim is to provide spatial information in a robust and resource efficient way to reach desired goals faster. It is based on a combination of a network-map theory and a neuropsychoanalytical episodic memory model. This combination is realized by adding topological linkage to the experienced locations that are being recorded within the episodic memory. The purpose of this realization is to create an orientation system based on the human being as an archetype for artificial intelligent systems. Reason therefore is, that compared to other systems it provides a faster and robuster way of orientation and route planing. This would be suitable for applications that do not require a pinpoint localization, or for a kind of pre localization for other positioning systems to reduce their workload. Tests show that the system meets the expectations but in real world realization depends on qualitative sensor data.

Kurzfassung

Die Arbeit befasst sich mit dem Design und der Realisierung eines Systems zur Selbstpositionierung und Orientierung für autonome Agenten. Basis hierfür bilden Modelle und Theorien der menschlichen Psyche. Das Ziel ist es, auf robuste und effiziente Weise Weginformationen zu liefern, um so Zielorte schneller zu erreichen. Hauptbestandteile sind eine Network-map Theorie sowie ein neuropsychoanalytisches Modell eines episodischen Gedächtnisses. Zweck dieser Arbeit ist es ein Orientierungssystem zu modellieren das auf den Charakteristika des Menschen basiert. Grundmotivation hierfür ist, dass im Vergleich zu anderer Orientierungssystemen das menschliche System schneller und robuster ist. Es ist verwendbar für eine Vorlokalisierung, um den Arbeitsaufwand für ein anderes Positionierungssystem zu verringern, oder für Anwendungen die keine punktgenaue Lokalisierung benötigen. Tests zeigen, dass das System die Erwartungen erfüllt, aber auch, dass bei Anwendung in realer Umgebung eine sehr starke Abhängigkeit von der Qualität der Sensordaten besteht.

Acronyms

API	Application Programming Interface
ARS	Artificial recognition system
ASM	Area semantic Memory
BW	Bubble World
ID	Identification
Id	Identity
LM	Landmark
LTM	Long Term Memory
MASON	Multi-Agent Simulator of Neighborhoods
PA	Psychoanalysis
PC	Perception

Contents

Chapter 1

Introduction

Today when we think about positioning and orientation, we nearly instantaneously think about technologies like GPS, the "Global Positioning System" [1], to get our exact location on earth in form of coordinates. It also has become very common to use some kind of route planing software to compute the best way to a desired destination and have a calm computer voice tell us exactly where to go. But models of a far more basic and far more important kind of positioning and orientation are about to be used in this thesis. Humans do not look at their GPS receiver and their software to check the right coordinates and direction if they want to go to their refrigerator and have some juice. For this every-day kind of task, humans, as animals use their memory about previous paths to check their current position and to find the necessary steps and actions to plan their way.

A computational implementation of such a memory based, human like orientation and self localization system is about to be designed in this work. The purpose is to give autonomous agents in some degree the ability to orientate with the same robustness like a human. By using such an approach, the self localization and the path decision can not only be based on basic distance data, but also on experience data. In connection with the ARS decision unit that is modeled using the definitions of psychoanalysis, this can be the emotional and drive data that was experienced earlier. Thus, the results become much more qualitative in terms of finding the correct location during self positioning as well as choosing the best route in path planning.

1.1 Motivation and Mission Statement

Technologies like the mentioned GPS are great and important tools for human kind, but it does not work in every environment. It fails for example when the location is within a building. Moreover, it is dependent on its large Satellite network in "Medium Earth Orbit" (MEO[2]) without which, no localization would be possible. So if one cannot use GPS in a house to get to the refrigerator to get a drink, and there was no memory based positioning, one would most certainly die of dehydration.

This rather radical example should make it clear why such an individual positioning system is so important. So it is one of the main goals to create a human like self localization and orientation system for AI applications, because of its robustness. Regarding to the description of Solms ([ST02], p. 160), the human mind stores every event in our live in an "Episodic Memory" and uses that memory all the time when performing different tasks. For example, it is this kind of memory we use when saying "I remember, that I had eggs for breakfast". It is a memory of past situations. And it is also this memory we activate when we want to move somewhere. We

[1] Satellite Navigation system developed in 1970 [GPS09]

[2] Between 2000 km and 35786 km; GPS is in 20200 km altitude

build a kind of inner map based on these memories which is used for path planing. For example, the memory of passing a green door before arriving in the kitchen. In cooperation with many others it can be re-used if we want to go to the kitchen again, because we remember that we saw the refrigerator there. One also localizes oneself based on these past informations, which is especially important, because, one has to know where he is before he can go to his destination. This is not a localization in form of coordinates like with GPS, but by knowing the current location in reference to other learned places. Nature has already perfected such a system within every one of us and we do now want to create a model of this great tool, and implement it into artificial autonomous agents. The result should be a human like orientation behavior within this artificial agent.

The goal of the model is not to influence the decision making of the autonomous agent in any way, but to give it additional information it can use to achieve its goals faster. As a consequence for the simulation described in 2.5.3, the system should alter the agents lifetime. This information might be the necessary actions to reach an energy source. Without that knowledge, the agent would try to find the energy by randomly going in a direction, and if it does no reach the destination within time, it "dies". This could be avoided by using information from a memory based orientation system.

A second reason for this model is, that, since it estimates the position within abstract areas, it is more resource efficient than common vision based localization systems, which use techniques like metric maps, search trees or the eigenspace model [AJL02]. Especially when acting in larger environment. The positioning within areas can be used for example if the Robots application does not require it to know its current location within little tolerance and it is sufficient to know the area it is located in. This efficiency is important for real world implementation within mobile robots. Of course there are already attempts to more efficient approaches. An example would be the segmentation into View-Invariant-Regions [SON96]. But the technique requires an a-priori knowledge of the environment to perform the localization, which leads to the third motivation for this model. It does not need predefined map data but has the ability to collect this knowledge during exploration. Not only in terms of recognizing new areas, but also by partly recognizing, tolerating and reacting on environmental change.

As already mentioned, localization in this model means identifying the region the agent is currently located in. If a more precise location estimation is required, the system could work as a kind of pre-localization to decrease the workload for a more complex positioning program. A similar example would be the Assisted Global Positioning System (aGPS), used in most mobile phones these days. This system performs a raw positioning using GSM-Localization, which determines the area of the base station the mobile phone is connected to, as first order localization. The GPS system then performs the exact localization much faster.

1.2 Background

Human orientation is a research topic that is still not fully understood. Thanks to technological advances in neuroscience, the neuronal activity in the hippocampal region due to self localization processes is partly known. Unfortunately knowing the processes in a computer does not lead to understanding the program that controls it. Therefore, theories have been developed that try to explain these findings (see [Spe50], [Web95], [Tol48] [FH94]). The one that is used in this work is the Byrne theory of topological network maps [Byr79], as it seems to be the most plausible one ([Bug97], p. 10). It describes a separation of the world into smaller areas defined by the surrounding elements or landmarks. Tests performed by Byrne show, that the memory about the

connection between these areas is very crude but that this little knowledge is sufficient for humans to orientate safely. In later research done by Moar and Carleton [MC82], the existence of a more vector like representation with additional direction and distance informations is postulated. This combination of network map and vector information is used in this thesis to form the world knowledge.
The second part is about the saving and organization of these informations which is done by implementing a human inspired memory system. Similar to the orientation, human memory systems are still under investigation but there exist several theories and models. Here the definitions and explanations of Tulving and Solms are used ([ST94], [ST02]). Part of their theory that is important here is the definition of semantic and episodic memory (see Section 2.4). While the first one holds learned fact information, the later one records experiences in form of events. This fact memory is used here for storing the area definitions while the experiences represent the remembered traveled path. Data from both is needed to perform the self localization and later on the orientation and path planning which creates the final output information in form of route directions to a desired goal.
There exist many different computational implementations of self localization and orientation systems (see Chapter 2), but only very few are comparable with the one that is to be designed in this work. In the already mentioned View-Invariant-Regions approach by Simsarian et. al. [SON96], a similar approach in terms of segmentation is used. Depending on wall edges in sight, the world is separated into regions that can be identified by the system. This is used mainly for self localization in terms of detecting in which region the agent is currently residing in. Thus, the work only complies with the first part, which is the segmentation theory, but the human oriented memory system for saving the area and path knowledge is not covered. This on the other hand is the main topic in [HDN03]. Wang Ching Ho et al. use an autobiographic memory system, which is similar to an episodic memory, for saving an agents path and giving it the possibility of redoing steps if necessary. They save events like a change in moving direction or the encounter of special objects like food sources. With this data, steps to reach a previously found food source for example, can be planed as future actions. This is very similar to the model described in this thesis but additionally a semantic memory system is used to save the knowledge about the area structure.
By combining both parts, a human like way of collecting spacial knowledge as well as the possibility of creating useful knowledge from it is available.

1.3 Connection with the ARS Project

In the future automation systems, the complexity is going to rise continuously due to the growing amount of input data provided by various kinds of sensors. It is expected, that common automation systems will not be able to handle and to process this massive amount of data. To react on this development, the Artificial Recognition System project (ARS), as described in 2.5, is heading on a new path in artificial intelligence. The team is working on the creation of a computational realization of the psychoanalytical model, postulated by the famous Austrian psychiatrist, Sigmund Freud. The goal is to create a computational implementation of modeled principals of the human mind. This should lead to human like behavior within the artificial agents which is being tested in a simulation environment called the *Bubble World*[3] (see Section 2.5.3). Since humans are also confronted with a huge amount of body sensor data that has to be processed, this principles should also help pushing future automation systems to a higher level of performance. The ARS model and the Bubble World simulation are described more thoroughly in Chapter 2.5

[3] also known as the Bubble Family Game (BFG)

However it is not only human decision making that is important. The ARS model tries to find a way to collect information in a natural human manner, that is used by the decision unit to even make a decision. Before using this information for decision making it is preprocessed by basic functions to create a more abstract piece of information (see [Vel09]). Based on the principles of Phrenology[4], Jerry Fodor created a theory about the modularity of the mind. This theory states, that the basic functions of the brain are functionally separated into modules that are not necessarily exactly localizable in a spatial manner.

In his definition these modules have the following properties:

Domain specific: responsible for only one kind of information like visual, audio or taste.

Inaccessible: meaning that the internal mechanisms are not seen from outside, like we cannot realize how our visual module processes the informations received from the eyes.

Informational encapsulated: the processing within a module is not affected by information from an other module, as our vision is not affected by out hearing.

Automatic: meaning it has no on/off button and it operates as soon as it receives input.

Fast: this property is self explaining, when you open your eyes, you get your view of the world almost instantly.

Innate: the performance of the module is inborn and cannot be developed.

Fixed neural architecture: means that there are neuronal systems associated with this module.

It would cause a long discussion whether the theory is right or not, but it seems appropriate for basic functions like the preprocessing of audio or video information, for smelling or also orientation as part of spatial information. The processing of this vital information, in form of an additional information provider for the ARS decision unit is about to be modeled and discussed in this thesis. The module has the purpose to process environmental information and information from the semantic and the episodic memory (see Chapter 2.4). It than provides the result, an information about the actual position and information about routes to momentarily desired places, to the decision unit. The route proposed by the system is not meant to be mandatory, but is used as an additional information source for the process of deciding the agents next steps.

The designed orientation system is being tested with the ARS implementation, using the "Bubble World" simulation. In these tests the memory creation, the ability of self localization and finally the guidance functionality is being tested. In the end it should not only help the agent to survive longer, but to generally reach it's desired destination faster than without such an orientation. Since in this virtual environment only the basic behavior can be checked, the reaction to real world data has to be evaluated otherwise. Therefore, an implementation in an e-puck robot is done (see Section 2.6). In this environment the influence of imprecise sensor values on the mentioned functionalities is analyzed.

[4]The science of identifying the function of particular Brain regions is called Phrenology. It was developed around 1800 by Franz Joseph Gall (March 9, 1758 - August 22, 1828) and was forced throughout the 19th century. Today it is called Localism [Phr09].

Chapter 2

State of the Art

Designing such an orientation and self localization system is an interdisciplinary work that contains localization, route planing, and memory theories. The following sections give a brief overview about the themes that are most essential for the thesis.
First of all, common approaches in self localization are being discussed. These will be map based, landmark based and probabilistic methods as well as the methods based on relative positioning. This part is described in more detail, because of its importance for the later work, as it includes the definition of a place by the means of segmenting the agents' world.
Also the Topological Network-Map is shown, which is a possible model of the human spatial memory organization performed by the Hippocampus. The exact details about the human spatial memory are not yet known in detail but the Topological Network-Map provides a reasonable theory.
To gather the necessary information about path finding, a short trip into the world of route planing is also covered in this chapter. Essential routing algorithms will be shown, especially the Dijkstra algorithm and extensions to make it more efficient.
After that the theoretical concepts of human memory are explained. Especially the episodic memory, as it is an important part for this work. Besides a description of memory models, the used implementation as well as the ARS project's model itself are shown. Moreover the used simulator, the Bubble World, is described because it provides the actual framework for the realization of the orientation model.

2.1 Self Localization

As already stated at the beginning, one has to know its current position, before beginning a journey. We can track the evolution of self localization far back in time. Not even to mention the human self localization, as theories about it are used within this thesis, that has been designed by evolution billions of years ago. But we have yet continued to develop several methods to reveal our position, like for example by using the stars as waypoint with the help of a sextant [Sex09] which is still common in nautical navigation as additional resource. But as it is in human nature, we not only try to improve ourselves and our concepts, like we did in this area of research by creating the Global Positioning System, but also to realize concepts within artificial agents. Here we are in the challenging position to take evolution into our own hands.

Over time there have been many different attempts to realize self localization within autonomous agents. Starting from a memory representation of the whole world to a representation using landmarks, as also done in this thesis. In the following sections the most relevant methods are discussed.

2.1.1 Map Based Localization

In the map based solution, a copy of the real world is saved in the agents', or robots', memory and the position is calculated by collating the actual surrounding with the memory map.

Method Overview

The agent creates a map representation of its immediate area of sight using the sensors it has at service. The resulting local map is trusted to be a part of the large world map it has stored in its memory. Afterwards the local map is being compared with the world map to find the position that results in the best match (see Figure 2.1). This position is considered to be the current position of the agent.

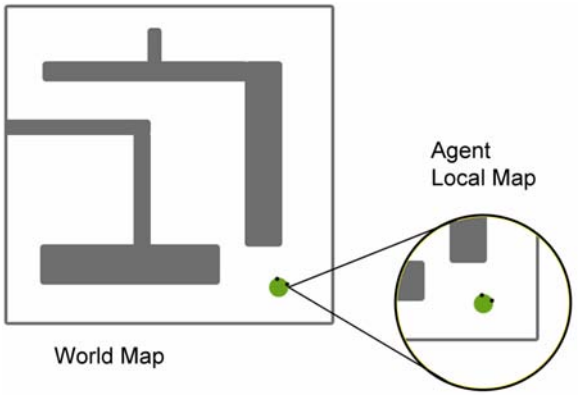


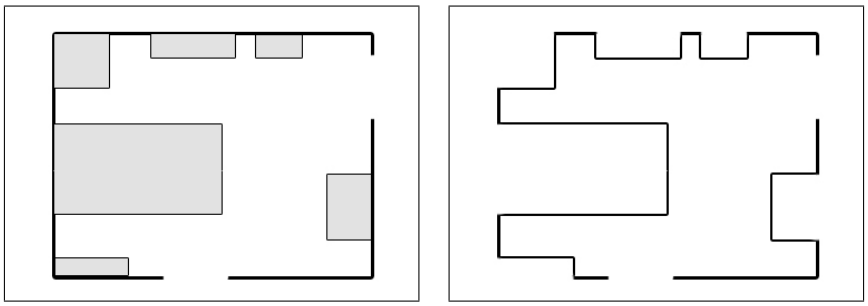
Figure 2.1: World map and extracted local map the agent is able to see

Memory Representation

Most challenging in map based localization is the organization of the world map representation within the memory. The accuracy of the topology information has to be [SN04]

- precise enough for the agent to fulfill its purpose,
- matched with the sensor precision,
- and matched to the available resources without resulting in exploding system complexity.

There are two main ways of memory representation. *Continuous* representation and *decomposed* representation, that adds some simplifications to the environment for easier processing



(a) D* example using standard map segmentation (b) D* example using quadtree map segmentation

Figure 2.2: Room feature extraction example

Continuous Representation This kind of approach saves a very precise 2-Dimensional map with the position of every environmental feature that exists. Such a feature extraction is restricted by the ability of the hardware. Thus a usable feature might be a surface or an edge that can be detected using a range sensor(see Figure 2.2(a),2.2(b)). So it is a suitable approach for sparse environment, as the required storage rises with the amount of environmental features. A major drawback is, that an extension to a 3-Dimensional representation would result in massive growth of computational complexity. But since the features are saved with their exact shape and their exact location with a continuous valued point in the coordinate system, this technique is highly accurate and can be used for precise localization. Of course this precision fades when using more discrete local coordinates.

Decomposed Representation The abstraction, or simplification of the real world results in the need of less computational expense, but also lowers the accuracy of the positioning. So the map representation and the kind of abstraction to use, strongly depends on the application. There are several ways to add simplification to the large world map. Siegwart & Nourbakhsh [SN04] list the decomposition strategies: *exact cell decomposition, fixed decomposition* and *topological decomposition*

Exact cell decomposition does a segmentation of the maps free space into areas, that are saved as independent nodes. The exact location of the agent within a single area is not determined. Only the node in which it is located is of interest. The agent can then use the node network for path planing. A detailed example can be found in [STG]. In this strategy the accuracy of the environmental features is not touched by the simplification. Hence, it is called a lossless process

Fixed cell decomposition is a discretization of the environment and therefore of the environmental features. It can be seen like reducing the amount but increasing the size of pixels on a computer monitor . A famous and simple representation is the occupancy grid. A matrix variable of binary values with the size of the discrete world areas. The matrix fields are initiated with "0", meaning that there is free space in this area. If the agent detects an obstacle in one of the areas, by using its range sensors, the matrix element that represents this area is set to "1". Depending on the world size and the grain size for discretization, the memory requirement is defined. This process can be enhanced by using adaptive cell decomposition, which uses different cell sizes for discretization. Smaller ones near a feature and larger ones in open space (see [SN04], p. 205). A kind of similar approach was done to improve the effectiveness of the D* path planing algorithm [YSSB98b] (see Section 2.3.2).

Topological decomposition, in contrast to the previous methods, does not record geometric features, but environmental characteristics. Using these characteristics, the world is segmented into areas, or nodes, similar to exact cell decomposition. The areas are then saved together with their connectivity. This approach is very similar to the topological network map (see Section 2.2).

Advantages/Disadvantages

The advantages and disadvantaged are rather obvious. With a complete environment representation it is easy to do path planing in a continuous step. This can be a very precise technique if high quality sensors are in use. Another much more important advantage is, that no additional

orientation aids like landmarks have to be installed.
On the other hand problems arise if no high precision sensors are available or if the agent acts in an environment that has a periodic topology, where several location look alike. In such cases map based self localization results in highly unreliable position information. Additionally, in contrast to the proposed system, this technique is very memory consuming. Because of this, and the fact that this results in heavy memory access, it is resource inefficient. Hence it results in poor operation performance within large areas.

2.1.2 Relative Position Estimation

Relative positioning, also called dead reckoning, can be compared to a human walking in the dark. The environment cannot, or is not, used to define the current position. Instead, the agents' movement is analyzed and recorded constantly. This way, it knows its current position relative to the point in space where it started the recording. Information that is used for the movement recording is coming from either the knowledge about the activation of movement actuators, like a bearing sensor attached to the wheels. In that case the method is called *Odometry*. The data can also originate from sensing information, like from acceleration sensors, gyroscopes, etc. This approach is called *Inertial Navigation*

Odometry: Standard Odometry utilizes information from the motion system, as for example from the wheels. If a position sensor in the wheel senses a turn of 10°, this information is calculated into real world movement of the robot using additional information like the wheel size. The essential fact is that action in the movement system is directly mapped to movement in real world. Here is where the system becomes unreliable, because facts like inaccurate wheel dimensions, unknown condition of the floor or bad wheel grip are sources for errors. As this is an incremental system, which means that the new position is based on the information about the old ones, the errors accumulate too and result in a heavily wrong localization. Therefore relative positioning generally is only used in combination with other systems for re-calibration of the current position. An other usage would be providing a pre localization for an other positioning technique like map based localization, which is then called multiple hypothesis positioning[SN04].

Standard odometry has been extended to *vision odometry* to make it also suitable for legged robots. Here the image flow is analyzed to calculate the current movement. Such a system is used in the "Mars Explorer" that is active since 2003 [CMM06].

Inertial Navigation: In contrast to odometry, inertial navigation does not rely on the motion system to be coherent with the actual robot move. Incoherence can be caused when the robot is not able to move as it is supposed to, like in the case of bad wheel grip. Instead inertial navigation uses inner sensors like an accelerometer, a gyroscope and a clock, to keep track of the robots' movements. The sensory information is used to calculate speed, traveled distance and heading direction, which can be added up to calculate the position. It is a more reliable approach than pure odometry, but still there is the problem of errors piling up due to imprecise sensor data and calculation. For the same reasons mentioned with odometry, this system is only used in combination with techniques like map based or landmark based (see Section 2.1.1 and 2.1.4), for calibration.

2.1.3 Probabilistic Methods

Similar to relative positioning, probabilistic methods are used in multi hypothesis positioning [SN04]. While other pre-localization systems do not add a probability of correctness to the

possible location area they propose, the probabilistic methods do so. This approach leads to higher effectiveness and hence additional research work was done.

The basic idea is to calculate the conditional probability, that the robot is in location $\mathbf{l_t}$ at time $\mathbf{t}$, depending on the information $\mathbf{i_{0...t}}$ about all the previous movements. This information $\mathbf{i_{0...t}}$ contains movement action $\mathbf{u}$, as in odometry, and perceived action $\mathbf{z}$, like in inertial navigation, coming from different sensors. This results in (2.1)

$$p(l_t|u_{0...t-1}, z_{0...t-1}) \tag{2.1}$$

Using the Bayes Theorem leads to (2.2)

$$p(l_t|u_{0...t-1}, z_{0...t-1}) = \frac{p(u_{0...t-1}, z_{0...t-1}|l_t) * p(l_t)}{p(u_{0...t-1}, z_{0...t-1})} \tag{2.2}$$

$p(l_t)$ can be associated with the probability of the location before updating the belief state, meaning the result of the previous calculation. The denominator $p(u_{0...t-1}, z_{0...t-1})$ does not depend on the location, but on the precision of the sensors, and is the same for all updates. The essential element is $p(u_{0...t-1}, z_{0...t-1}|l_t)$ which takes a special model for evaluation ([SN04], p. 216). Equation 2.2 is used to calculate a refined belief state. For receiving the new state based on the latest sensor data the information about all the previous states has to be considered.

$$p(l_t|u_{0...t}, z_{0...t}) = \int p(l_t|l_{t-1}, u_{0...t}, z_{0...t})p(l_{t-1})dl_{t-1} \tag{2.3}$$

Popular realizations are the *Marcov Method*, the *Kalman Filter* and the *Monte Carlo Localization* which are described in the following.

Markov Method: The Markov model is the basic method to apply probability to state estimation. It constantly updates its belief about its current position, by using its previous beliefs and the latest sensor information. With this information and an arbitrary probability density function, the robot receives a number of possibilities for its location. The amount of possibilities can range from a hundreds to millions. The core of the Markov localization is equation 2.2 and 2.3 which are called the Markov assumptions . It consists of a continuos perception update and action update using these two assumptions.

Kalman Filter: R.E. Kalman created a recursive state machine based on the Hidden Markov Model. His theory was, that the Markov model lacked in efficiency, because of its generality and that the real demand to localization was, to consider the properties of all the various sensor types used for robot localization([SN04], p. 227; [Kal60]). He called it a sensor fusion problem. For this fusion, 2 or more sensor readings are taken simultaneously, each with its own variance $s_1/\sigma_1^2 ... s_n/\sigma_n^2$. This data is merged, using the weighed least square technique equation 2.4

$$S = \sum_{i=0}^{n} w_i(q - q_i)^2 \tag{2.4}$$

The merged information q results of the calculation of minimum S by setting the derivate of S equal to zero and using the inverse variance as weight as shown in equation 2.5.

$$w_i = \frac{1}{\sigma_i^2}; \frac{\partial S}{\partial q} = \frac{\partial}{\partial q} \sum_{i=0}^{n} \frac{1}{\sigma_i}(q - q_i)^2 \tag{2.5}$$

This approach can be extended with the consideration, that the sensors are not read at the exact same moment. In that case, a time variable has to be added [SN04].
The real difference to the Markov localization lays within the perception update. While Markov uses all the sensor information for the update process, Kalman proposed a multistep approach. In a first step a prediction about the new position is generated. The environmental information of the predicted data is compared to the actual sensor readings. The best matches of this comparison are then being fused, and used for updating the believe state.

Monte Carlo Localization: The Monte Carlo Localization is applied when a topological representation of the states is used, because, in combination with Markov localization, the precision would leak massively. Monte Carlo Localization implements a new form of space discretization to deal with this problem. It uses a random distribution of samples from the state space [Flo05], [DDBT99]. Each sample is connected with a probability of being the real position of the robot, by comparing the sensor readings with reference information.
Figure 2.3 shows an example of the iteration process. First a random set of possible positions is created. Then, after the robot has moved, the new set of probable position is calculated by updating the elements of the old position set with the known movement information. The data set is then weighted with additional information about landmarks and the probability of seeing these from the position in process. $P(s_t|l_t)$; s_t sensor readings at time t. This iteration process continues using the resulting position set from the last processing

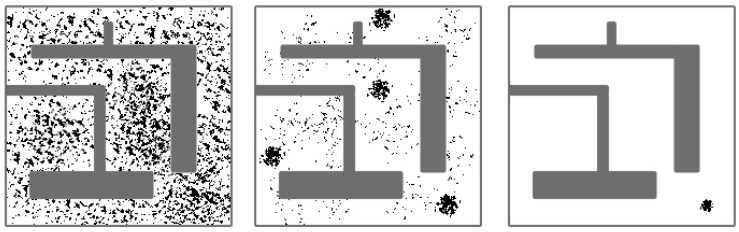


Figure 2.3: Example of the Monte Carlo localization iteration process

The probabilistic methods again use odometry and inertial navigation which has the property of accumulating errors, as already described in Section 2.1.2. Therefore it is not used in such a form in this work.

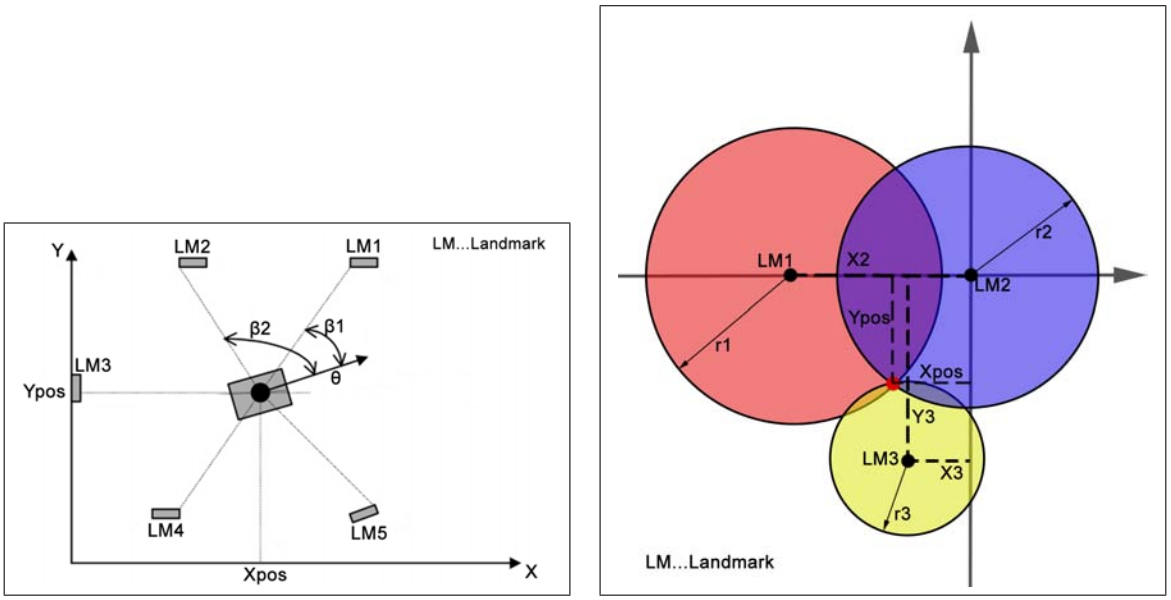
2.1.4 Landmark Based

Landmark based localization is a technique that relies on the identification of environmental reference points for positioning. With the identified references and the previously known global position of all the landmarks, the actual robot position can be calculated. The precision of this method is directly related to the quality of the used references and the kind of sensors that are equipped. These reference elements can be of various nature.

- *Natural references*, like doors, windows, room edges or lights. As no environmental preparation is needed, it is the most preferred method, but also the most difficult one, because of the complex feature extraction.
- *Artificial references*, like floor markings or specially installed elements. The landmarks can be placed on demand. Therefore this is the most accurate version of landmark based positioning.

- *Active references*, like ultrasonic beacons. When using active beacons, no vision based system is required which saves resources, but more expensive landmarks have to be installed.

Landmark based localization consists of two tasks. First of all, the landmarks have to be recognized as such. This is done with image feature extraction or feature recognition using range sensors. Here lies the advantage of the active beacon, as in that approach only the direction of the signal origin has to be measured, which is by far easier. The second task is related to the actual positioning which is again done in two steps. The first step is the calculation of the robots relative position to the landmark. In the second step the previously determined relative position is combined with the information about the landmarks absolute position. With this combined data the robots absolute position can be found. The actual calculation is done by using either *triangulation* or *trilateration*.



(a) Triangulation (b) Trilateration

Figure 2.4: Example of Triangulation[GL06]

In triangulation, the bearings to the different landmarks are measured and used. Figure 2.4(a) shows an example situation. For the position calculation in this 2-dimensional world, the angles $(\beta_1, \beta_2, \ldots)$ to at least three landmarks are necessary for receiving a unique position. Equation 2.6 is used for the actual calculation[GL06] of the position defined by the coordinates Xpos and Ypos. θ represents the robots orientation, (b_{xi},b_{yi}) define the absolute position of landmark i, and (x_l,y_l) is the robots position.

$$\beta_i = tan^{-1}\frac{b_{yi} - y_l}{b_{xi} - x_l} - \theta \tag{2.6}$$

Trilateration uses, information about the distances from the robot to the various landmarks. Also here a minimum of three landmarks is required for localization in 2-Dimensional space, and four for 3-Dimensional positioning. The reason for this is, that in 2-Dimensional space overlapping circles are used but spheres are formed in 3^{rd} dimension. While three circles, with different center, can only overlap in one single point, three spheres result in two possible points. Therefore the fourth sphere is needed to also receive a unique point of overlapping. Figure 2.4(b) shows an example for trilateration in 2-Dimensional space. In this case the available information is the distance to the landmarks (r1, r2 ,r3) as well as their coordinates like X3 and

Y3 for the position of landmark 3. Equations 2.7 - 2.8 describe the calculation of the actual position coordinates Xpos and Ypos. The basic idea is the same for the 3-Dimensional space.

$$x = \frac{r_1^2 - r_2^2 + d^2}{2d} \tag{2.7}$$

$$y = \frac{r_1^2 - r_3^2 - x^2 + (x-i)^2 + j^2}{2j} \tag{2.8}$$

Two problems arise from this approach. First of all, the position of the landmarks must not change, because it would result in constant error in positioning information. Moreover the precision of the calculated position depends on the deviation of the landmarks. If all of them reside close to each other, meaning nearly no deviation at all, the calculation will not be very accurate. The second uncertainty is noise in sensorial data, which is a similar problem in other positioning systems.

An important implementation of landmark based localization was done at the Stanford University [LL92]. Here the landmarks were realized as unique pattern printed on an ordinary sheet of paper that was placed on the laboratory ceiling with an average distance of 2m. The test robot was equipped with a single CCD camera oriented to the ceiling. With this test set, the robot was able to locate himself within a few cm.

2.2 Topological Network-Map

This model of biological spatial memory was first proposed in the article "Memory for urban geography" published by R. W. Byrne in 1979 [Byr79], and is one possible theory. Others would be the Stimulus-Response theory or the Cognitive maps theory.

Stimulus-Response theory: Orientation consists of a set of routes which have a list of landmarks along the way and for each landmark the information about the required action to get to the next landmark is saved. When a stimuli arises in form of a landmark in sight, the reaction, response, is performed according to the route information [Spe50] [Web95] [RF96].

Cognitive map theory: Is very similar to map based localization (see 2.1.1) where exact geographic information is saved [Tol48] [FH94]. There is evidence that this form is also used in human orientation as short-term information for a few seconds. Instantly after analyzing a room, humans are able to navigate throughout without collision even with the light turned off, meaning they have very precise information about the space. Solms ([ST02], p. 143) calls the memory which contains this accurate spatial information, the working memory.

Although many models were developed for biological orientation, "Topological maps stand the best chances to form the basis of a plausible model for biological spatial memory" [Bug97].
In this theory Byrne states that the mental representation of the world is separated into nodes with no exact geographical information and no distance data between each other. Instead of vector distance information, the topological connectedness of the nodes is saved. His work is based on experiments with students who ware familiar with a particular geographic region. He asked them to draw a line relative to a scale reference, that represents the walking distance between 2 locations [Byr79]. He summarized the results of this experiment in the following 3 points.

- The route between 2 points within a town are valued longer than one out of town.

- Routes that have more bends are also valued longer.
- Shorter routes are proportionally overestimated, compared to longer ones.

Byrnes second experiment that supports his theory was dedicated to human angle judgement. This time the students were asked to draw several junctions with great focus on the street angle. The results revealed the absence of angular information. The candidates judged nearly every angle, which were between 60° and 120° in reality, approximately at $90° \pm 5°$.
Based on these results Byrne stated that no distance nor precise angles were needed for orientation, and that the order of the locations was sufficient for navigation. He compared his map with the map of the London Tube, which has also no exact angular and distance information.
But Byrne does leave open the possibility of a hybrid form of network/vector maps, which would be supported by Moar and Carleton [MC82] who's experiments showed an improvement of direction judgement with growing experience. So this experience leads to a stronger vector like part within the cognitive map.
In the following the basic research results and believes of neuroscientists about human spatial memory in combination with topological network maps are described. Further on an example of robot implementation is shown.

2.2.1 Human Spatial Memory

Neuroscientists have postulated a strong connection between the spatial memory and the hippocampus for a long time. This was due to experiences with patients and animals suffering from lesions in the hippocampal region, and having trouble with spatial learning tasks. Modern technology made it possible to analyze the action of hundreds of neurons in parallel to get a more detailed view of the processes in the hippocampus. Using this technological advantage, the work of O'Keefe and Dostrovsky [OD71] could be carried on. In 1971 they observed a higher activity of specific neurons within the hippocampus, when a rat was put into a familiar environment. They named these neurons place cells. After more thorough investigation, neurobiologists realized that it was a specific combination of place cells that raised their activity in a specific previously known region. Tanila et al. [TSSE97] showed that these regions were described by landmarks, as the activation could be controlled by correct arrangement of obstacles. But as the place cell combination stayed active for some time after the room was darkened, it was clear, that the cell activity was not due to simple sensorial stimulation. This means, that the encoding of places into a place cell combination is a more complex process that works with highly processed sensorial data. Later theories add additional data to the place encoding, like information provided by head direction cells [Tau07] or grid cells. Research is still in progress about how the hippocampus processes and combines this information to encode the place. These encoded places fit into Byrnes theory of network maps. Figure 2.5 shows example data from a place cell,

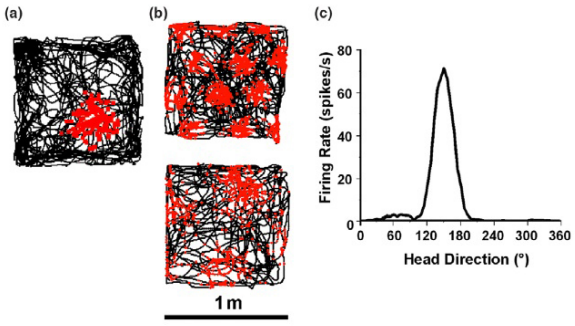


Figure 2.5: (a) Place cell reaction; (b) Grid cell reaction; (c) Head direction cell reaction. [Jef07]

a grid cell and a head direction cell recorded from a rat that was put into a 60cm square box. In Figure 2.5a the black line represents the rats path within the square. The places of high place cell activity are marked red. Figure 2.5b shows the same for the grid cell which is activated in several locations. The firing rate of the head direction cell for different head directions is shown in Figure 2.5 c. It can be seen, that this cell can be assigned to a small angular range. From this measured place cell one can get a picture of the world segmentation for the biological topology map. How these place cell combinations are connected to each other is though still in research.

2.2.2 Robotic Implementation of Topological Network Map

Although the details about the biological topology network maps are not solved yet, there exist several artificial implementations of this theory ([SON96],[Bug97],[BW94b]). The advantage, and therefore reason for using the method is its robustness. [BW94a] Common landmark or image based robot localization systems, as described in Chapter 2.1, are sensitive to viewing conditions, work intensive and provide no mechanisms to adapt to changing environment. The process as well as some real world implementations are shown in the following.
Topological network map creation consists of 2 essential steps, which is the "*Place Encoding*" and the "*Place connection*". The place encoding does the actual segmentation of the world into smaller areas, or scenes, as named by Byrne. The challenge in this task is to find references for a place that provide a unique encoding, to make sure the right area is remembered when it is reentered. This refers to the identification of the *what* and *where* [BW94b]. As landmark based localization, it has the strong requirement for landmark or feature recognition. Result of the recognition is, knowing "what", which landmark, is located "where", in which position from an egocentric point of view. The recording of these features can be implemented using various techniques. Starting with a 360° image capture, using a rotating camera, a conic mirror like done in [FSMB98], or with a 360° distance reading [RH96]. Every combination of "what" and "where" defines a landmark node. A set of them describes the actual scene as mentioned in [Bug97]; $S_i = \{(Landmark_{i,1}, Location_{i,1}), (Landmark_{i,2}, Location_{i,2})...\}$. This also means, that accuracy and the speed of the method depends on the quality of the "what" and "where" data. During traveling and data collection an ongoing list of states is being saved; $S_1, S_2, S_3....$ So the robot traveled from S_1 to S_2 and so on.
The next step is saving the connection of these states by recording every action that was performed between experiencing two different states; $a(S_1, S_2)$. Actions can be for example "turn left and right wheel for 3 x 360°", "turn left wheel 30°", and so on. In the human realization of a topological network map, the action description has a higher level of abstraction. Like describing to somebody the way from the Vienna University of Technology to the Stephansplatz. "Go across Karlsplatz to the Metrostation", "Follow the red signs to get to the U1 Metro", "Take the U1 in direction Kagran for 1 Station", "Follow the signs that say Stephansplatz to reach the exit". A more brief explanation of abstract human node connection description is found in [MTBF03].

Bugmann et al [Bug97] describe 2 different representations of topological network maps, called *sparse* and *distributed* representation, depending on the amount of nodes that describe a scene.
In the sparse representation only few nodes describe the scene. This could be one or more specific artificial landmarks that are set for this purpose. The purest sparse representation has a single node for each scene, which are connected with directional lateral links. An example for such a pure sparse representation can be seen in Figure 2.6(a). Here we have two sequent scenes, one defined by the node N1 and one defined by N2. These nodes are then linked to mark the connection between the scenes. An additional "transition" network records the actions between

two scenes which can be used for reactivation in forward and backward direction. In this linked list, forward planning, from S_1 to S_2, can be done by setting S_1 as the active scene and redo the following actions until the goal scene S_2 has been reached. Backward planing, from S_2 to S_1, on the other hand is done by activating the goal scene S_1 and propagate through the action list until the start scene S_2 is found. Of course these actions in between are redone backwards. The distributed representation encodes the scene into a set of nodes $N1$. An extra linker network learns to reproduce the S_2 from S_1, and the other way around, by activating the right set of nodes. Figure 2.6(b) shows this representation in a graphical way with the scenes $S_1 \dots S_4$. As in sparse representation, a transition network learns the action combination $a(S_1, S_2)$ and also the process of forward and backward planning is similar to the sparse representation.

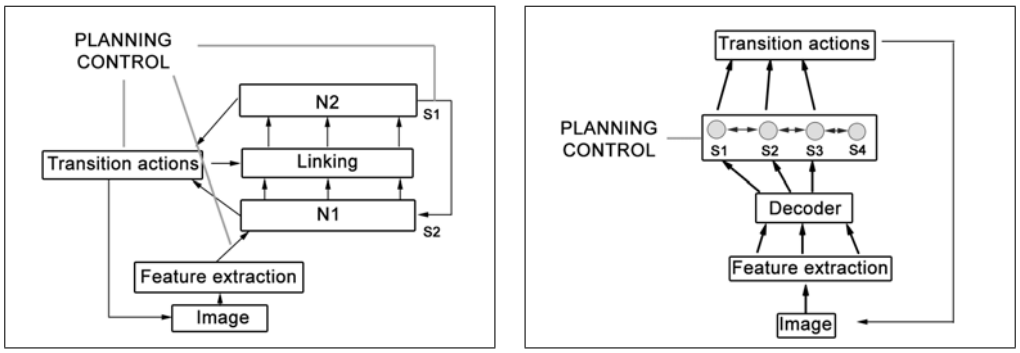


Figure 2.6: Sparse (a) and distributed (b) representation of topological network map

Examples of Computational Implementations

K.T. Simsarian et. al. [SON96] called their work about topological network map, "View-Invariant Region" (VIR) which is primarily related to "Place encoding". In their model they do a segmentation of a previously known world into scenes by using wall edges as landmarks. A scene is defined by the wall edges that are visible from that position. As the topology map is pre-computed the robots only task is to do the "Place encoding" to find the right scene within the list. The initial world topology map generation is done in four steps which are described in the following.

1. Do a *point visibility scan* for each map vertex to find view boundaries and vertex label sets, and add view boundaries to the map.
2. Calculate the interior vertices from view boundary intersections.
3. Traverse each view boundary to find label set for new vertices
4. Extract VIR's, which now includes the view boundaries, from the augmented map and compute VIR label sets from the vertex label sets

An example result of this process is shown in Figure 2.7 where a workplace with 10 edges is processed with VIR decomposition [SON96]. The resulting map then contains 23 View Invariant Regions (VIRs).

As the robot has to work with the information coming from a 360° range sensor scan, his process of "place encoding" differs from the pre-computation. The robot takes a 2 step approach. First a raw selection of possible actual scenes is performed, by counting the number of visible edges in the range sensor data, and compare the amount with the one saved in the scenes from the topology map. The second step is comparing the length of the visible perimeters with the

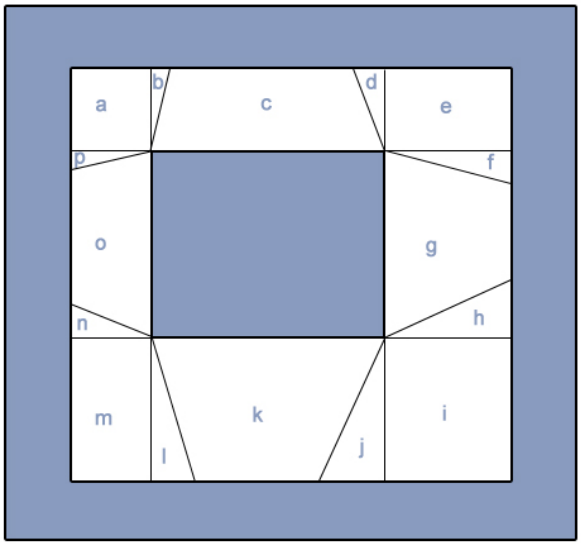


Figure 2.7: Decomposition result

ones saved in the list of possible actual scenes. The one that fits best is likely to be the correct scene.
More sophisticated approaches to the robots implementation are possible, but not proposed by Simsariasn et al. Also for route planing they refer to route planing algorithms as treated in Section 2.3.

Bachelder and Waxman [BW95] used lights as artificial landmarks and a neurocomputational implementation for network learning. The system is separated into two neuronal parts, *place learning and recognition* (PLA) and *action consequence learning and prediction* (ACLA)
The PLA uses a vision system that performs object recognition for artificial landmark exploration, the "what", and additional sensors for body and head orientation to receive the landmark direction, the "where". The dimension of the place region is defined by a threshold value of likeliness between the panoramic views and, as a consequence, between the sets of "what" and "where" combinations. For each place region there exists an activation profile that represents the similarity to the saved scene for this region. The place center has the highest activation and the intensity is dropping when leaving the midpoint (see [BW95], p. 277). The activation level at which these profiles overlap (approximately 60cm from the center in this realization) represents the threshold value for place recognition. As the overlapping is not always at the same activation level, this bears a small but existing probability of error.
The ACLA performs the construction of an aspect graph to save the transitions between the place regions. This is done by recording the motions in terms of forward movement and turnings, and combining it with the place region data coming from the PLA. Additionally to the aspect graph creation, the ACLA provides evidence for the place recognition system, as it has rough information about the geographic position of the place regions.
In contrast to the previous example their test robot named MAVIN (Mobile Adaptive Visual Navigator) performed place learning on its own by traveling along a defined path that guaranteed to cover the whole test region.

2.3 Route Planing

Route planing is the task of finding the optimal path through a network of nodes. This could be a street network in which a person has to find its way, or an information network like the

internet where the path of data packets has to be calculated. These connections between the nodes can be extended with additional information which is called a graph's *metric*. Depending on whether the metric provides data about the connection length or the connection costs, optimum path refers to the shortest or the cheapest one.
More sophisticated algorithms add a *heuristic* to their information set to decrease the size of their search area and thus to increase the searching speed. A heuristic is a function that provides information about the costs or the distance of a specific node from the goal node, based on additional data like GPS coordinates. This is used by the algorithm to primarily do calculation using the nodes which most likely lead to the optimum result.
But despite the nature of the network or the property the route should have, this is a basic problem of graph theory that has been worked on for a long time. During that time several solutions like *A*-, B*-, Bellman-Ford-, Best-first search-, Bredth-first search-, D*-, Depth-first search-, Dijkstra-, Floyd-Warshall-, Hill climbing-, Johnson's- or Uniform Cost search - algorithm* have been developed (see [CLRS09]). Three of these, namely the Dijkstra, A* and the D* algorithm, will be explained in the following.

2.3.1 The Dijkstra Algorithm

The Dijkstra algorithm, that was proposed by E.W. Dijkstra in 1959 [Dij59], is the most basic solution for finding the shortest path between two given nodes P and Q. It uses the maps metric to find the shortest path, which would be the optimal path in this case, but does not take advantage of any heuristic. For performing the algorithm the nodes are divided into three sets.

Set A : Contains the nodes for which the shortest path to the starting point P has already been calculated.

Set B : Contains the nodes that have at least one connection to a node in set A.

Set C : Contains all remaining nodes.

Based on these 3 sets the algorithm consists of 3 steps.

1. Starting point P is added to node set A
2. Though to the change of node set A, node set B is being updated and the node from the new set B that has the smallest distance to P, according to the metric, is searched and selected.
3. If the selected node is equivalent to Q the shortest path has been calculated. Otherwise, the selected node is added to node Set A and the process continues with step 2.

Using these steps the searching area grows in form of a search circle around the start node P until the goal node Q is reached. This inefficient growth results in rather poor performance and a time complexity that is a function of n^2, where n represents the number of nodes within the final circle [YJYQ03].

To deal with this inefficiency some extensions have been proposed. Noto and Sato [NS00] boosted the algorithm by starting it from both sides. From the starting node and the goal node, using the 4 node sets A_P, B_P, A_Q, B_Q. The process is finished when there exists a node that is part of node set A_P and A_Q. By using this approach 2 searching circles are generated with approximately half the radius of the ordinary Dijkstra search ($r_e = r_o/2$). Due to the smaller area, as can be seen from equation 2.9, it implies less nodes that have to be considered.

$$(r_o^2 \times \pi) > (2 \times (\frac{r_o}{2})^2 \times \pi) \tag{2.9}$$

In their tests using a network of 2000 × 2000 nodes, with start and goal node having a distance of 450 nodes, the ordinary Dijkstra took 3926,72 seconds. With the proposed extension the time could be lowered to 811,82 seconds, which is nearly a fifth.

An other method for improvement was developed by W. Yimin et al. [YJYQ03]. They used a hierarchical segmentation of the graph to decrease the number of nodes that have to be considered in the search. The calculation starts in a higher abstraction level with a graph whose nodes (N_{abst}) contain a more detailed subgraph with the detailed nodes (N_{det}). This can be seen analog to a connection of city districts as abstraction nodes and the actual street maps within these districts as subgraph. First the path from $N_{abst,start}$, that contains the $N_{det,start}$, to $N_{abst,goal}$, that contains the $N_{det,goal}$, is calculated. In a second step the paths within the N_{abst}'s, that are included in the previously calculated abstract path, are searched and combined to receive the final result. The method was tested in a graph with 10000 nodes (N_{det}) that was divided into 100 subgraphs (N_{abst}). Using this set, they observed an average search time of 0,2 seconds for shortest path calculation.

2.3.2 The A* and D* Algorithm

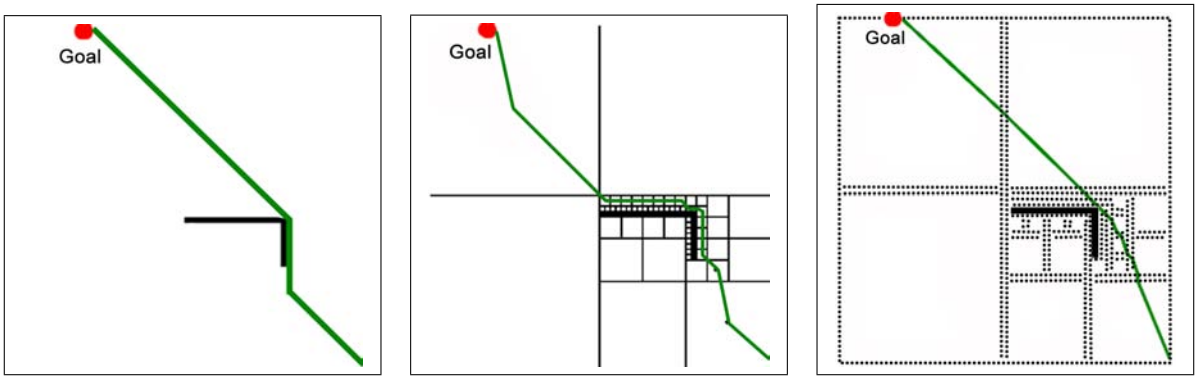
An other way to make the Dijkstra algorithm more effective, but also more complex, would be to add heuristic information, as mentioned in the overview. The combination resulted in the A* algorithm, proposed in 1968 by P. Hart et al.
Dijkstra's basic idea remains the same. The difference lies within the calculation of the node that has the minimal distance to the starting node (see "The Dijkstra algorithm" 2.3.1; Step 2). In this calculation the value $h(X)$ of node X that has been calculated by the heuristic function is added to $g(X, S)$, the calculated distance from the starting node S. The sum $f(X, S)$ is therefor calculated $f(X, S) = g(X, S) + h(X)$. The node with the smallest sum is selected and used next for calculation. This way the searching area does not expand in a circular shape because the nodes that are closer to the goal node receive a higher priority. It now expands like an ellipse that grows in the direction of the goal node.

The D* algorithm, which was developed by A. Stentz, is basically an A* algorithm that is able to deal faster with a dynamic graph metric. Therefore it was named "Dynamic A* algorithm" or D* for short [Ste94],[Ste95]. Dynamic changing in the map may occur, if the initial map information is not complete or not 100% correct. This could result in a situation where a path, calculated using for instance the A* algorithm, is blocked, for example by a road blockade (Road network graph), or a damaged network connection (Internet graph). If this is the case, the same algorithm could be used to recalculate the path in the updated graph. The problem is, that this process of recalculation can be very time expensive when using simple A*, and therefore the D* algorithm has been developed [Ste95].
D* works with the same node separation into 3 sets like the Dijkstra algorithm (see "The Dijkstra algorithm" 2.3.1). In the following explanation set B is called "Open set", and set A is named "Closed set".

The initial path calculation is done the same way as the A* but working from the goal node to the starting node. In this phase for every node X a key value, which is the minimum path cost (or length, depending on the desired metric) to the goal node G, is calculated and saved.

$k(X,G) = min(h(X,G))$. This key value is later on used to separate the nodes contained in the open list into two subgroups. "Raised" nodes, who's actual path costs are higher than the calculated key value, and "Lower" nodes who's actual path costs are equal to the key value. After the initial step all nodes within the open list are "Lower" nodes
As soon as a map change or blocking is sighted, the node that contains the blocking is added to the open set that is left from the initial calculation. This node is set as "Raised" and the nodes that have their optimal path to the goal via this blocked node are updated to the new costs (or length) and so also set to "Raised" and added to the open set. This new open set is now used for new path calculation starting from the remaining "Lower" nodes.

D* is mainly used for outdoor robot path planing. For this purpose the environmental map is segmented into connected nodes to form the graph. Depending on whether the node represents a free area that can be passed with little effort, or an area that contains an obstacle that cannot be passed, like a tree or a building, the path costs are defined low or high. An example of a planed path using such a standard segmentation is shown in Figure 2.8(a). The map is separated into nodes, each with a defined size. The highlighted green line represents the calculated path through the nodes.



(a) D* example using standard map segmentation (b) D* example using quadtree map segmentation (c) D* example using framed quadtree map segmentation

Figure 2.8: Standard D* segmentation and extension [YSSB98a]

With the purpose of optimizing the calculated path and to speed up the calculation itself, a new form of segmentation was introduced by A. Yahja et al. [YSSB98a]. This optimization merges some of the segments used in the standard segmentation and lowers the amount of nodes that have to be considered in calculation. This extension used *quadtrees* for area separation. When using quadtrees, a region is divided into 4 equally sized quadrants. As can be seen in Figure 2.8(b), if a quadrant contains an obstacle it is subdivided again. A quadrant of free space is not divided any more. When performing these simple rules down to a minimum segment size, an optimized graph is produced [YSSB98a].
In a further step, that had the purpose of path smoothing, the surroundings of the nodes were filled with minimum size nodes. This way much more heading directions are available compared to the standard segmentation that allowed only 8 directions $(0°, 45°, 90°, 135°, 180°...)$. This extension is called "Framed-Quadtree". Figure 2.8(c) shows that the "Framed-Quadtree" looks similar but has additional segmentations along the borders.

2.4 Human Memory Model

In this work the memorization and remembering of areas and paths is an essential part. Therefore a model of the human memory based on the findings of Baddeley [Bad97] and Tulving [Tul83],[ST94] and described by Mark Solms [ST02] is used. As this work will be integrated in a decision unit (see Chapter 2.5.2) that is based on Freud's 2^{nd} topographical model, it is important to find a memory model which can be connected with the psychoanalytic model. Hence, the neuropsychoanalytic approach is used in the ARS project as it tries to find connections between neurology, psychology and psychology areas of research. Solms bases this conclusion on different factors ([DFZB09], p. 20). For one, Freud did not focused on the physical representation of the psychic apparatus. Therefore technological development could not accelerate its development. With the now available deeper knowledge about neurology of the mental apparatus however, the theories about the unconscious can be re-conceptualized.

2.4.1 Overview

The capability to remember things is one of the most important ability of the brain, without which survival would not be imaginable. A complex network of neurons that forms our memory enables us to drive a car, to cook or to learn, and gives us the possibility of advancement. The knowledge about learned processes, experienced events or studied facts are being used in every day decision making and for common activities like walking.
In this model the human memory is separated in a timely manner according to the demands. In a raw classification it consists of a *short term memory* and a *long term memory* (see Figure 2.9). The term short-term memory is becoming less important among cognitive scientists because it is separated into immediate and working memory, as described in the following.

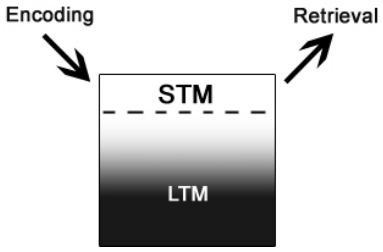


Figure 2.9: Short-term memory (STM) , long-term memory (LTM) [ST02], p. 143

Short-term memory: This is the memory where activities that are executed, and impressions that are observed at that very moment, or that lay back a few seconds in the past, are being saved. By focused thinking it is even possible to actively keep a thought even longer in the short-term memory. It also holds memories, that are recollected from the "long-term memory", when humans remember something.([ST02], p. 143).
There is a particular difference between actively keeping a thought in memory and the passive memory usage during perception of the environment. This difference bears an additional separation into immediate memory, representing the passive part, and working memory, as the active part of short-term memory. After a few seconds, or, when actively holding a thought, at least when going to sleep, the information held in the short-term memory is being filtered and transfered into the long-term memory, meaning it is being entrenched.
The physiological details about short-term memory is not entirely solved yet but it seems to build circuits with a feedback loop for self activation to keep the memory "alive". The

stronger the memory, the stronger the self activation and the better the consolidation, transfer, into the long-term memory. Better consolidation means that, because of the stronger self activation more synaptic connections are generated and the synapses develop a lower activation potential. So this particular memory can be activated by a less strong impulse and is easier to remember.

Long-term memory: Here, everything that lies back more than a few seconds is saved and available for remembering. Depending on the kind of data the long-term memory is separated into 3 parts, namely *semantic memory*, *procedural memory*, *episodic memory*.
The semantic part holds facts and concepts that we verbalize when saying "I know that...". This can be facts like "the earth is a sphere" or "voltage is resistance times current". In the procedural part, learned skills and abilities are stored. Examples would be the ability to drive a car or play Beethoven's "For Elise" on the piano. A trained pianist is able to play it without having to consciously remember every note. Also walking is something that was learned during childhood and saved into the procedural memory so we do not have to think about every movement when making a step. Finally the episodic part is used to save events and situations that have been experienced in the past. When remembering these elements we use the phrase "I remember...". Like for example "I remember that I had eggs for breakfast". This part of the long-term memory with its functions is discussed more detailed in the following section.

2.4.2 Episodic Memory

As explained previously, the episodic memory is part of the long-term memory and is used to save and recall encountered scenes and events. It is an autobiographical memory that gives us the ability to re-experience an event that took place in the past.
Events are defined as something that occurs in a specific situation but also contain emotions, feelings or drives. It has a defined start and end time ([Tul83], p. 142) and does have a temporal order in which the single events may overlap due to synchronous occurrence. According to Tulving, events can also be subdivided into simple and complex events. While simple events are short, like something dropping down, complex events have a larger duration, like a holiday trip. The model of episodic memory has 3 special tasks which are *encoding*, *storage* and *retrieval*.

Encoding: The purpose of the encoding process is to actually create a memory based on perceptional data of the event. The term episode refers to the ongoing process of encoding a series of events. This encoded event, or scene with all its features, is called engram. Due to the possible decomposition of the event into features the mind has the ability to remember similar scenes and use these previous experiences for learning.
A very crucial element in this process is the triggering. When does an event end and can be encoded. Unfortunately, as it is an unconscious process, it cannot be determined in an exact time, but it has to be done on a regular basis every time an event is transfered to memory.
There are theories whether a saved event is being modified when a very similar event occurs, or if a new engram is encoded. Tulving holds the view, that a new encoding takes place but both get a reference to each other and additional corresponding information is saved within the old memory ([Tul83], p. 42).

Storage: The function of storage is responsible for keeping the memory. Here the process of consolidation plays a big part. It does not only transfer memories from the short-term to the long-term memory, but also shifts them in deeper levels of memory organization and filters the ones that need not be kept. This means that, according to the rule "use it, or

lose it" ([ST02], p. 146), the synapses that formed these memories are being erased. On the other hand, synapses/memories that are activated more often are also activated with less effort, meaning they are remembered more easily. Activation is also easier when the memory contains a very strong feature, like an intensive emotion or a very rare and strong perception, like an accident.
This "forgetting" is an important point in memory. Freud and Tulving quote that something as forgetting is not possible. Memories are stored forever but the possibility of access vanishes with rare usage. Tulving separates this into the availability and the accessibility, whereas forgetting means that the memory is still available but not accessible any more ([Tul83] page 203).

Retrieval: With this function, recalling a specific past memory is possible. It may occur in an active way, when we actually try to remember something, or spontaneously when a thought just pops in our mind. The basic process remains the same. By using the encoded event that was just perceived, past events that show similar features are searched in memory. Retrieving is not done all the time but only if the mind is in a kind of retrieval mode. Being in this mode to remember things is only important in the episodic part of memory. The semantic part is in a constant retrieval mode. Associating a picture of an elephant with the word "elephant" is done without having to actively think about it.

2.4.3 Computational Implementations

In the following, two example implementations of the described memory model are shown. The first one is especially important since it is also used within the ARS model implementation.

Episodic Memory for Autonomous Agents

In [DGLV08] a computational model of an episodic memory is described, which was developed in connection with the ARS project shown in Section 2.5. This model, which was implemented in java code, is used as a part of the decision unit and works with the perceptional data coming from the simulation environment named "Bubble World" (see Chapter 2.5.3).

In this model the composition of the current situation is saved in events. These contain the environmental information in form of template images (example: there is an energy source near on the left side) and also information about the inner state. The inner state contains the energy level (hunger), the drive intensity (hunger, play, etc.) and emotions (fear, anger, etc.), as well as the current action that is performed.
Additionally to the events, the model defines scenarios which are a sequence of events that belong together. An example for such a scenario would be searching for food. This scenario contains several states that have to be experienced for reaching the goal. The start of a scenario is recognized with the occurrence of a specific event. In the example it might be the energy level falling below a defined threshold. Similar to the beginning, the ending is recognized by a defined event. Figure 2.10 shows an example of such a scenario consisting of 3 states that are reached with the occurrence of event 1 and event 4 (see also [DGLV08]).

An important point in this implementation is the triggering of the encoding of new events into the memory. It is realized in form of a priority value for the current event, that is calculated by doing a weighted summarization of the salience of the feature-sets (Template-Images, Drives, Emotions and Actions). This salience is also calculated considering the salience of the elements within the feature-set. In the case of an emotion element (fear, anger...) would be high when a strong change in intensity value, like a sudden strong anger, occurs. For environmental data,

the degree of matching with a template image is used and the salience of actions is calculated reciprocal to their endurance.
The implementation also considers "forgetting" by lowering its initially calculated salience over time. As it is also used for recalling a memory, this is similar to the previously described process of loosing accessibility to the memory.

For memory retrieval the whole sequence of recorded memories is searched and compared to the event that is currently in cue for encoding. The comparative results are ranked according to the likeliness of matching. The best match is returned. The model does not only provide the possibility of spontaneous and deliberative recall, but allows also to recall the previous and succeeding events from the recorded event-chain. This gives the decision unit the possibility to base its further actions on previous ones.

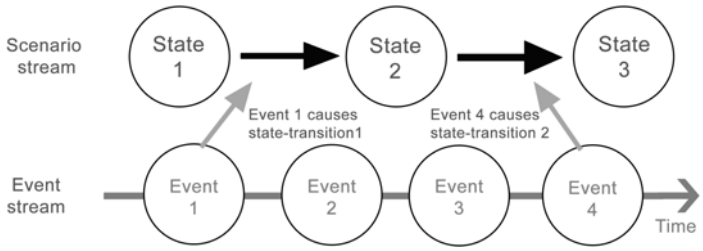


Figure 2.10: Event stream and scenarios [DGLV08] (event 2 and 3 are not important for advancing in the scenario)

Control Architecture for Autobiographic Agents

Wan Ching Ho et al. developed 2 different memory based control architectures which they tested in a special virtual environment created with the "Virtual Reality Modeling Language" (VRML).
In their *Trace-back* approach, a memory encoding of perception, preformed action and inner state is done. The encoding has an event-based trigger that starts encoding when the agent encounters a new object(energy source) or performs a change in action (turning right). As a second time-based triggering mechanism memories are encoded every fixed number of time-steps. This recording results in a list of actions that contains the agents' orientation and the traveled distance. Effects like "forgetting" are not considered in this model.
As soon as the inner state changes, for example to "hunger", a back-trace is performed. As the encoded action contains movement direction and distance information, every move can be reversed until a memory entry is found, that contains an encountered energy source.

An other approach developed by Wan Ching Ho et al. is the *Locality* memory agent. Here only the latest 4 events are being saved with an event based trigger. But additionally, a list is created that contains the action that has to be performed to get from one object to an other. Example: water - food; direction 33°, distance 20 meter. These entries can be used to reach an object on a direct way without having to perform unnecessary steps as in trace-back.

2.5 Artificial Recognition System

The Artificial Recognition System project, or ARS project for short, was launched in 2003 at the Vienna University of Technology. It started with the future vision of intelligent building automation systems, that will be able to handle the massive amount of data coming from the

continuously rising sensor count. Based on this mass of data the system should be able to make decisions on its own. In cooperation with psychologists, computer scientists do basic research on artificial intelligence by using the psychoanalytical model developed by Sigmund Freud as a foundation.

2.5.1 Overview

The core of the system in development is a powerful autonomous decision unit that will be able to manage unknown situations on its own and, in contrast to ordinary systems or human control, use the whole set of information available for making its decision. The result will be a system that reacts in a similar way as the human brain. The body, which is equipped with millions of sensors, also hands its information over to the brain which disposes based on this data. Inspired by the human, the ARS model is divided into 2 different systems. A system that receives and processes perceptional data, the ARS-PC (Perception), and one responsible for the decision making, the ARS-PA (Psychoanalysis), which is described later in this chapter [ZDML08], [DZL07], [DLP+06].

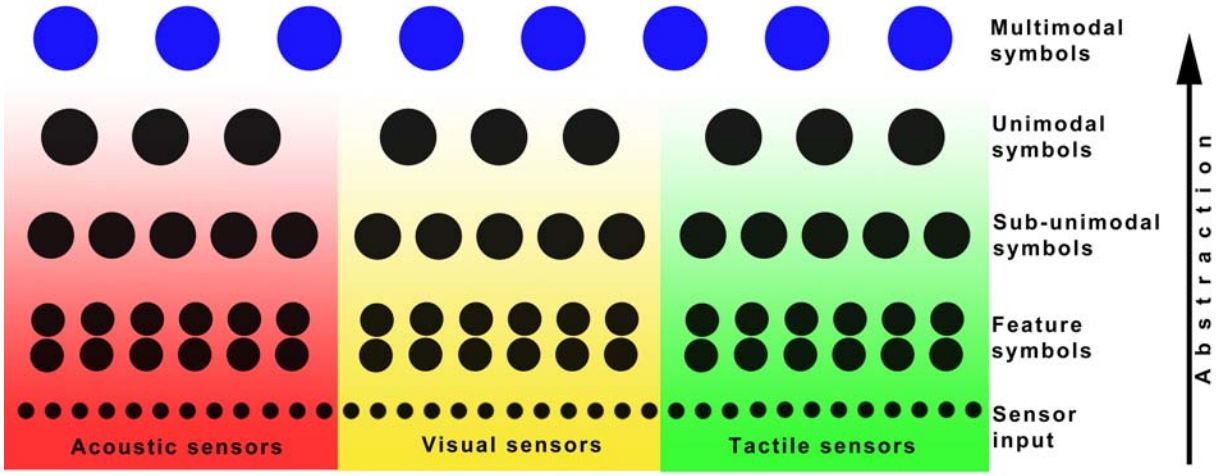


Figure 2.11: Example for a neuro-symbolic network architecture [Vel09]

The ARS-PC project [VLBD08] works on processing environmental information to get a more dense representation. The perceptional data that is being processed by this system might come from a movement sensor, a light barrier, a camera or any other kind of sensor. The task is to interpret the incoming intelligence to receive a higher level information. An example for this would be when a person enters a room, the light barrier installed at the entry and a movement sensor installed in the room sends a signal. The sum of these signals represent the low level information that should be processed to a more abstract piece of data that represents "a person entered the room". This can be seen as a preprocessing of information as it is performed by the human perception. R. Velik [Vel09] named this pieces of abstract information *neuro-symbols* and described their creation in a hierarchical model that uses the low level data as input. These neuro-symbols form a neuro-symbolic network [VB08] whose architecture can be seen in Figure 2.11. The basic inputs are combined to more and more abstract pieces of information represented by feature symbols, sub-unimodal symbols, unimodal symbols and multimodal symbols. The last one stands for the final interpretation of the data.

In the following sections the basics of Freud's model and the model developed by the ARS Project is explained. This presentation is followed by a description of the simulation environment, called *The Bubble World*, that is used for testing.

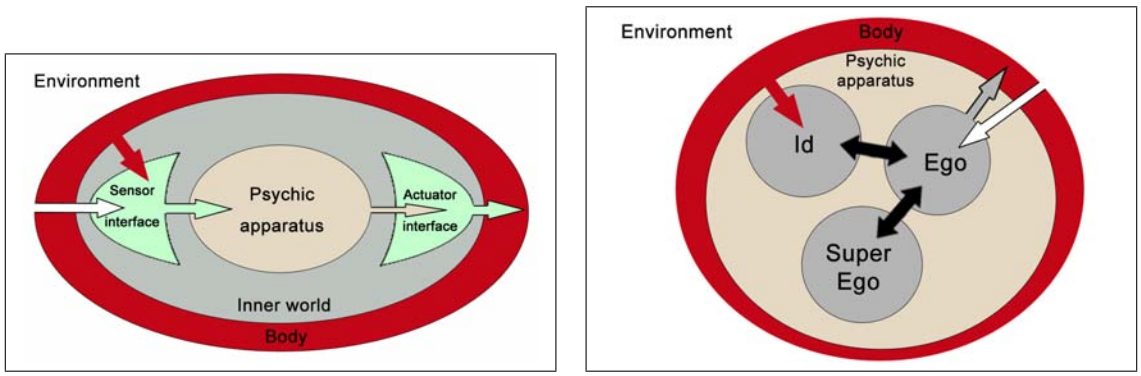
2.5.2 The Psychoanalytical Model

With the psychoanalysis Siegmud Freud created a theory about mental processes. Parts of this theory form the base structure for the ARS. Especially the 2^{nd} version of the topographical model. Therefore it is explained more thoroughly in the following.

Freud's 2^{nd} Topographical Model

In 1900 Freud defined the fundamentals of metapsychology with the assumptions in his publication "The Interpretation of Dreams" ([DFZB09], pp. 15 - 17). This concept became more specific in 1915, where he defines it as a system that reveals where in the psychic apparatus (topography), by what causes (dynamics) and at what costs (economics) mental processes occur. He associates the dynamics with the drives which are responsible for behavior. The economics are determined by a psychological energy that is necessary for and regulates behavior and is not to be mistaken with a technical form of energy ([DFZB09], pp. 71 - 72). Finally he defined the unconscious, the preconscious and the conscious as the locations in his first topographical model. In a later version, also called the structural model or second topographical model, he makes Ego, Id and Super-Ego the actors of personality.
The Id holds the characteristics of the unconscious part that was defined in the first topographic model while the Ego includes organizing integrative and synthetic functions. All ideals and prohibitions are kept in the Super-Ego part which seems to dominate the Ego.

The ARS Psychoanalysis Model in Detail



(a) (b)

Figure 2.12: (a) Information flow among environment, body and brain; (b) Information flow within *Psychic Apparatus* [ZDML08]

The psychoanalysis part of the project deals with the actual decision making process based on the available internal and external data provided by the perception, drives, memory , etc. The first ARS-PA model [DLP+06] included parts of different theoretical models and was therefore not entirely consistent. For this reason a second version is currently under development that relies solely on the previously described Freudian 2^{nd} topographical model. For that reason the ARS version two consists of the three major parts, "Ego", "Id" and "Super Ego". The basic information flow within the whole system, and especially within the *Psychic Apparatus*, which actually contains the ARS model, is shown in Figure 2.12(a) and 2.12(b).
In Figure 2.12(a) the information flow through body and brain is displayed. The basic information is received by the sensor interface from the environment as well as from the body itself. This information is processed by the psychic apparatus which then causes actions to the body through the actuator interface.

Figure 2.12(b) shows, that the "Ego" as well as the "Id" receive the previously mentioned available data, but only the "Ego" has the authority to engage an action by sending requests to the body's actuator system.
Internally the "Id", which responds to drives like hunger and the resulting demand for food or energy, suggests a process to fulfill these basic needs in order to balance the internal demand variable-set. It is a purely reactive system that is activated when a physiological stimuli arises, meaning a demand reaching a certain threshold, and does not care about the consequences the proposed action would cause. The upcoming activation of drives, which are actually interpretations of bodily stimuli, are being evaluated by a quantification system, that is included within the "Id", and passed over to the "Ego". Depending on the intensity of the drive, the "Ego" will handle this process or not. But before causing any action, the "Ego" consults the "Super Ego" which gets detailed information about the situation from the "Ego". The "Super Ego" itself holds and manages a set of demands, ideals and social restrictions in the form of memory traces that have been collected during simulation time, or predefined situations within a database. Based on this available information the "Super Ego" will give feedback to the "Ego" who, after collecting both evaluations about the situation, has to find a consent between the "Id" and the "Super Ego". This "finding a consent" is the actual decision making part after which the resulting action is forwarded to the movement control system which is also contained within the "Ego" [ZDML08].

2.5.3 Simulation Environment

The simulation environment, called *Bubble World*, is a virtual playground which, as well as the decision unit, was designed in Java, using the provided libraries. Additionally a framework named MASON [MAS09], a multi agent simulation environment with extensions for physics simulations, is used. This physics engine is needed to simulate the interaction between the body and the environment and thus to provide the external sensor information. The bubble world replaces its previous version, the *Bubble Family Game* (BFG) [RLVF06].
Its purpose for the project is to provide the required conditions to test and evaluate the proposed decision unit, when implemented in an autonomous agent. An autonomous agent is defined as an independent system that is able to autonomously interact with its environment to perform tasks, follow its own agenda and effecting its future sensing [FG97]. Therefore the Bubble World simulates the required external and internal stimuli as well as the necessary environment.

Stimuli: *Vision*, needed for recognizing objects and other agents in range, *acoustic* for communication with other agents, *smell* for detecting areas and other agents and *movement feedback* provided by a physics engine for detecting a collision with obstacles or other agents. As for internal stimuli, *odometry* used for recognizing its own movement and position, as well as a *health state*, which refers to the agents energy level, have been implemented. In a later version, currently under development, this list is being extended by more stimuli.

Environment: The simulator creates a virtual world with various *landscape* that consists of *passive entities* like different energy sources, one that can be consumed alone, and one that needs help from other agents for consuming. *Active entities* are the agents themselves which are called simple or cognitive agents depending on the set of inner stimuli they are equipped with. The second kind of environmental elements are *obstacles* which have different properties like for example "moveable". Figure 2.13 shows how these different elements are grouped.

In this environment the agents main goal is to stay alive as long as possible by consuming energy sources if necessary. For evaluation purpose several agents with different sets of

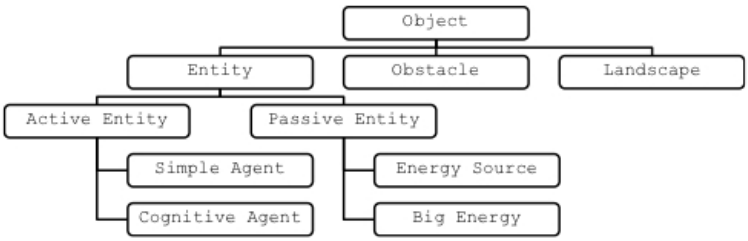


Figure 2.13: Implemented object types in Bubble World Simulator

emotional and memory features are deployed in the simulation and their behavior and survival time is monitored. The agents are able to do defined actions, like *search food*, *get help*, *play*, *dance*, *fight*, *promenade* , etc. . These actions are performed depending on the decision that was made by the "Ego" based on the available information like emotion, drives, health or perception.

2.6 The e-Puck Robot

The system described in this work is also tested under real world conditions to test the influence of sensor error. Virtual testing is sufficient to check basic concepts and functionality [Pfe99]. This is why the implementation and testing in the virtual environment is dealt with more deeply in this work. For the additional factor of sensor error the e-Puck implementation is used. An e-Puck robot is therefore utilized as agent embodiment. The e-Puck robot [MBR+09] is a small robot system designed at the École Polytechnique Fédéral de Lausanne. Its purpose is mainly for educational and research usage. It is a wheeled robot driven by 2 step motors with position sensor output which can be used for odometry approaches (see Section 2.1). The whole system is controlled by a Microchip digital signal processor (DSPIC30F6014) which is mainly programmed in C. Using different plugins however, it is also possible to use move sophisticated programming languages and environments like Matlab or Java. Besides the already mentioned actuator motors, there exist several other interfaces with the environment.

Distance Sensors: The e-Puck is surrounded with an array of 8 infrared distance sensors (TCRT1000) which can be used for collision avoidance. Since the effective sensor range of about 3 cm is quite low, it cannot be used for long range environment scanning.

Lights: An array of 10 LED's (Light Emitting Diode) are circled around the robots' corps. Their purpose is for lighting up the environment or for simple displaying an inner status.

Color Camera: The most important sensor however is a small camera mounted at the front. In combination with the 8 kByte internal flash memory of the processor it is possible to gather color images with a resolution of 40*40 pixel. The update rate with this setting is 4 images per second.

Accelerometer: A MMA7260 accelerometer sensor is implemented which can be used to assist for example the odometry.

For the use with extensions like a USB interface or additional cameras, a serial data connection is available. As an additional communication interface with other robots or with a base station, it is also equipped with infrared remote control and a bluetooth chip which, in combination with a boot-loader, can also be used for programming.

Webots

Webots is a simulation and programming environment for different kinds of robots available for Windows, Linux and Mac OSX. It consists of a set of libraries for robots like Aibo, Mindstorm, Boiloid, Kherpa and many others. Among these also the e-Puck robot. It gives the possibility to create a virtual environment in which a model of the robot can interact with its available sensors and actuators. While most of the environmental conditions and robot properties like lighting, material reflection, sensor certainty can be configured, a basic error like wheel slippery always remains. Besides the simulation part it consists of a development environment and a C compiler for the robots. In combination with the mentioned libraries all the robots abilities are supported. As also mentioned before plugins for Matlab programming are provided
As in the case of the e-Puck, Webots also consists of a special firmware that can be loaded to the robot. By using this feature it lets the user load the program directly to the bots' memory via the Bluetooth connection and have it operate in parallel to the simulation. The wireless connection is then used to transfer debug and output data to the computer. All these possibilities make it very easy to work with the e-Puck and gather testing data.
Figure 2.14 shows the simulation environment of the Webots interface.

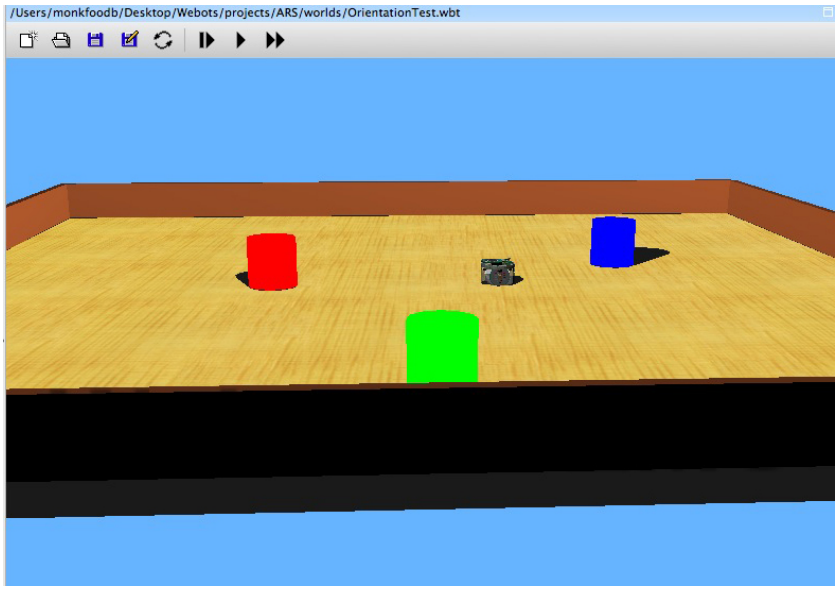


Figure 2.14: Webots programming and simulation interface

Chapter 3

Model

The purpose of this work is the design and implementation of a human like self localization and orientation system. In the following chapter the design in form of the model is described in detail. It is shown in 2 separated parts. One dedicated to the self localization and one to the orientation system. As it is being implemented within the ARS project it is also based on concepts of the human mind.

3.1 Self Localization

Self localization is the first large part of this model. The goal is to encode places which the agent encounters during traveling, and to recognize locations it has already visited. The recognition is not only based on the environmental information but also uses emotions and drives to identify a certain place if necessary. Moreover the model does tolerate a certain degree of environmental change and is able to adapt to it.
The base for this part is formed by the concept of topological network maps (see Section 2.2) to create the representation of the environment. It defines how the knowledge for the representation is gathered. To comply with Byrnes' concept [Byr79], it is done by segmenting the environment into smaller areas (see Section 3.1.1) and by linking them with experienced connections (see Section 3.1.1). To stay conform with the model about the human mind, this information is saved within implementations of the memory systems described in Section 2.4. The actual self localization in terms of knowing in which area one is, is done with a remembering system (see Section 3.1.1). Similar to the human self localization, the remembering model uses emotional and drive data for location estimation. To meet the requirement of tolerating a certain amount of environmental change, an assumption system (see Section 3.1.3) is defined that keeps the knowledge up to data.
The module is designed to work with the decision unit defined by the ARS project (see Section 2.5). Therefore it utilizes the provided vision system as well as the emotions and drives system, to gather the necessary information to work with. It will give the autonomous agents the ability to locate themselves within the Bubble World Simulation (see Section 2.5.3). This ability will be important for the second part of the model, the orientation.

The part of self localization starts with the encoding, which deals with the mapping of the current information to an area. It also describes how the movements between the areas are being recorded and shows how these areas and transitions are saved within the memory.
In a second part, the retrieval is concerned with the remembering of areas and mentions the handling of dynamic change.

3.1.1 Encoding

Encoding is the very first thing that is done in this model. It uses the information about encountered landmarks to define the area the agent is located in. The area is then connected with information about the current drives and emotions. In the following the term area is defined as the union of locations who's encoding have the same result when adding a specific tolerance to the input values. This means that if the environment in two positions looks alike within a certain range of tolerance, the positions belong to the same area.
The task of encoding consists of two different subtasks. One is the actual world segmentation, the separation of the environment into the different areas based on the mentioned information. The other is the recording of the transitions between the areas which is a kind of dead reckoning system (see Section 2.1.2). With that information the connections between the areas are known and a rough overview of the world is available. So the encoding part manages a detailed list of known areas and creates a network map by recording the traveled path. The area list is used later on by the retrieval module for the self localization process. Whereas the orientation module utilizes the network map to plan paths to a specific area.
For the storage of the area list and network map, the encoding module cooperates with two different memory systems. One represents a semantic memory and the other works as episodic memory (see Section 2.4). The semantic memory holds the detailed area list, which means knowledge about which areas are known and how they look like. The episodic memory on the other side saves the emotional data during the journey together with the transitions and links them to the area the recorded event occurred in. This way areas can be associated with feelings and drives.

Area Segmentation

Area segmentation has the important task to map the gathered environmental data into a unique area. This means it separates the world into smaller pieces, that are called areas, based on the objects that are in the agents range of sight. The input for the process is the environmental data provided by a vision system. This data consists of the position of landmarks relative to the agents' position. Its precision together with a defined tolerance is directly related to how many different areas exist and thus in how accurate the positioning is.

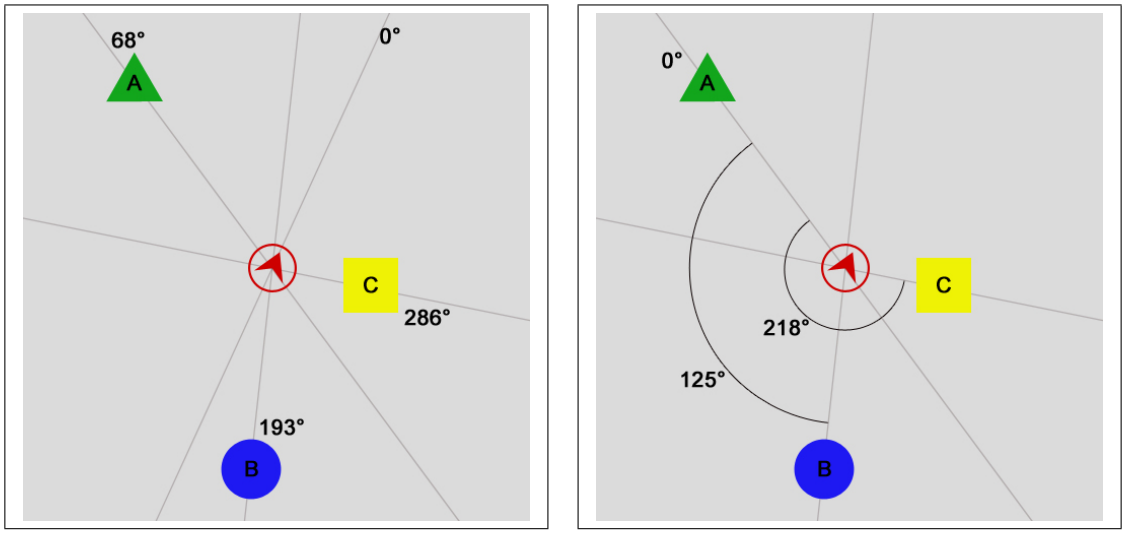
The areas are mainly defined by the environmental information, meaning by the landmarks. These landmarks are objects that are most likely to be stationary, meaning they will not move in near future (see Section 2.1.4). A human analogy would be that a street sign is a more preferred landmark than a car. As the car might be gone or has changed its position. In the used simulation there is a clear separation between stationary and movable objects which eases a lot. In a real world implementation an object recognition system would have to define a kind of stationarity property to the objects. Emotions and drives provided by the decision unit are the second part of the input data. They are saved within an event of the episodic memory, together with an assigned ID number that identifies the encoded area. In case the environmental data is not sufficient for identifying an area, these feelings are used as additional filtering criteria.

The environmental data set provided by the vision system is called a perception list. It has the form of an array that contains the name or ID number of the recognized object, its distance to the agents' position and the position angle relative to the agents' heading direction. An example can be seen in Figure 3.1(a). In this situation the agents' vision system has identified 3 landmarks named A, B and C. The perception list containing more precise information about the relative positions of the landmarks in this case can be seen in Table 3.1.

ObjectID	*Angle*	*Distance*
A	68°	38 *mm*
B	193°	24 *mm*
C	286°	15 *mm*

Table 3.1: Perception list example

The first step in encoding the area is to normalize and filter the perception data. Filtering here means the separation of landmark objects and non landmark objects like food sources. This is done with the objects property of being stationary or movable. As will be mentioned later, the categorization is a requirement to the object recognition. In the following process only the landmark objects are used to define the areas. The normalization is done by selecting a *main object* from the landmarks and referencing all other angles to this relative zero angle like shown in Figure 3.1(b). The reason is not to require a compass like reference angle. It can be seen like a compass that does not point to North but to the main object of the current area. Doing so makes the system more independent and robust because in a real world realization such a reference angle sensor would mean an additional source of error. By performing a normalization the area looks the same, independent of the viewing direction.

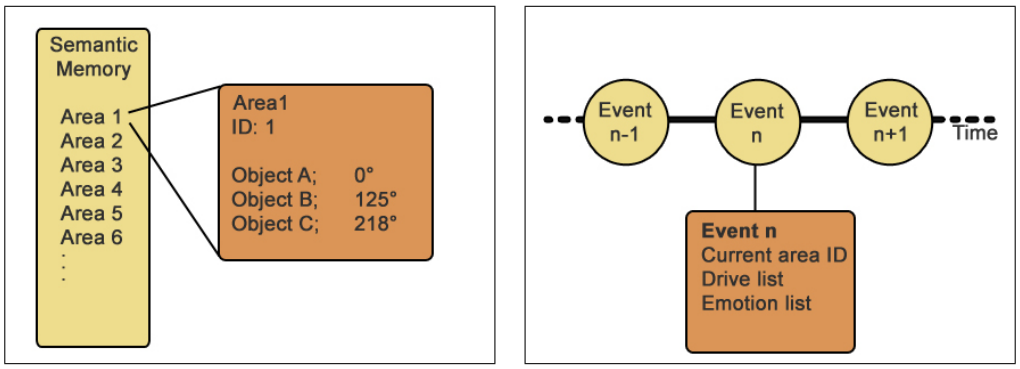


(a) (b)

Figure 3.1: (a) Example perception with 3 landmarks (A, B, C); (b) Example perception after normalization

The downside of this approach of course is that the main object has to be clearly identified at later encounters which makes a proper selection necessary. As main object, the one that is most likely to be solid is to be used to make sure it does not move or vanish at any time. That means a requirement to the object recognition that has to use properties like size, mass, movability, etc., to define the degree of stationarity.
In this work the objects' distance information is not used for area definition, but it would be a possible extension for future work to get more precise position information (see Section 6.2).
Based on the normalized list of perceived landmark objects, the area is defined and, as the example assumes that it is the first encounter with this area, saved within the semantic memory module.
Figure 3.2(a) shows the encoded area from the example location in Figure 3.1(a). The area entry here contains the normalized informations about all the visible landmarks as well as the

assigned ID number of the area. Figure 3.2(b) shows that only this number is saved together with drives and emotions within an event in the episodic memory.



(a) (b)

Figure 3.2: (a) information saved within semantic memory; (b) information saved within event of episodic memory

Together with the normalized object list, the up-to-dateness of the area is saved which is a value with the range 0 . . . 100. This value represents the likeliness that a previously encountered area is still there and is the first step towards the adaption to environmental change. A value of 100 means that the area most likely still exists and vice versa. It can also be used in a similar way as forgetting and influences for example the planing of a path. The value is altered due to 4 different causes. The first is a reduction of the value over time, representing the process of forgetting, or loosing access (see Section 2.4). The second one is due to a hit or miss from the assumption system. The system (see Section 3.1.3) is triggered every time an area border is crossed and gives an assumption about where the current transition will end. This guess is based on informations about previous transitions. An assumption is said to have missed when an area transition that was predicted by the assumption system to end in a specific area X in fact ended in a different one. In that case it has to be assumed that there occurred a change within area X and that it is not up-to-date any more. This massively lowers the up-to-dateness level. Area evaluations due to assumptions are explained more detailed later in Section 3.1.3. The third case is when the memory system is not absolutely sure to know a location. Such a case is explained more detailed in Section 3.1.2. Only the fourth cause raises the up-to-dateness value. It is done when the agent re-experiences a previously known area when reentering it. By seeing the area again, its memory about it is refreshed.

Additionally to the semantic memory a reference of the current area is saved within the stream of events in the episodic memory module (see Figure 3.2(b)). This way every generated event holds the current area. By doing so not only a memory of movements is available, but also a connection between area and the emotions and drives that might emerge from residing in the area. The drive and emotion data will become important when remembering areas and when planing paths.

The encoding consists of area and event encoding, but the two are separated in terms of triggering. In contrast to the event encoding, the area is to be encoded constantly. This means after every performed move or on a periodic time base. This is done to make sure to always have correct knowledge about the current area. As an already known area is not added again to the area list residing within the semantic memory, this method does not affect the memory usage. The event encoding on the other side has several triggering conditions. Besides the one described within [DGLV08] which also contain triggering on strong emotion change or drive change, an

additional triggering has to be added that reacts on area transitions. Thus a new event is also being encoded when the agent is crossing an area border. This forces the question about how a border crossing is detected. The answer is given by the retrieval unit that is described in Section 3.1.2 who's task is to search for known areas that have been encountered previously. A transition is recognized in case 2 consecutive retrieval processes have different results.
Depending on the amount of different areas, the demand for memory can vary. The semantic memory reaches a certain size as soon as all possible areas are discovered and only extends this size due to environmental change. The episodic memory on the other side, can grow constantly. When just focusing on the event encoding due to area transitions, the growth rate is depending on the movement activity as well as on the amount of areas. Therefore the tolerances have to be adjusted in a way that allows a localization with the required precision but also avoids inefficient memory usage.

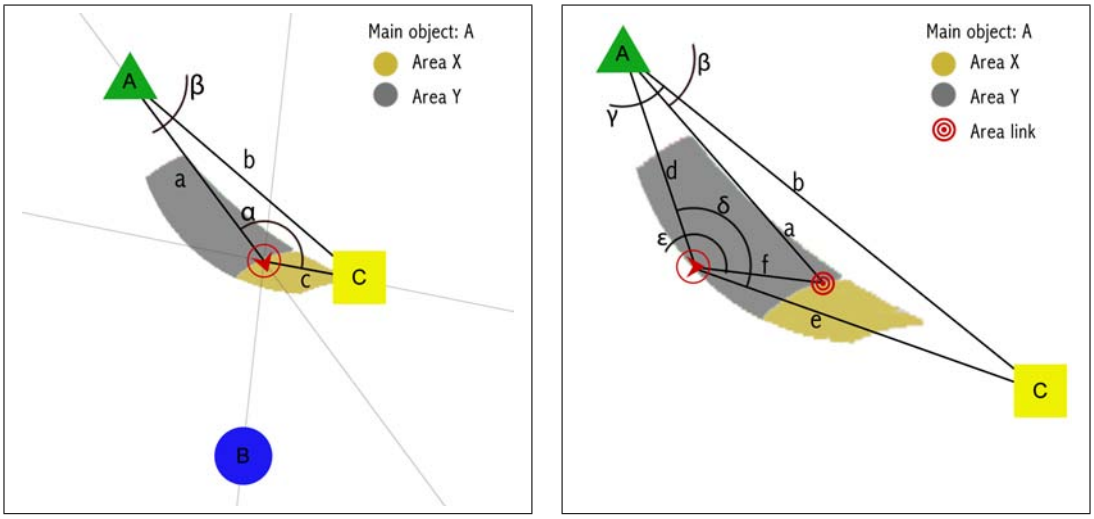
The quality of the segmentation can be measured by the way locations that form an area are scattered. In an optimal segmentation the locations would form a single unit. This of course strongly depends on the amount and quality of different landmarks. In the worst case several positions on the map that have no connection to each other belong to the same area because the environment just looks similar. Such a situation might happen when very few landmarks are used that also look alike. In that case errors can occur within the retrieval unit as well as in the path planing unit. The reason is, that it cannot be determined which of the scattered areas with the same ID is the current one. On the other hand if the object recognition becomes very detailed and thus the amount of landmarks becomes very high, an additional factor could be added. This value represents the importance of an object for the area definition. An exaggerated example would be if blades of grass in a garden are recognized as landmarks, their importance would be rated very low because they exist in a large amount of areas.

Area Link Coding

The link encoding has the task of recording the performed movement every time the agent crosses an area border. This is necessary for gaining the knowledge about how the areas are connected to each other and thus how to get from one area to a neighboring one. In combination with the area definitions this makes a rough map like knowledge about the world available. Based on that knowledge the path planing as well as the assumption system can be realized. With the recording, the topological network concept is extended with a more detailed area connection information. An approach that follows the idea of Moar and Carleton [MC82] (see Section 2.2).

The recording itself is performed by a dead reckoning algorithm, that uses the information shown in Figure 3.3(a) for link encoding. The shown distances a and c as well as the angle α are provided by the vision system. B and the angle β have to be calculated as it is shown later. In contrast to common relative positioning systems like the ones explained in 2.1.2, this algorithm does not record constantly. It is triggered only when an area transition occurs. As mentioned before, a transition is recognized when 2 consecutive retrieval processes, performed by the retrieval unit (see Section 3.1.2), have different results. If this is the case the link encoding uses the position information from two of the encountered landmarks for encoding. It is sufficient in this case to use only two landmarks as the third information source that is necessary for a precise calculation is the order of the landmarks.
Based on the provided data, which is the distance to the two landmarks as well as their angular relation (see Figure 3.3(a)), the necessary information to recover the linkage position can be calculated. The fact that at least two landmarks are necessary to define a link means additional

demand to the quality and the amount of the landmarks. But it also is consistent with human orientation system. [FAW07] This shall be show with the following mind experiment. A person is standing in a room in which the walls are to far away to see. The only thing in sight is a circular platform looking the same from every direction. This would be the agents view with limited range of sight and using a low level vision system when it only sees a single landmark. Based on that view it is not possible to tell in with direction a certain destination would be. Two landmarks would allow to do so and the precision rises with the amount of landmarks as was shown by Fitting et al. .



(a) (b)

Figure 3.3: (a) Information used for link encoding; (b) Information necessary when reusing link

The calculation of the link is a mixture of triangulation and trilateration described in Section 2.1.4. It starts with the angular and distance data a, c and α as soon as the agent crosses the border while traveling from area Y to area X. Using these values and the law of cosine, b and afterwards β can be calculated using equation 3.1 and 3.2.

$$b^2 = a^2 + c^2 - 2ac\cos(\alpha) \tag{3.1}$$

$$\beta = \cos^{-1}\left(\frac{a^2 + b^2 - c^2}{2ab}\right) \tag{3.2}$$

The representation of these distance and angular values can be seen from Figure 3.3(a). To find this position again and to pass it correctly in the future the values a, b and β as well as the current heading direction relative to the main object have to be saved. This is done within the event that is being recorded by the episodic memory unit. By doing so, the agents whole journey is being recorded and gives the ability to redo previous transitions. An example for such a linkage can be seen in Figure 3.4. It shows how the sequential events within the stream are linked with movement data.

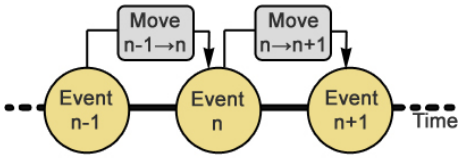


Figure 3.4: Area linking example

This calculation is performed twice per transition. Once using the vision data that was recorded before crossing the area border and once using the one after. The first one is important when trying to pass the link in the same direction again. The later one is used to make the link also passable to the other direction. The reason for this is that it might be the case that in the goal area there are different landmarks to identify the link position. It has to be stated here that it is assumed, that an area link can be used in both directions. Situations in which a door can only be passed in one direction are not considered and are topic for future work mentioned in the Section 6.

Figure 3.3(b) shows a typical situation in which the encoded link is used to re-find a path from one area to an other. The agent is currently residing in area X and intends to travel to area Y. As such a transition was made before (see Figure 3.3(a)) the necessary information to do so is already saved within the episodic memory and can be reused. Therefore the saved transition information a, b and β and the information d, e, δ from the current vision data is used. The meaning of the variables is shown in Figures 3.3(a) and 3.3(b). Based on these values, the necessary heading direction relative to the main object ϵ can be calculated as shown in the following. Additionally it would be possible to calculate the distance to the link position which will become important for the assumption system shown in Section 3.1.3

The calculation process is done as follows

$$\gamma = \cos^{-1} \frac{d^2 + b^2 - e^2}{2db} \tag{3.3}$$

$$\lambda = \beta + \gamma \tag{3.4}$$

$$f^2 = d^2 + a^2 - 2da \cos \lambda \tag{3.5}$$

$$\epsilon = \cos^{-1} \frac{f^2 + d^2 - a^2}{2fd} = \cos^{-1} \frac{b - a \cos \lambda}{2f} \tag{3.6}$$

This encoding has two important requirements that might affect its performance.

- The same 2 landmarks that were used for the encoding have to be recognized again when reentering the area and trying to calculate the way. This might not be the case if for any reason the retrieval tolerance is set very low (see Section 3.1.2)
- The second problem is related to the exactness of the object distance and angle information. Of course the more precise the measurement the better the calculation outcome. The worst case scenario in this case would be a wrong direction angle that leads to an undesired area and the link quality is lowered. In this situation it is not clear if the error occurred due to change of the environment or because of false calculation

As the first implementation of the system is mainly tested within a virtual simulation (see Section 2.5.3), the vision information is very precise and should cause only little error. The reaction to real world sensor error is tested within the e-Puck implementation described in Section 4.5.

3.1.2 Retrieval

The retrieval system performs the vital task of remembering an area and gives an answer to the main question of a self localization system, "Where am I right now?". This step has to be done every time the agent performs a movement in order to keep the knowledge about the current location up to date.
It checks whether an equivalent place has already been encountered at an earlier time or saves it,

if it is an unknown and thus new area. To do so, it utilizes the semantic as well as the episodic memory system as knowledge base. The retrieval process itself consists of two parts. For one the search for similar known areas. This is a first filtering of the entries within the semantic memory based on environmental data only. In a second step, in case the first filtering instance has several results, an additional evaluation is performed that ranks these results to find the one that is most likely correct. This second filtering additionally includes emotional and drive data for comparison. The triggering of the process is done every time a step was made and an area is encoded as described in Section 3.1.1. That currently encoded area is handed to the retrieval unit which decides if there already is an existing equal or if it has to be saved within the semantic memory. One way or the other it returns an area ID code which represents the actual answer: The current location is within area... .

Area Search

The area search is the first step in remembering. It performs a first raw comparison of the entries within the semantic memory system and the current area. In the following, the newly encoded area is named "current area" and the previously recorded entry within the semantic memory that is being compared with is named "memory area".
The task of the area search is to filter the memory areas by comparing only the environmental information about encountered landmarks. That means it is only important if a memory area looks the same, not if it is the same feeling to be there, according to the emotions and drives. This filtering can be interpreted as an abstract version of finding the location of the local map within the world map in map based localization (see Section 2.1.1). The world map is saved within the area memory and the current visible landmarks represent the local world. To raise the performance of this process it is done in 2 steps.
First, only the entity of the objects itself is compared. Therefore it is examined whether the memory area consists of the same objects as the current area regardless of their position. It checks if the agent is able to see the same objects as in the area it remembers. To qualify for the second step a memory area has to have at least a defined amount of similar objects as the current area. This *similar-object-tolerance* is set as a percentage value. A similar-object-tolerance value of 20 % for example means that more than 80 % of the objects within the memory area have to be the same as in the current area. 0% tolerance would mean that the memory area and the current area have to contain exactly the same objects. The objects similarity is calculated by using the defined equation 3.7.

$$\text{object similarity} = \frac{\text{amount of similar objects}}{max(\text{objects in memory area, objects in current area})} \tag{3.7}$$

With this raw filtering step the workload for the second more complex step is reduced. Thus the overall processing time can be lowered.

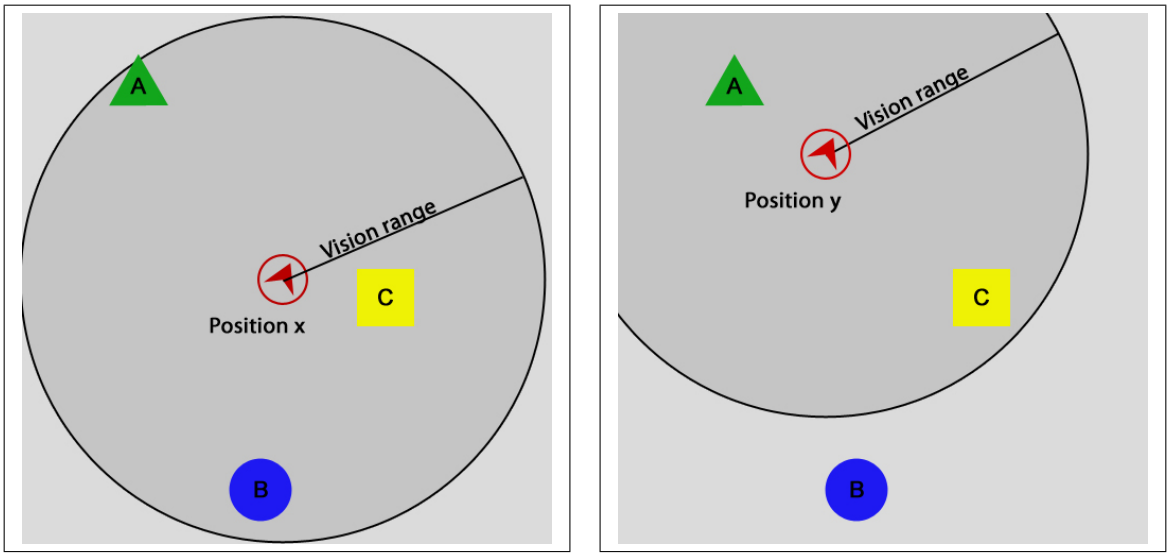
The second iteration compares the angular position of the objects. To do so, the reference angle has to be adjusted first, by not using the selected main object but the same as is used in the memory area. Therefore the current area is normalized again with reference to the element that is the main object within the memory area. With the same reference object the angles of the other landmarks become comparable. Now every similar object except the main object is checked if it lies within the same discrete angular segment. This *angular-tolerance* can also be set manually and is the second possibility for system adjustment. The third one is the *similar-positioned-object-tolerance* that is also defined as a percentage value. For example, a similar-positioned-object-tolerance value of 10 % means that more than 90% of the similar objects have to be within the angular-tolerance. The object position similarity value calculated

according to equation 3.8 is compared with the similar-positioned-object-tolerance and used for the final filtering.

$$\text{object position similarity} = \frac{\text{amount of similar objects in similar position}}{\text{amount of similar objects}} \tag{3.8}$$

In one sentence, the potential areas resulting from the area search are the ones, whose amount of non similar objects does not exceed the similar-object-tolerance and who have not more similar objects out of the angular-tolerance than defined in the similar-positioned-object-tolerance.

In the end most of the time there are several possible potential areas whose values are within the tolerated range. In such a case the similarity values are combined and used for ranking the areas to find the most suitable one. Therefore the product of the object similarity and the object position similarity is used. This sorted list is then kept for further processing, but the first entry within the list is expected to be the current area. The important thing to point out here, is that in case of an environmental change the combined similarity value of the pre change current area, the area the location belonged to before the change, is not 100% any more. It means the best result of the sorted potential area list does not fit perfectly. It is assumed that the environment has changed and that this best mach is the pre change current area. In such a case where there is no perfect match a new area is recorded to adapt the change. The up-to-dateness of the pre change current area is then lowered. Depending on the defined tolerances the pre change current area still has a tolerated combined similarity. Thus, it is still within the returned list of potential areas and is so still available. Thus the current location is in area 3 but it also looks a little as if it could be in 1 or 2.



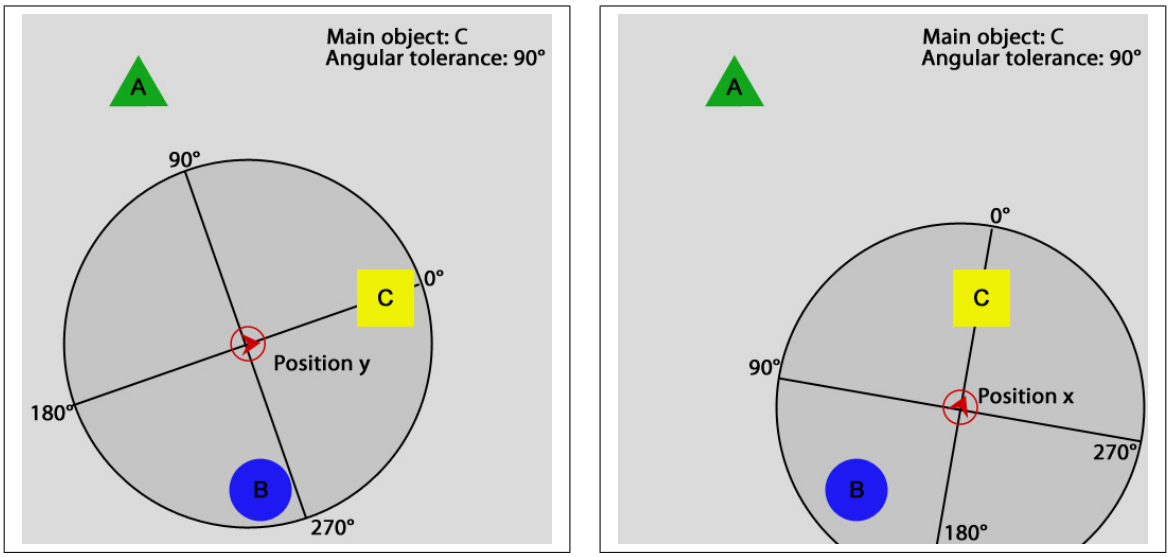
(a) (b)

Figure 3.5: Example for usage of Similar-Object-Tolerance

Figures 3.5 and 3.6 show usage examples of the 3 tolerance values. In Figure 3.5(a) a new area that contains the three shown objects A, B and C is being saved at position x. A retrieval process is started when the agent is situated at position y (see Figure 3.5(b)). In this case the object B is not in its vision area. Depending on the settings of the similar-object-tolerance, the area recorded in Figure 3.5(a) passes the first selection stage or not. As only 2 objects are the same here the object similarity value is $66.\dot{6}\%$. This value is calculated by using equation 3.7. Therefore, it would require a similar-object-tolerance of at least $33.\dot{3}\%$ to pass. If however the

similar-object-tolerance is lower, the memory area is not used for further comparison. Otherwise if the value is greater or equal to the $33.\dot{3}\%$ the area is added to the potential area list and used for object position comparison.

In Figure 3.6 an example for the usage of the angular-tolerance as well as the similar-positioned-object-tolerance is shown. Again a new area is being encoded at position x. When the agent is located at position y at a later time a retrieval process is started. In this example the area recorded at position x passes the first filtering step as it holds exactly the same objects as the current area at position y. Now the angular values become important. In a first step, the main object of the current area is adjusted to be the same as the one of the memory area, which is object C in the example. As it always resides at angle 0°, this main object is not used for further angular comparison. For object B, the position in the memory area is within the second discrete angular section which is, due to the angular-tolerance of 90°, between 90° and 180°. In the current area object B however is situated within the third discrete angular segment from 180° to 270°. Depending on the settings of the similar-positioned-object-tolerance the memory area stays in the potential area list or is being deleted. As in this extreme example the similar-positioned-object-value is 0, the similar-positioned-object-tolerance would have to be 100% to keep this memory area in the list.



(a) (b)

Figure 3.6: Example for usage of similar-positioned-object-tolerance and angular-tolerance

Besides the quality of the landmarks and their positioning, these three tolerance values have a strong influence on the quality of the world segmentation and thus on the performance of self localization. For example a large angular-tolerance of 90° would result in much less areas and thus in a much lower precision than a 15° tolerance. A very low value on the other hand, like 1° would produce an inefficient high amount of very small areas. So, depending on the application and on the environment these values have to be adjusted as needed.

Besides the assumption system which is explained in Section 3.1.3, the shown tolerances play an important part in reacting to environmental change. If the similar-object-tolerance for instance is set high enough, a suddenly missing object would be tolerated. As explained before, besides the newly encoded area for the changed environment, the old area that was correct before the change is also saved within the potential areas list. With this still existing knowledge

about old, now changed, areas, the old transitions that connect these areas can still be used for a better path planning. So the environmental change has less effects on the system. Of course these tolerance values would also have impact on the overall system performance in case there are in general only few landmarks to work with. On the other hand if enough landmarks are provided, a larger tolerance would only cause minor performance reduction and would help with the environmental change. This is a trade of that has to be decided by the user.

Area Evaluation

In case the area search has more than a single result, an evaluation of the potential areas has to be done. This situation might occur when to many similar Landmarks and thus some similar looking areas exist within the world map. Therefore the variable set for the filtering has to be extended. It is done by not only using external input, like the environmental data from the vision system, but also internal data from the agents body itself. For this purpose the information from the episodic memory is used. It contains emotions or drives that might have been experienced in a particular area and actions that have been performed here. So the area is not only remembered because it looks the same but also because it feels the same to be there. An area for example might cause some kind of fear. If the fear is experienced later again it might be within the same area as before. Solms ([ST02], p. 106) calls it the sixth sense. "... if you were deprived of all sensory images (drawn from present and past perception), you would still be conscious. You would still be aware of your inner state ...". The agent is not deprived of its external inputs but it cannot solely rely on them so it has to use its sixth sense, the inner state to get a final result.

To check the similarity, the emotion and drive entries simply are being compared to each other. Therefore for every potential area the episodic memory is searched for events that were recorded within this area. For each positive result the event entries are compared and the deviation from the potential area is added up.
The process is related to the fact that certain places can trigger certain feelings or require special actions. An example is a place where someone was attacked and that will later be associated with fear or anger. It is the same with a place where someone had to accomplish a very unusual task.

- **Emotions:** As an emotion is simply an expression of the inner state, there are various ones (Solms [ST02], p. 105). The basic emotions, fear, anger and lust, are also called evolutionary-motions as they seem to have grown hardwired in our mind and thus seem to be similar for every human (Solms [ST02], p. 112). Within the ARS implementation they are measured as values 0...1, so in this system the deviation can be calculated very easily.
- **Drives:** Are the physical reaction of emotions and are also limited to hunger, seek and play. They are measured in a similar way as emotions and thus the deviation can also be calculated as a simple value
- **Actions:** The ability for different actions is very limited within the simulated agent. The list of currently implemented actions within the ARS simulation contains move, eat, flee, swim, dance and so on. As for comparison they can only be the same as the current one or not, so the compare result can be 0, not the same or 1, the same.

The emotions, drives and actions of every episodic memory entry that occurred in the same area as the potential current one is compared with the current emotions, drives and actions. For

calculation equation 3.9 is being used. The lowest of all these values is saved and used for the final ranking of the potential areas.

$$\begin{aligned} \Delta\text{inner state} &= \Delta E_{fear} + \Delta E_{anger} + \Delta E_{lust} + \Delta D_{hunger} + \\ &\quad \Delta D_{seek} + \Delta D_{play} + \Delta A_{move} + \ldots + \Delta A_{dance} \end{aligned} \tag{3.9}$$

When the inner state deviation for every potential area has been evaluated, the one with the lowest deviation value is said to be the correct current area. Different to the inner state consideration within the path planning system shown in Section 3.2.2, the age of the memories is not being considered here.

3.1.3 Assumption System

The main goal of the assumption system is to deal with dynamic environmental change. It performs a "guess" on where the agent is heading to when crossing an area border. This way it can be checked whether the agents knowledge of the world is still up to date or if it has to be altered. At every transition an assumption process is triggered and the expected area the transition will result in, is calculated. Based on the prediction, change within the environment can be noticed and appropriate actions are being triggered. In Figure 3.7 for example, when crossing the area-border of area "A" the assumption system returns that this particular movement will result in residing in area "B". As the result is correct the knowledge is assumed to be up to date and no further action is required. How to calculate this assumption result and which measures are being taken in case of a false result is shown in the following.

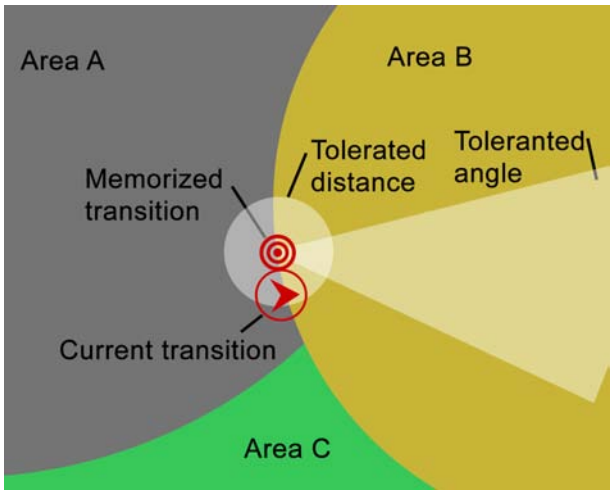


Figure 3.7: Example for heading and distance tolerance in the assumption system

Assumption Calculation

The assumption calculation is done by utilizing information about all the previous transitions that have been saved within the events of the episodic memory. This memory holds the knowledge about the agent's whole journey. It therefore also has the information if a similar transition was performed before and where it ended. Based on this transition list a two step filtering process is performed to find the most similar one.
First of all only those connections that started in, or are leading to the area in which the current transition started or ended in, are being selected and used further on in the next step. This is simply done by comparing the start and goal area ID within the saved transitions with the area

ID the agent was located before crossing the area border. By using transitions from both areas, also both, starting and goal area, can be tested in one step.
In the second filtering the memorized transitions are compared to the current one in terms of similar location and similar movement direction. Both values have to be within a certain tolerance to qualify for being the correct compare match. An example for such a comparison situation is shown in Figure 3.7. It shows the range of the tolerated distance from the memorized transition and the tolerated angular deviation from the transition angle at recording time.

The position similarity comparison is done using the same system of equations as shown in the section "Link Encoding" (see Section 3.1.1 and equations 3.10 to 3.14). The calculation is based on the vision data from 2 of the landmarks within the current area, as well as the data saved within the memorized transition. Using this information the position, of the memorized transition relative to the current position can be calculated. Here the distance f from the current position to the position, in which the memorized transition occurred, is the important value to calculate. The closer the memorized link is to the current one, the more likely it is to be the same transition and also to end in the same area. The size of this tolerated area is to be set by the user according to environmental properties. If a very large set of landmarks is available to work with, the areas are rather small and thus a smaller tolerance should be set. For environments with very rare landmarks it is exactly the other way around.

$$a, b, \beta \quad \ldots \quad \text{data from memorized Transition} \tag{3.10}$$

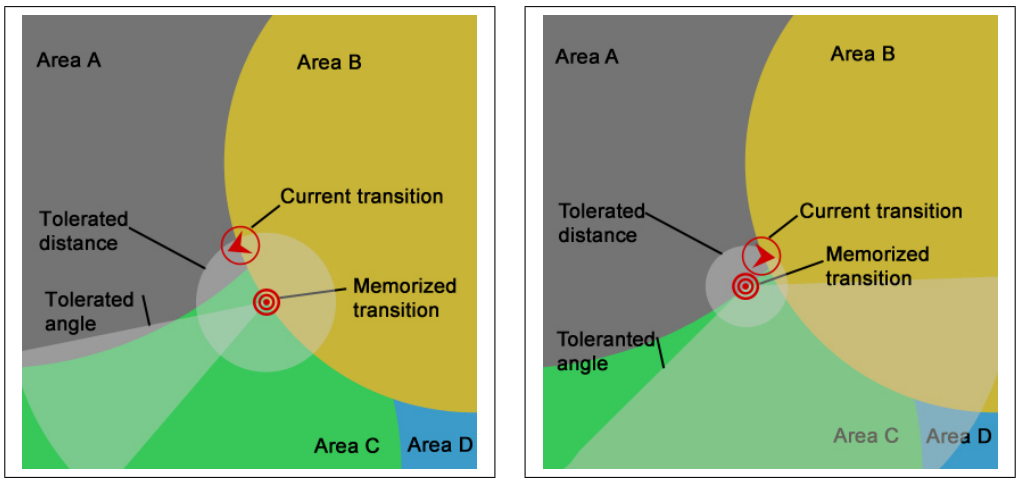
$$d, e \quad \ldots \quad \text{Current distance to main and secondary object} \tag{3.11}$$

$$\gamma = \cos^{-1} \frac{d^2 + b^2 - c^2}{2db} \tag{3.12}$$

$$\lambda = \beta + \gamma \tag{3.13}$$

$$f^2 = d^2 + a^2 - 2da \cos \lambda \tag{3.14}$$

The heading direction difference is calculated by using the current relative angle between walking direction and the main object of the area. This moving direction value is also saved within the memorized transitions at recording time and is used for comparison here. Additionally to the position similarity, the heading direction when passing the border has to be within a certain range to qualify for a comparison match. The tolerance is the second criteria for



(a) (b)

Figure 3.8: Error situations at 3 area crossing point

transition similarity that has to be adjusted. It can have very strong influence on the result as shown by the possible errors in Figure 3.8. Errors can mainly occur near locations where more than 2 areas are crossing. Here the error rate can be very high if the distance tolerance has a large value, like in Figure 3.8(a), or the angular tolerance has been chosen very big (see Figure 3.8(b)). In both situations the assumption system can predict a false transition outcome. As the false result is also interpreted as an assumption miss the same actions as in the case of an environmental change are performed. This case would be an error of the system. Those two tolerances form the second step of the raw selection from the memorized transitions. But after a long journey, meaning a long time of recording, the agent might have acquired a very long list of transitions within its memory. This can cause the filtering to have more than one result. In the case these results all lead the same assumption, the process ends here. If otherwise the several resulting transitions lead to different areas, a post-selection has to be performed, which is done by adding more criteria to compare with.

- Compare the distance data more thoroughly. The closer, the more likely it is to be the correct transition
- Compare the crossing angle more thoroughly. The close to the original crossing angle the better
- Check the up-to-dateness of the resulting area. The one that leads to an area with higher up-to-dateness is more likely to be the right transition

Using these additional variables, a new rating-distance is calculated based on which the results are compared again. The transition with the best rating-distance is believed to be the correct one.

$$distance_{rating} = \text{distance} \times \left(\frac{\text{heading difference}}{\text{tolerated heading difference}} + \left(1 - \frac{\text{up-to-dateness}}{100}\right) \right) \quad (3.15)$$

The quality of this whole assumption calculation strongly depends on the quality of the distance and angular information that is provided by the sensors. In the case of the Bubble World simulation, the data is very accurate so the error rate is rather small. But if implemented in a real world robot system the uncertainty of the provided data is added to the defined distance and heading direction tolerances and may cause false predictions.

Memory Alteration Based on Assumption

As mentioned before the assumption system is meant to recognize and to deal with environmental change. Therefore the assumption result is used to trigger the modification of the believes about the environment.
The recognition of a change within the environment is realized by comparing the assumption with the real transition outcome. If the assumption system predicts a different area than the transition really results in, the area memory is being altered. This is done by lowering the up-to-dateness level of the predicted area, as it is very likely that there has been a change in that particular region. The change could be a new, a missing, or a moved landmark. For example a tree was cut down in the meantime and is now not available as landmark any more. Depending on the tolerance settings for the area retrieval (see Section 3.1.2), minor change of this kind might be even tolerated by the area recognition system. The lowering of the areas' up-to-dateness value can be interpreted as a loss of trust in the area memory. On the other hand if the results fit, the up-to-dateness level, and also the trust, is raised.

An other system property defined here is, that regaining trust in an area takes more time than loosing it. This simply means that the up-to-dateness level raises slower than it is lowered. For example an up-to-dateness value that has a range from 1 (max) to 0 is being multiplied by factor 1,5 to raise the level but by the factor 0,5 to lower the level. Regaining maximum thrust (level 1) again after one lowering process due to an assumption miss, takes 2 correct transitions to this area (see Figure 3.9). In step 1 an assertion miss caused a lowering by the factor 0,5 and thus a very strong loss in trust in the area memory. In the following 2 steps the agent twice reenters the area it thought had changed and a raising process is performed until up-to-dateness is at level 1 again.

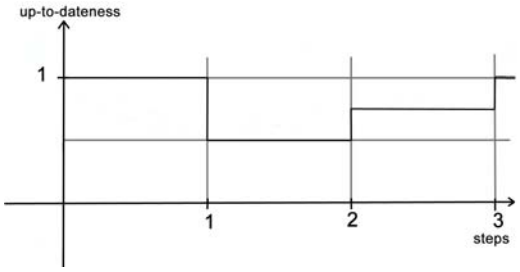


Figure 3.9: Example of up-to-dateness modification

So the whole information about the environmental change is saved within the up-to-dateness values of the areas. This information is mainly used by the orientation part of this system, which is explained in the following Section 3.2. It has to make sure that the proposed path is also a possible path. So it must only contain areas that are likely to still exist. Otherwise the path might lead to a dead end.

3.2 Orientation

Next to the positioning system the orientation is the second major part of this model. Its main purpose is to suggest paths to travel from the current area to an other desired area, if necessary. With the information about which objects can be found in certain areas, this part helps to arrive at these objects in the fastest known way. The model contains also reactions to select these desired goals unconsciously caused by certain emotions or drives.
The orientation system is actually the part that provides the output information which is handed to the decision unit. It is done with a path planning algorithm (see Section 2.3) that selects a path depending on the length, the actuality and the emotional stress it might cause. The user, in this case the ARS decision unit, receives only suggested information (see Section 3.2.3) about which direction to go to arrive at desired goal. The positioning system on the other hand mainly monitors, learns movements and creates an abstract view of the environment. So its purpose is to collect information about the world that can be utilized by the orientation system to calculate a path as a valuable information. This proposed path is not a mandatory action the agent has to take. It is a kind of additional information that can be used by a decision unit or not.
Alike the natural human orientation the path description is not done by giving instructions like "walk 100m in North direction and then 20m in West direction...". It uses a more abstract definition that does not require tools like a compass to find north or west. Rather it can be verbalized as "turn right at the church, then turn left at the park...". The orientation is only done by using the known landmarks and areas. The system has no knowledge about the exact geographic location of the areas in terms of coordinates. Therefore the system is not able to tell in which direction the desired target location is and does not calculate an unknown direct route. It rather searches for areas that connect the current area with the desired one. The agent can

then be navigated from one area to the next until the goal area is reached.
Figure 3.10 shows an example for the path planning. The colored sections are areas the agent has encountered before while the white section is unknown. Every color represents a different area. Landmarks that define these areas are displayed as green marks. The small Cs represent the links that where used during the travel and are saved within the episodic memory. In this example the agent intends to travel from its current position to the marked target area. The path planning system now calculates a path that uses the known connections between the areas to get to the goal. This example path is highlighted in red.

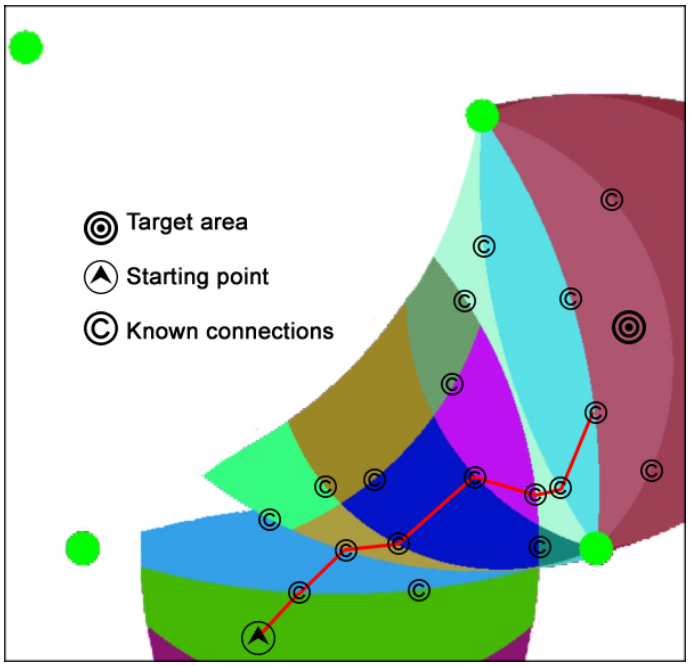


Figure 3.10: Example for a planed path

For being able to perform this kind of path planing, the data from the episodic and semantic memory, which was gathered by the positioning system, is used. By collecting informations about areas and transitions a kind of network map is being created in which the nodes represent areas and their connections are the transitions between them. This does not contain very detailed information about the world as a metric map would, but it gives a good overview. The rough information is enough to plan a path from one area to an other that is just using the learned transitions saved within the episodic memory. There are two questions that have to be answered by the orientation system in order give useful information.

- When should an orientation process be started? Meaning, when is a path calculation necessary.
- What is the fastest and safest possible way to reach the goal?

To answer both questions the orientation system holds two parts. The "Orientation Triggering" which gives an answer to the first problem, and "Path Planning" which deals with the later one. Both are explained more briefly in the following sections.

3.2.1 Orientation Triggering

Orientation triggering is all about when to perform a path calculation and to which goal. As orientation gives answer on how to get to a place, it is needed nearly every time an agent intends

to move. There are of course occasions when an orientation is not needed and also cannot be performed, because the goal area is unknown. This means when the agent does not know where it wants to go to, no path can be planed. An example for such a situation would be random exploration. Wandering around without any defined goal but with the intention to seek new areas in a non systematic way. An other one would be promenading.
Despite these situations there exist 2 ways to trigger an orientation process

spontaneous orientation triggering: Orientation is started due to a very strong drive or emotion. In this case the orientation system searches for a path to a place that fulfills the drive in order to balance the emotion.

conscious orientation triggering: Orientation is started because the agent consciously wants to go to a certain place. For example to meet an other agent or to return home. So in this implementation the conscious triggering can be performed by the ARS decision unit (see Section 2.5) by performing a kind of path planning request.

In conscious triggering the system already holds the desired goal area, because it was provided with the path planing request. So it is able to start the planning process right away. In spontaneous triggering on the other side the goal area is unknown at first. This process of making a decision where to go, is covered in more detailed in the following.

The spontaneous orientation triggering is consistent with phrenological concept of an automatic module mentioned in Section 1.3. The system is performing its task as soon as it receives input. The input can come from the emotion and drive system and cannot be controlled by the decision unit. A very natural human reaction for example is to seek for food whenever one gets very hungry. This can be put on a level with having a very strong drive for hunger and thus a strong desire for food or energy. The unconscious reaction of the orientation system is to plan the path to the nearest area that contains a food or an energy source. An example would be the kitchen or a shop near by. The important point is, that the drive can be fulfilled at this place. The direct opposite is with an emotion like fear. A place that causes a great fear also causes an unconscious path planning to a place that is known not to cause that fear and thus believed to be save. Again, the agents decision unit can decide whether to use the proposed path to the kitchen or to the save place, or to stick with the hunger and to deal with the fear. Either way the orientation system did its work which is choosing a goal area, planning a path and handing it over to the decision system. The necessary information for finding such a place is available within the episodic memory. As it holds data about the emotions and drives, it can easily be searched for a place where the currently alarming drive or emotion is satisfied. As this model is to be implemented within the ARS project the used inputs are the defined emotions and drives that can be seen in Table 3.2 ([DFZB09], p. 218).

$drives$	$hunger, seek, play$
$emotions$	$fear, anger, lust$

Table 3.2: List of emotions and drives implemented within the ARS project

If one of these elements reaches a certain threshold all the entries within the episodic memory are searched for places that have a very low value for this particular variable. This area is believed to contain something to neutralize the strong emotion or drive. For example, if the drive play exceeds the defined threshold, the episodic memory is searched for the entry that contains the lowest value for play. As every entry is saved together with the area it was recorded in, the desired goal area is found.

For special drives and emotions like hunger, an alternative way of finding a satisfying area is suggested. Since all the encountered non landmark objects are also saved within the area description, this information can be used to find an area that contains a food source. Such an approach does not require the agent to have eaten in this area before in order to experience and record an event with a low value for hunger. But on the other hand it makes pre defined definitions necessary, which objects satisfy which drives or emotions. In case there is more than one area that would fulfill the need, paths for all of them are being planed. The final selection is then made by the path planning as it returns the path to the best goal.

3.2.2 Path Planning

The actual path planning is started as soon as the goal area is defined. Either by the decision system when performing conscious triggering, or by the orientation system itself during spontaneous triggering. With this information and with the transition data kept within the episodic memory, the planning process is ready to begin. Based on the agents' past journey, parts of this path are used to from a new one.

First of all the agents' journey is being extracted from the episodic memory data. Therefore every performed transition is taken and is collected within a separate list. Additionally to this extraction, emotional area costs are calculated. This means that strong negative emotions like fear that may have occurred within an area is added up to a costs-like value that is later used to choose the optimal path (see Section 2.3). The value represents the emotional stress the agent experiences when residing within the area. It is calculated by comparing the saved emotional value for fear, anger and lust with a defined threshold level as shown in the following Section 3.2.2. The extraction process covers both tasks. It scans every event in the memory for transition entries needed to form the network map, and for entries with strong negative emotions to bring the emotional costs up to date.

Emotional Costs

The emotional cost is a value saved together within the area definition and can be raised and lowered. The reason is that an area might have caused fear once but in a later stay within the place this emotion did not occur again. So for defining the emotional costs, the exceeding of a threshold and the time is relevant. Emotions that came up a long time ago are not as important as recent ones. Similar to regaining trust in an area mentioned in Section 3.1.3, loosing a fear takes longer than growing a new one. As the episodic memory system saves the data on a timely basis, the number of the memory entry is use as time factor. For raising and lowering the emotional costs the factors $\frac{1}{0,3}$ and $\frac{1}{1,3}$ are used as shown in equations 3.16 and 3.17. To make this more clear an example for an area named X is shown. In its journey the agent collected 170 entries in its episodic memory. Table 3.3 shows 4 of these entries. All of them occurred within area X and each of these 4 entries hold a value for the defined emotion fear. Let these 4 entries be the 50th , 102th, 141th and the 167th.

Within the simulation the variable fear can be a value between 0 and 1. This example shows that in entry 102 a strong fear occurred while residing in area X. The fear threshold (t_{fear}) for the emotional costs is set with 0,75. A value above the threshold is considered a very strong fear. Based on this information the emotional costs for area X can be calculated as follows

$$\text{if } fear < t_{fear} \quad \text{then} \quad f = \frac{1}{1,3} \tag{3.16}$$

$EntryNr$	Area of occurrence	$Emotion : fear$
50	x	$0,2$
102	x	$0,8$
141	x	$0,2$
167	x	$0,1$

Table 3.3: Example memory entries

$$\text{if } fear \geq t_{fear} \quad \text{then} \quad f = \frac{1}{0,3} \tag{3.17}$$

$$a_{em}(age_{memory}) = 1 - e^{\frac{-50}{age_{memory}}} \tag{3.18}$$

$$c_{em} = \prod f * a_{em}(age_{memory}) \tag{3.19}$$

$$\begin{aligned} c_{em,x} = {} & (\frac{1}{1,3} \times a_{em}(120)) \times (\frac{1}{0,3} \times a_{em}(68)) \times \\ & (\frac{1}{1,3} \times a_{em}(29)) \times (\frac{1}{1,3} \times a_{em}(3)) \end{aligned} \tag{3.20}$$

$$c_{em,x} = 0,22 \tag{3.21}$$

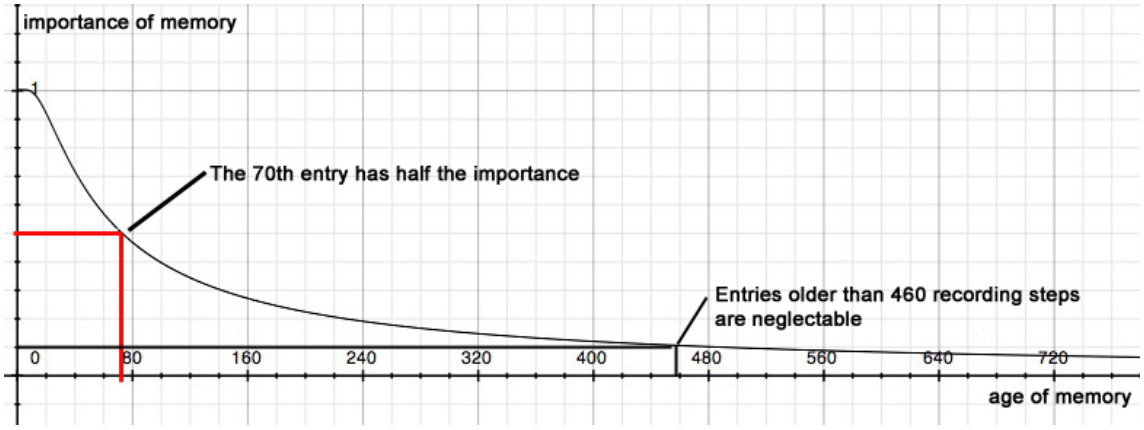


Figure 3.11: Example curve for age factor as defined in equation 3.18

Equation 3.18 calculates the memory weight as older memories are not as important as newer ones. Therefore with the age of the memory an age factor is determined. age_{memory} is calculated by subtracting the total amount of memories with the number of the used one. For the first memory which was the 50th within the memory list, the value would be $170 - 50 = 120$. So this memory was recorded 120 memories ago. As it is the oldest of the used ones it has the lowest importance. If there would have occurred a fear only in this first entry, it would have been nearly forgotten already. By adjusting the dividend in the power of e, the importance curve can be manipulated. As can be seen from Figure 3.11 , with the current setting of -50 the 75th entry is about half as relevant as the first one. For example setting it down to -20 would make only more recent memory entries important, and the 30th entry would already have half the relevance as the first one.
In equation 3.19 the emotional cost factor is calculated. It is the result of the product of all the factors f of each relevant entry multiplied by the age factor a_{em}. With experiencing only one moment of fear within this area that was a relative long time ago, the emotional costs are rather low. If however also the last 2 visits, in entries 141 and 167, within the area caused a fear than the value would be $4,1$. So a negative emotional association with a place fades with time and can be compensated by new positive experiences.

With this information the path planing algorithm is able to compute a much more qualitative route that causes much less emotional stress.

Path Calculation

In this part of the system the actual route is processed. Meaning the transitions from the list are being picked that have to be taken in order to reach a desired destination. The computation can be done using different existing algorithms. In this work, the realization with the Djikstra algorithm is shown, as well as a more basic trace-back realization similar to the implementation of Wang Ching Ho et al. [HDN03]. This trace-back algorithm is important because it is being used in the first implementation. To make later versions more effective and productive, the Djikstra calculation should be used. By doing so a larger variety as well as shorter paths can be calculated, as will be shown later on.

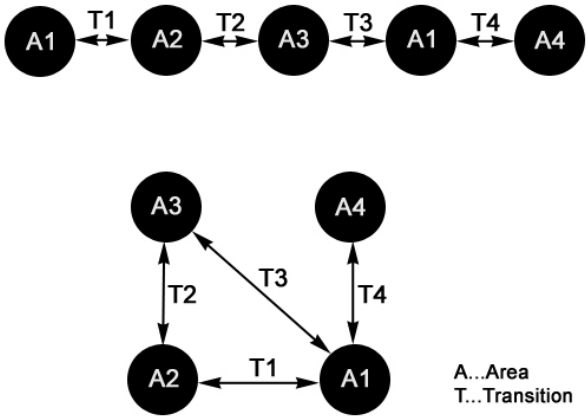


Figure 3.12: Network map from transition list

For the Calculation all the knowledge, that was gathered during traveling, is used. Now there are 3 different information data sets to base the path decision on.

Transition list: A list of places that are known to allow a movement from one area to an other. Based on this information the network map can be created.

Area up-to-dateness: The information about whether an area is likely to still exist or not. The up-to-dateness is used to alter path costs. As places that are uncertain to exist, are being avoided to make the path as save as possible.

Emotional costs: Information about how the agent feels about an area. Whether it fears this place or not. These values are also added to the path costs as the path that causes the least emotional stress is desired.

Figure 3.12 shows an example of how the transition list can be interpreted and used. Firstly as a sequence where the traveled path can be recapitulated. This representation will be used for the trace-back approach. A better alternative is to see it as a network map. The difference is that areas and transitions that have multiple occurrence in the sequential representation are now being combined, as can be seen with area 1.

Trace-back approach: This approach is based on redoing previous steps one by one. The whole journey is seen as a single long sequence. A cohering part can be taken and redone.

An example would be the travel part from area 2 to area 4 which would also contain crossing area 3 and 1 in between. The search algorithm for such an implementation is now explained.

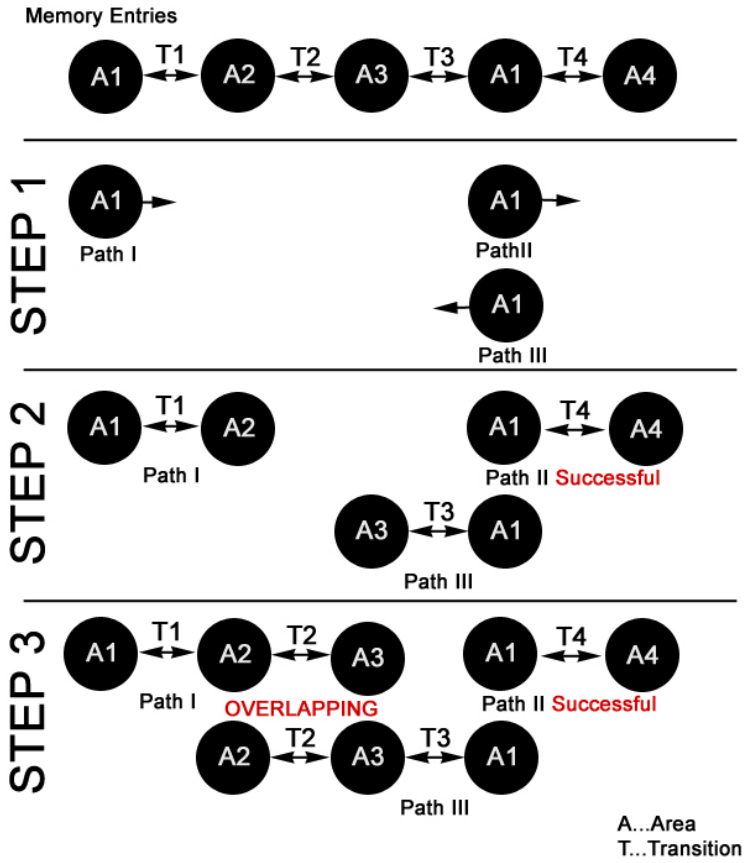


Figure 3.13: Example trace-back approach

The paths are separated into 3 lists, the ones that succeeded to reach the goal, the ones that did not succeed and the ones that are still in process. Additionally they are classified into two categories, a forward path and a backward path depending on the order the steps have to be redone to reach the goal. At first the transition list is searched for transitions that start or end within the area that the agent is currently residing in. For each one found, a new path is being created and added to the in-process list. It is also classified as forward path, if the transition started within the current area, and as backward path if it ended within the current area. An example would be an agent residing in area 1 having knowledge of a sequential transition list like in Figure 3.12 and trying to go to area 4. The process is shown in Figure 3.13. Here we would have a forward path I starting from the very first encountering of area 1 and containing transition 1. A forward path II starting in the second memory of area 1 and containing transition 4. Finally a backward path III also starting here and containing transition 3.
This set of paths are now being extended and divided into the ones that lead to the desired goal and the ones that do not. In one step, every path from the in-process list is extended with the next transition within the sequential representation. For backward paths the previous one is used for extension. If the newly added transition leads to the desired goal, the path is added to the succeeded list like path II in step two. There are two reasons for a path to be set as not succeeded. One would be crossing with an other path, like forward path I and backward path III would in step three. In that case both paths are moved to

the not-succeeded list as they cannot lead to the desired goal any more without containing an other, shorter path that has also been calculated. The other possibility would be a too low up-to-dateness level of an area the path is crossing. Or a too high emotional costs value. A path that is too uncertain to exist and one that causes to much emotional stress is not acceptable. The terms "too low up-to-dateness" and "too high emotional costs" mean that the values are below or exceed a defined threshold. In case none of these situations occur the new transition is added and the path stays within the processing list.

The filtering is done as soon as there are no more paths left in the processing list. Finally the paths that were found and kept within the succeeded list are ranked to find the best possible one. This ranking is done by calculating a combined path cost value from the path length, which is the amount of transitions that have to be taken to reach the goal, and the up-to-dateness level as well as the emotional costs of the crossed areas.

$$\text{path costs} = ac + \sum_{a=0}^{ac} \big((100 - \text{up-to-dateness}_a) + \text{emotional costs}_a\big) \tag{3.22}$$

$a \ldots$ area on path

$ac \ldots$ amount of areas on path

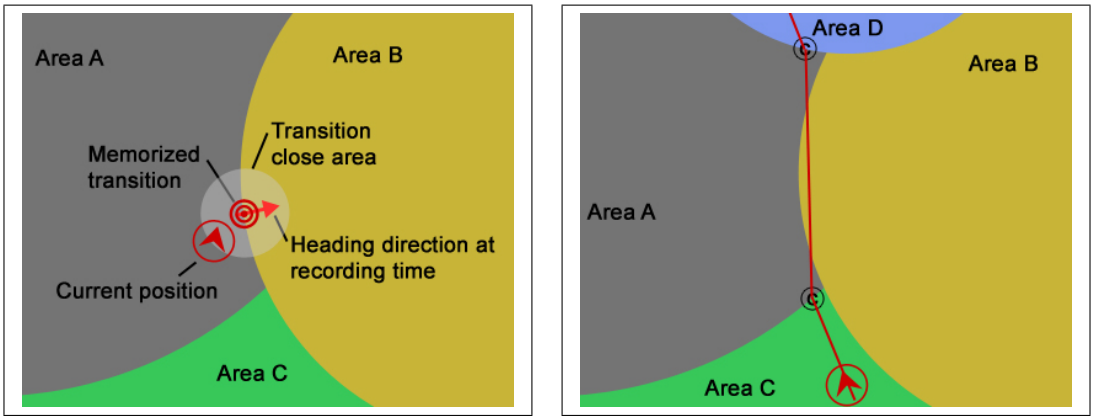
The path with the lowest path cost is considered to be the best one in terms of actuality and emotional stress, and is provided to the decision unit.

Djikstra approach: This approach splits up the described sequence into single transitions and can thus calculate more complex connections between areas. As the Djikstra algorithm has already been described in Section 2.3.1, it is done more roughly here. This time the areas are separated into different lists. The "open" list, the "queue" list and the "processed" list.

1. At first the current area is added to the processed list.
2. Areas from the open list are searched, for which a transition exists to one of the areas within the processed list. If the area up-to-dateness is below a, or the emotional costs are above a defined threshold it is removed from the list. Otherwise the area is added to the queue list. A new path is created leading to it by using the found transition. If there exist several similar transitions, same starting area and same ending area, only one path is created. From these paths the one that has the lowest costs according to equation 3.22 is selected.
3. If this best path leads to the desired goal, the search is finished, otherwise the area it leads to is added to the processed list and the process continues with step 2.

The result is the best path in terms of actuality and emotional stress. This approach requires more computation than the trace back, but it also allows more efficient paths.

Based on these two approaches, the example in Figure 3.12 shows very clear how different the efficiency can be when using a sequential interpretation with trace-back or a network map with Djikstra algorithm. In the sequence a path from area 2 to area 4 would take the transitions: T2, T3, T4. In the network map on the other hand the same path would only take T1 and T4.



(a) (b)

Figure 3.14: Path direction calculation additions to compensate imprecise transition data and intermediate border crossings

3.2.3 Path Following

After a path has been planed a method is provided to follow it to the goal. Since the path is a trip from transition to transition, the location of the transitions are defined as intermediate goals. Starting from the current position, the transition from the path that leaves this area in goal direction is used as next goal. To find that goal, the direction and distance to the transition is being calculated by using the equation system 3.3 in Section 3.1.1. By using these equations the agent can travel to the transition location. Problems might occur due to the resulting shapes of the areas with the used segmentation. As can be seen from Figure 3.14(b), when following a calculated path using the connections C-A A-D, for some reasons it might happen that other borders are crossed in between. But this does not mean that the path is not followed and must thus not be deleted. Therefore all intermediate border crossings cause no additional reaction until the agent has reached the *transition close area*. The close area, as can be seen in Figure 3.14(a), is a region with a distance to the transition smaller than a defined value. If however the transition has been reached and the following border crossing is not as planed, the path is being deleted since it is assumed that is not being followed and has become obsolete.
When arrived at the transition close area the system directs the agent according to the heading direction saved within the transition data (see Section 3.1.1). This means that the agent passes the border in the same direction, relative to the main object, as at the recording time. So it is ensured, that the crossing is correct and the agent arrives in the next area as planned in the path.
As mentioned before, the first area crossing that happens after the agent arrives in the close area of the transition is assumed to be the next within the planned path. So the results of the area remembering system after the border cross (see Section 3.1.2) are compared with the next area according to the path plan. It is important that all of the potential areas suggested by the remembering system are compared. By doing so, a change within the environment has less to no effect because the list of potential areas also contains the old area entries that despite the change, have a tolerated similarity. This has already been explained more detailed in Section 3.1.2.

Chapter 4

Technical Realization

In this chapter the realization of the defined model is shown. The implementation has to be connected with the ARS model and thus relies on the provided framework. The parts that are of special interest due to their connection to this work are the decision unit, the vision unit and emotional and drive unit. While the first one actively uses the system and its resulting information, the later ones provide the input data the system is working with. However, here only the orientation system implementation is described including the necessary extensions within the provided episodic memory system [DGLV08].

In the first section an overview of the system design and the connection between the different subsystems as well as the interface to the decision unit is shown. After a brief explanation of the *controller class*, the clsOrientation, the other subsystems are described in detail. These are named the *area semantic memory*, the *dead reckoning*, the *assumption system* and the *path planning system*. Additionally to these main classes the supportive classes for areas and transitions are mentioned. Further on the necessary extensions to the episodic memory system that was implemented within the ARS system by Andreas Gruber (see [DGLV08]) are shown. These modifications made it possible to trigger event encoding due to occurring transitions and save the travel data in the memory stream.
The final part shows a real world implementation within an e-Puck Robot. This work also made it necessary to design a decision unit and a low level vision system to gather the necessary data for the orientation. This vision as well as the implementation details are shown.

4.1 Overview

The implementation of the ARS model itself is done in JAVA object oriented programming language due to its platform independency and available supporting libraries. The Bubble World simulator utilizes a framework named MASON (Multi-Agent Simulator Of Neighborhoods [MAS09]). MASON is an open source java library core that provides utilities for multi agent simulation in 2D as well as in 3D. It was developed in joint effort between George Mason University's Evolutionary Computation Laboratory and the GMU Center for Social Complexity. Combined with a physics engine extension it serves as the base of the whole simulation environment. This base is currently being extended to serve as a complex artificial world for the autonomous agents called *Bubbles*. With this preset, the orientation system is also written in JAVA language using the open source Integrated Development Environment named Eclipse [ecl09].

As can be seen from Figure 4.1 and the class diagram in Figure 4.2, the implementation consists of 5 subsystems. Each one of them represents one of the main model functions described in the previous chapter. An additional system part that does however not belong to this imple-

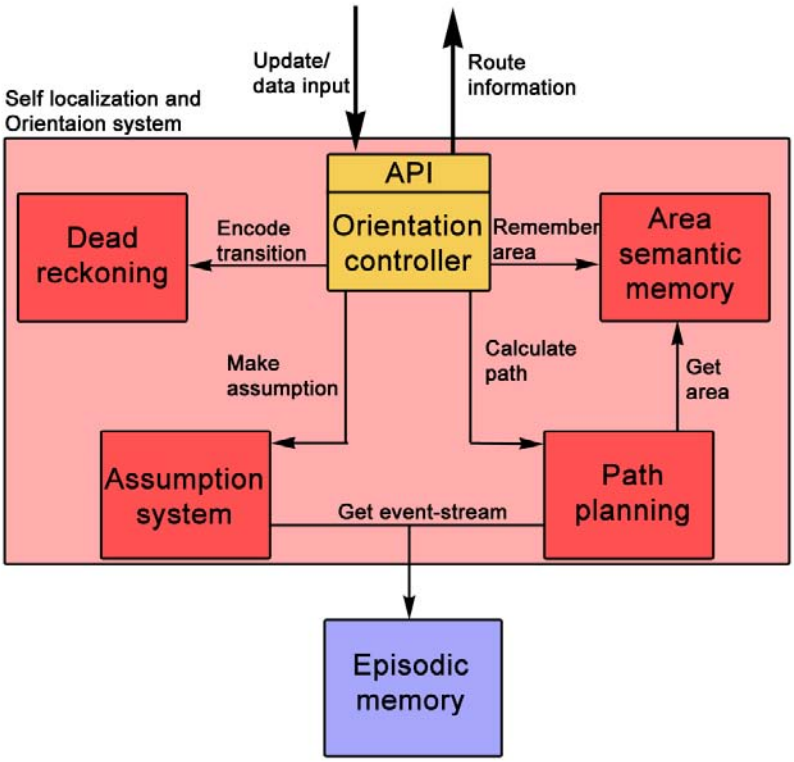


Figure 4.1: System Design Overview

mentation is the episodic memory module which is already provided. Data from this external system is gathered by direct function calls using a link to its instance which is stored within the clsOrientation at initialization time. The orientation system interface itself consists of a single API that provides 3 basic commands. These are the *orientate* function which provides the current inner and external state to the orientation system, the *planPath* function for conscious triggering of the path planning process and the *pathGetDirection* function that is used to get the route information. Besides these commands the system works autonomous to be consistent with the concept of phrenology.

4.1.1 Functional Submodules

The heart of the module is the orientation subsystem that provides the interface to the user and manages all the other components. After getting input information from the user it triggers the recognition process, if necessary starts a path planing, a dead reckoning or an assumption process. It can be seen as the main controller of the whole system. Now to roughly explain the dependences of the subsystems.
Starting from the Orientation part, the controller. It holds the instances of all the other parts and contains the orientate function which is used to hand the vision information as well as the data from the emotion and drive system to the module. With this data the area semantic memory subsystem is triggered to start a process to remember the area which is equivalent to self positioning. Depending on the answer the following steps are decided. In case two consecutive remembering processes have different outcome, meaning an area border has been crossed, the dead reckoning system is called to encode the current transition. Additionally the assumption system is used to make a prediction about the transition outcome. To do so, the previous transitions from the episodic memory are used for comparison. Finally if decided by

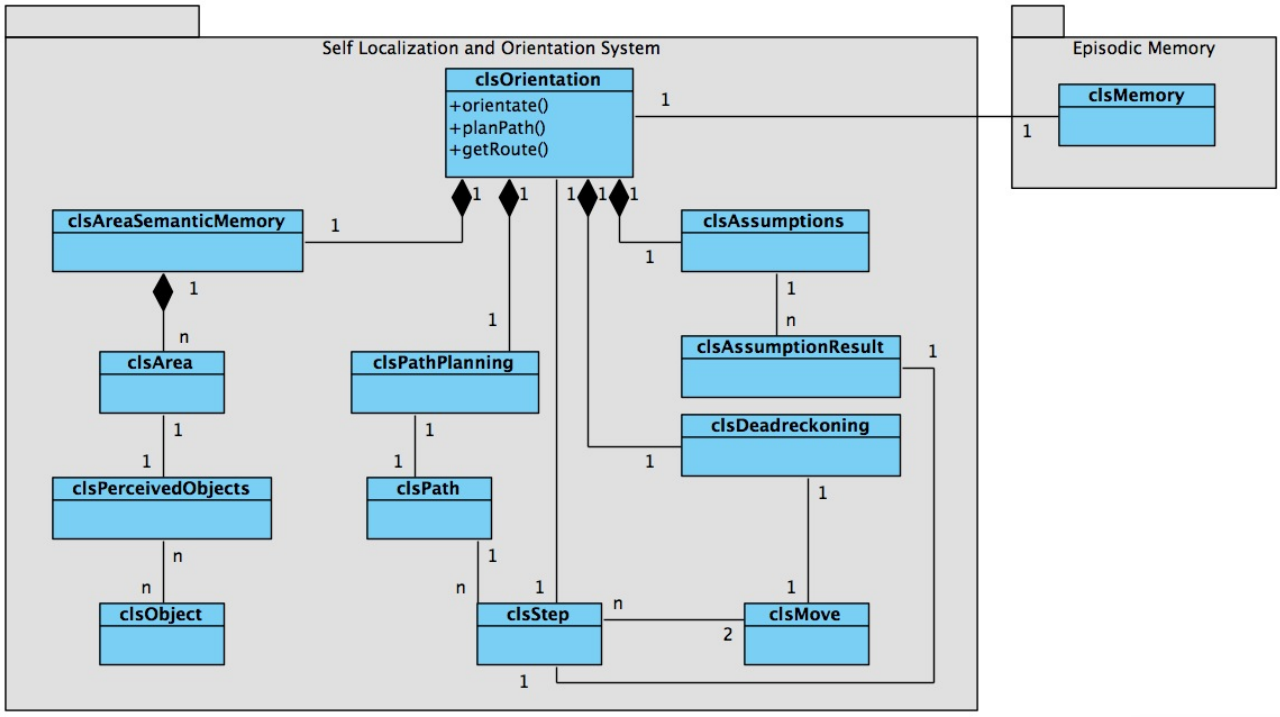


Figure 4.2: Orientation and Localization System Class Diagram

the orientation subsystem or requested by the user, the path planning is activated to plan a route. Therefore it collects information about the previous journey which is saved within the episodic memory and requests area information from the area semantic memory to evaluate the paths.

4.1.2 Self Localization and Orientation Module API Functions

In the module there exists only a single interface which is implemented within the orientation subsystem. Because most of the work is done autonomously, this API only consists of 3 different commands.

orientate: It is the most important function as it delivers the current data about the vision and the emotions and drives to the system as input data. With every call an update process is started, so the call rate also defines the rate of self localization processes. Again this is consistent with the concept of an automatic module defined in the phrenology. As soon as the input data is delivered, the localization process, the unconscious triggering and the assumptions are performed. Usually an update should be done after every movement to keep the position knowledge up to date.
The function requires 3 parameter. First of all the perception data, which is a list of objects within view range, their distance, and their position angle relative to the heading direction. This data is provided by the agent's vision system which performs an object recognition using the data from the vision sensors. The orientation module relies only on the ID number of these identified objects which is assigned by the object recognition. A bad object recognition also results in a low performance of the self positioning. The data itself is stored and handed over within a clsPerceivedObj class which is further explained in Section 4.3.1.
The second parameter is the emotion list. This consists of an array of all the current emotion information. By organizing it as an array and providing the same basic functions

for each emotion the set stays extendable for future implementations. As mentioned before the current ARS implementation includes the emotions fear anger and lust.
Very similar to the emotions the third parameter is a set of drives. Here the ARS implementation has the drives hunger, seek and play. Both parameters, emotions and drives, are used for self localization, in case the environmental information does not lead to a unique result, and the unconscious triggering of the path planing process.
Both, emotion and drive data, are provided within a clsPerceivedImage container which is described in more detail in [Gru07]. Due to the ongoing changes made within the simulator as well as the decision unit this class will not stay in its current form.
The return value of the orientate function is the ID number of the current area. It can be seen as an abstract representation of the current place the agent is located in. This is kind of similar to the knowledge of a person to be located in Vienna.

planPath: The planPath function can be used by the decision unit to consciously trigger a path planing process if desired. When used, the orientation subsystem executes a path calculation process within the path planning unit. The necessary parameter for this function is the ID of the area the agent wants to go to. After the path calculation is finished the function's return value is a boolean value which is true if the planning process has a result, meaning a path is found, or false if not. In case the path calculation is successful the decision unit can retrieve the path information by calling the PathGetDirection function.

PathGetDirection: In case a path has successfully been planed due to conscious or unconscious triggering, by calling this function the decision unit can get the resulting path directions. It requires no additional parameter but uses the latest planned path which is saved within the orientation system. A planed path is organized in steps as will be shown later. A single step represents one transition that has to be performed to get closer to the desired goal area. The PathGetDirection function uses this step information to calculate the direction to the transition that has to be taken next to follow the planned path. The return value is the direction angle in degree to this transition, relative to the current heading direction. In case the agent performs a false transition not contained within the planned path, a new path to the desired location is calculated and new direction information is provided. In case no path or no directions could be calculated due to a missing object, an error is returned and the agent has to travel back until a path can be planed, or it has to continue searching on its own

4.2 Orientation Controller

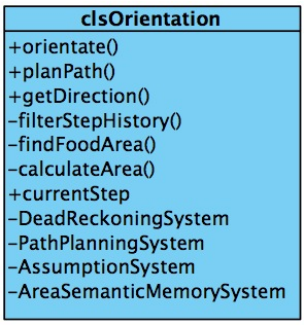


Figure 4.3: Orientation class function Overview

As mentioned before, the clsOrientation acts as a kind of main controller for the whole system. Thus with the creation of an clsOrientation instance, all the depending subsystems, area

semantic memory, path planning, assumption system and dead reckoning, are also generated. In the following, the main functions of this controller are shown more briefly (see also Figure 4.3). These mainly consist of the interface functions orientate(), planPath() and PathGetDirection(), but also assisting functions like filterStepHistory().

orientate Function

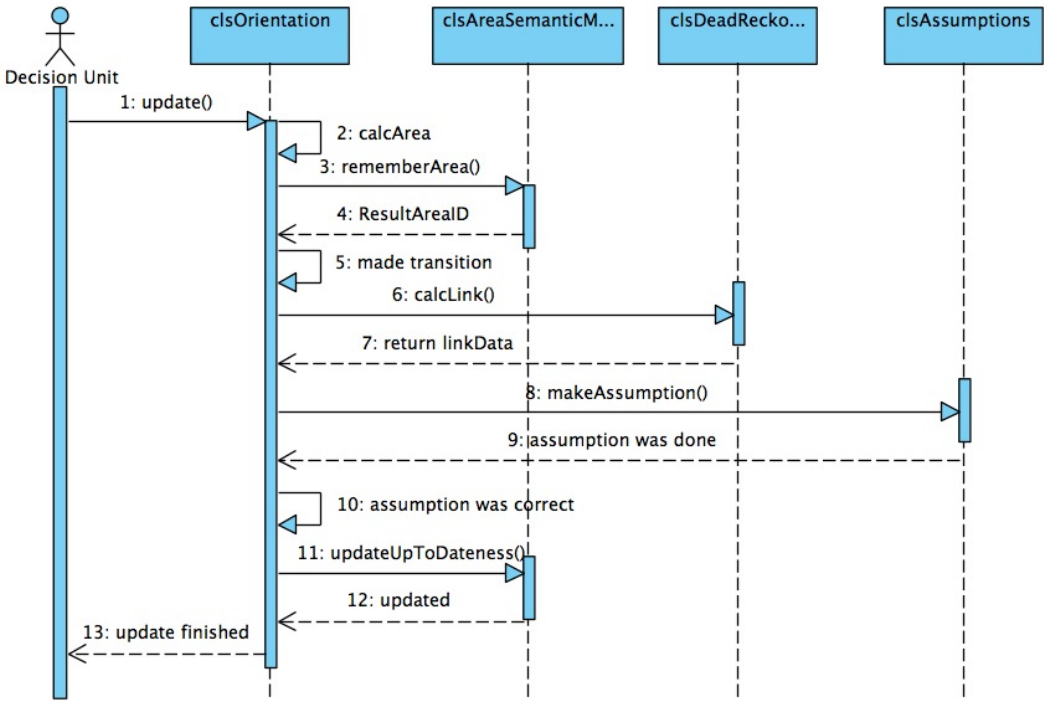


Figure 4.4: Sequence diagram of the orientate function after a transition

Figure 4.4 shows the sequence diagram of the orientate function. It is being called by the decision unit, and depending on the provided data the orientation class performs several update steps. At first the perception data is converted into a clsArea instance by calling the calcArea function which performs an initial normalization as explained in Section 3.1.1. Depending on the object properties, which can also contain *stationary*, the normalized data is then separated into landmark and non landmark objects and stored within a clsArea container (see Section 4.3.1). By doing so, locations that for example contain food source can be found much easier (see Section 3.1.1). In combination with the definition that food sources satisfy hunger, a path to react to this drive can be planned. The clsArea class itself is being explained further in Section 4.3.1. Using the new area, a self localization is performed by calling the rememberArea function within the instance of clsAreaSemanticMemory. The necessary parameters are the newly generated clsArea instance of the current perception data and the tolerance values similarObjectTolerance and similarObjectBearingTolerance described in Section 3.1.2. The actual process of area recognition within this class is described more detailed in Section 4.3.1. The returned information however is a sorted list of potential current areas. The first area entry in the list which has the highest similarity value is assumed to be the correct current one. Based on this piece of information and the past data, it is estimated if the agent just crossed a border and thus made a transition. That is the case when the previous area ID, recognized by the area semantic memory system, does not mach the current one.
In this situation several additional actions are being processed. First the link data is being calculated according to the explanation in Section 3.1.1. It is essential that this step is per-

formed twice. Once for the forward move, from the previous area to the current one, using the previous perception data. Once for the backward move, from the current area to the previous one, using the current perception data. The reason for doing so is that the areas might contain different landmarks and finding the transition point again from within one area requires different information than within the other one. By saving both information within a clsStep container, the transition can be used to travel in either direction. The container, generated by the dead reckoning subsystem, is then handed to the controller class which provides it to the episodic memory system. There a recording is triggered, due to the border crossing, which saves the step data.
The next step that is being done after a transition, is calling the makeAssumption function within the clsAssumption. It causes an assumption to be made according to the definition in Section 3.1.3. Again depending on the outcome of this subsystem additional actions are initiated. In case an assumption could be made, the up to dateness information of the current area is being updated. It depends on the correctness of the assumption whether the up to dateness is incremented or decremented.
With this step the update process is finished and the result, meaning the current area ID is returned to the decision unit. The case that is not mentioned here is the spontaneous path planning due to drive and emotional data. The reason therefore is, that because of the current redefinition of the ARS model the emotion and drive system can not be used for the implementation and testing.

planPath Function

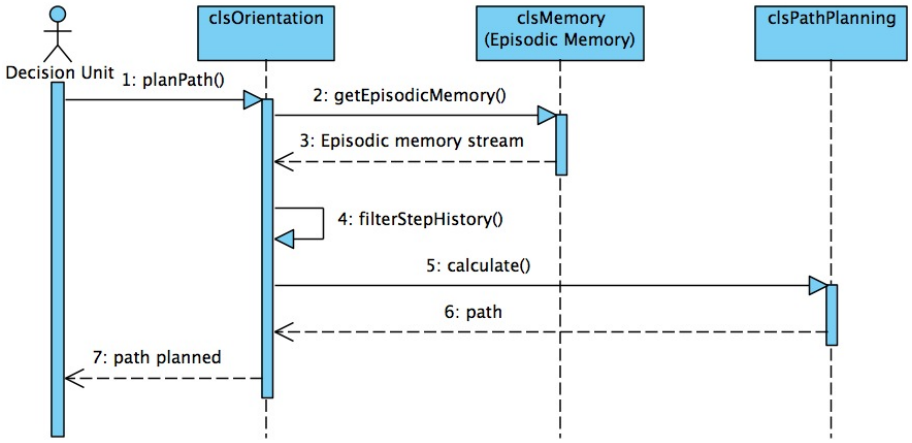


Figure 4.5: Sequence diagram of the planPath function

As it is shown in Figure 4.5, the planPath function call is almost directly forwarded to the path planning subsystem which calculates a path according to the description within Section 3.2.2. As an intermediate step the required input data for the path calculation is generated by calling the filterStepHistory function. This extracts all entries from the episodic memory stream that were recorded due to a transition occurrence. The return value then consists of an array of steps that represent the agents' previous journey. This information, and a connection to the area semantic memory, for getting the up to dateness and fear factor of a specific area, is all that is needed in order to calculate the path. The return value of the calculate function is a path array consisting of several clsStep instances. Such a path construct is shown more clearly in Section 4.3.3 where the whole path planning subsystem is shown. A pointer to the generated path instance then is saved within the controller, the clsOrientation instance, and is used to calculate the direction to go when following the path. As a return value the function gives a

true or false, depending on whether the planning process within the path planning subsystem was successful or not.

filterStepHistory Function

Because the episodic memory stream holds not only those events that give information about the agents' journey, this specific data has to be extracted for functions like the assumption or the path planning. The filter operation reduces the large episodic memory stream to a much smaller list of transitions.
The used episodic memory, as explained in Section 2.4.3, records situations in case a special event like a strong emotion or the occurrence of a transition. For the path planning for example, only the remembered transitions are important, therefore only these are being extracted from the memory stream as can be seen in Figure 4.6. The upper part shows an example event stream. Only the ones that were recorded due to a transition are used to form the path knowledge. It is essential to mention that, since every episodic memory event stores the whole data set consisting of action, emotion, drive and now the latest clsStep Instance, every event contains step data. But not every one represents a new transition. The clsStep instances within every event contain the information about the area in which the event occurred and are thus of great importance. For the filtering this step data is used. It contains the information about the last two orientation system update cycles at the time of recording. In the case of a transition it would be the starting and the goal area. If however there was no transition, both entries contain the same area which is the one the event occurred in. So only those steps are filtered out whose start and goal entries differ.

An other important thing about this function is its additional filtering functionality that should be used when implementing a more sophisticated path planning algorithm like the Dijkstra or the D* shown in 2.3.1 and 2.3.2. Its task would be to filter transitions that lead to or from areas the agent associates with great fear, or has no trust in, meaning it has a low up to dateness. Filtering here eases the later path planning process. In the current implementation however a trace-back approach is used which requires a continuous transition stream. Thus this filtering of uncertain or dangerous transitions cannot be done that way, but has to be done afterwards.

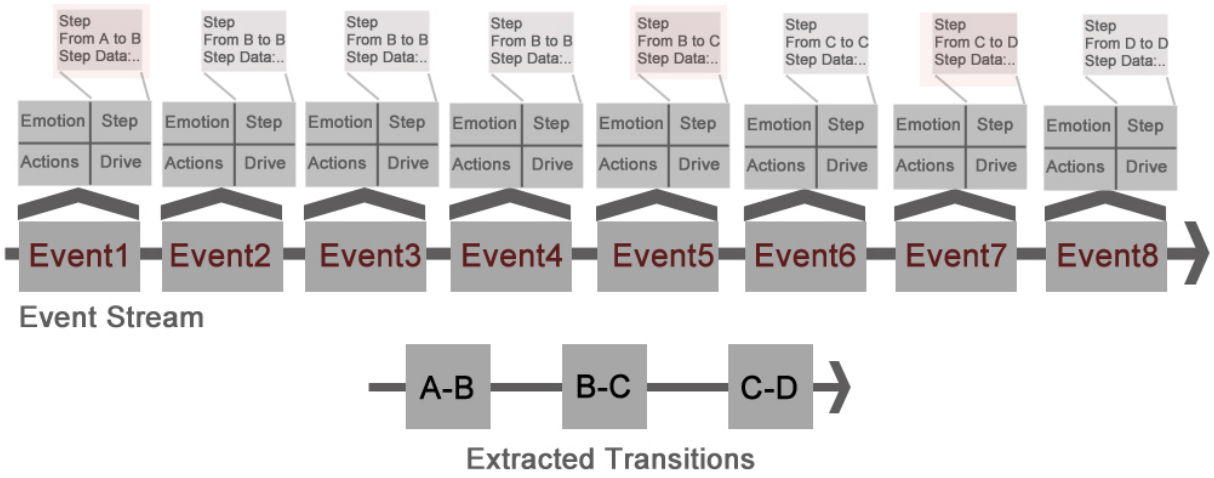


Figure 4.6: Transition memory extraction

pathGetDirection Function

The calculation of the direction is done directly within the orientation class, using, as stated before, the path calculated by the clsPathPlanning instance. It uses the data from the last step (clsStep) in the path instance to calculate the necessary turning angle to face the next transition point according to 3.1.1. Every time a transition has been performed according to the current step, the next step from the path list is set as the current one. The check whether a path transition has been performed or not, is done within the orientate function after a transition was detected (see 3.2.3).
Here the uncertainty within the vision data, the quality of the landmarks and their positioning has a great influence on the path following process and thus requires additional enhancements. A problem that appears is due to the imprecise data near transitions in which the main and the secondary reference object are in 180 degree angle to each other. The reason is that for 2 reference objects (main and secondary reference) the position estimation has 2 outcomes (see 2.1.4). So for the direction calculation, the 2 positions result in a calculation difference of 180 degree which can cause oscillation near the area border. The reason for this is that due to imprecise vision data the transition point might not be located exactly on the area border. To prevent this effect, the soft border that results from the defined transition close area (see 3.2.3) has to have a minimum size that depend on the amount of sensor error. So the passing direction, the same direction the border was passed the first time, is ordered sooner.

4.3 Subsystems

In this section the various subsystems of the localization and orientation are described. These are the area semantic memory, the dead reckoning, the path planning and the assumption system.

4.3.1 Area Semantic Memory System

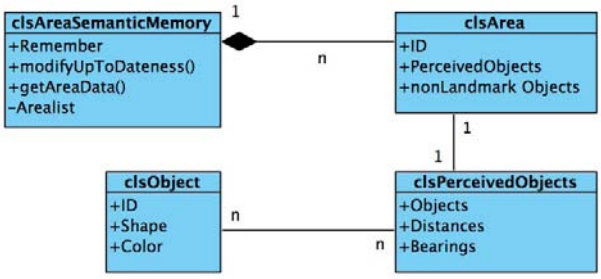


Figure 4.7: Class diagram of area semantic memory subsystem

The area semantic memory has the task of storing the encountered areas and to remember them. With the generation of the clsAreaSemanticMemory instance an array of clsArea instances is set up in which the area knowledge is saved (see Figure 4.7). Every area again consists of the normalized vision data within a clsPerceivedObjects instance and a list of non landmark objects. These are objects like food which are not used for the position recognition but for finding areas that contain an energy source or alike. The smallest data objects in this subsystem are the objects of the type clsObject. They contain the object properties like shape or color and an object ID which is a kind of abstract identification for the real object itself, similar to a name like "Eiffel Tower". In further comparison, only this ID is used to check if 2 object are the same. The actual assignment of an ID to an object is done by the object recognition system that uses data like the shape, color, etc., to identify new, and remember known objects. This means the

object recognition converts the objects within viewing range into clsObject containers and stores them together with their position data within a clsPerceivedObject instance.

Interaction with the area memory is only done by either starting a remembering process, an up-to-dateness update and readout or a direct call for an area entry for a link calculation. The most important function of course is the remembering function which is the actual self localization. Not in terms of coordinates, but by knowing in which of the remembered areas the agent is currently in. The function call itself requires normalized perception data stored within a clsArea container. This container is generated and filled with the perception data by the orientate function (see Section 4.2) during the normalization. The remember() function either adds an ID and stores it within the memory, in case it is recognized as a new one, or deletes it if a similar one is remembered. As shown in Section 3.1.2 the remembering process consists of two steps which are, filtering according to similar objects within the area and filtering according to the objects' location. Both uses the information stored within the clsPerceivedObjects class. In Section 3.1.2 it is also stated that these 2 steps each calculate a similarity value for the potential areas. This means, that in the end every area has an object similarity and object position similarity value. In the filtering process these values are used to remove the areas with a too low value from the list of potentials. In the end of the filtering however this value is now also used to sort the list according to the matching. Figure 4.8 shows an example result. The potential area list holds all the areas whose calculated similarity values are high enough to pass the filtering according to the defined tolerances.

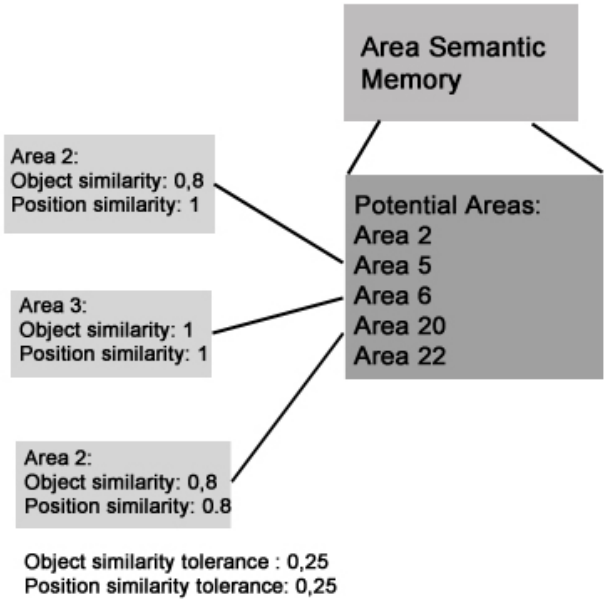


Figure 4.8: Object and position similarity values of the potential areas

The first entry within this list, which has the best matching, is assumed to be the current one. If this best calculated match however is not 1, meaning the current area is not 100% the same as the remembered one, the current area is also recorded as a new one, since it might actually be a new one. This situation can occur when the agent arrives in an area that looks very similar to an already known one, or when the area has changes for some reason. In either case the new, or the changed area is saved. In a later encounter this new recorded area will have a higher match and thus will be recognized correctly. In the case of a changed area this means that the old version is being forgotten for a standard self localization process. With this implementation it is not possible to tell whether a specific area has changed. Therefore the

old version cannot be updated and is still present within the episodic memory as part of the recorded path. Since path planning also uses this information, it might occur that the path includes an old area. This means that for the special case, where a transition to an area defined by the path is expected, not only the best match is checked. Additionally it is searched whether the expected area is among the potential ones. If so, the path part is said to be completed and the next one gets active.
An other trigger for recording the current data as a new area, is an empty potential area array after the filtering process. In this case the area is clearly unknown and has to be saved.

4.3.2 Dead Reckoning System

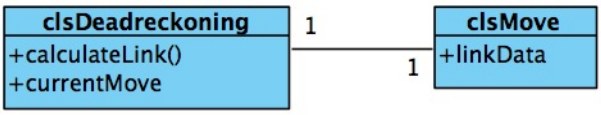


Figure 4.9: Dead Reckoning Class Diagramm

The dead reckoning subsystem is an essential part in the whole network map generation. It holds and updates a clsMove container with the required information to find a transition point from within an area. For the calculation the controller class calls the calculateLink() function and provides the current raw environmental data. In this case, raw means non normalized (see Section 3.1.1), which is essential to record the current heading direction relative to the main object.
The second necessary information for the calculation is the recognized current area. For easing up the process, the main and secondary objects are defined as the first and the second object within the list of clsObject containers held in the current clsArea instance. In later implementations, this selection could be replaced by a selection according to priorities. The main object should be the one known to be most solid and most unlikely to move. But since the simulation and the object recognition are not yet that detailed, the simpler version is very much sufficient.
Depending on the environmental data, the link calculation is performed 0 - 2 times. The usual case would be that the agent travels from one area to an other, both containing at least 2 objects. In this case the calculation is performed twice. Once using the perception and area data before the crossing and once with the data set after the crossing. By doing so, the transition point can be found from within each of the neighboring areas. In case one or both areas only contain 1 or 0 landmarks, a link cannot be calculated since the transition point could not be located precisely. Therefore the link calculation is performed less times. This also has an important impact on the traceback version of the path calculation because without a findable link the journey record becomes leaky. Meaning that the step history does not form a continuous stream that can be followed. Due to this fact the path planning looses quality. To prevent that effect, enough landmarks are being provided to ensure every area to have at least 2 landmarks to perform the orientation. The larger amount of orientation points also affects the self localization since it becomes more tolerant to errors and changing. Thus the larger amount of Landmarks is to be added any way. Also humans could not orientate within a homogenous room containing only one landmark in the middle (see Section 2.2.1)

4.3.3 Path Planning System

The path planning subsystem has the single task of generating a new clsPath instance and filling it with the necessary steps to travel to a desired area. So the only interface to the orientation controller is the calculatePath() function called due to conscious or unconscious triggering. For

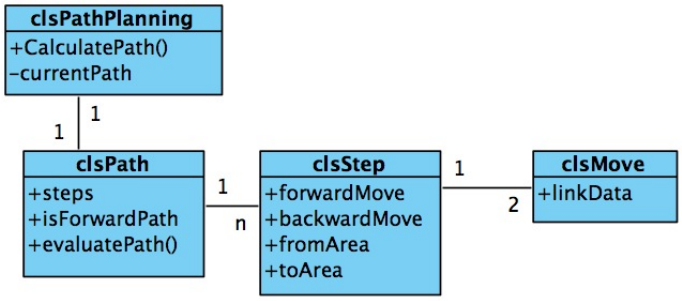


Figure 4.10: Path planning class diagramm

the process it only requires the ID of the goal area and the agents journey history. This history is extracted from the episodic memory by using the step history filtering function mentioned in Section 4.2. It basically consists of a list of clsStep containers that are generated by the dead reckoning subsystem when performing the link calculation after a border crossing. Every one of the steps represents a single transition point that has been encountered before. It holds information about the areas it connects and holds the data on how to find the transition point. As can be seen in Figure 4.10, the actual link position information is saved within the smallest path description unit, which is the clsMove container. It is needed to find a transition point from within an area. Since a transition location is surrounded by 2 areas and has to be found from within each of them to make the transition passable in both directions, 2 clsMove instances are required to define a step. Depending on whether the path is a forward or a backward path, which is also saved within the clsPath instance, the correct move data is used to calculate the traveling direction.

In the current realization, the path planning takes a cohering part of the journey steps and saves it within a clsPath container. This trace-back planning process is shown in Section 3.2.2. The resulting clsPath instance is then returned to the orientation controller, which uses it to calculate the required traveling direction to follow the path. Since the situation might occur that the process results in several paths, the best one is found by evaluating the available ones. The evaluation is performed within the clsPath container. As mentioned before, the emotional system could not be used for testing and thus the evaluation based on the emotional cost factors as described in Section 3.2.2 is not implemented yet. Therefore the path quality is estimated by using the path length and the up-to-dateness of the intermediate areas. The one with the highest up-to-dateness and the lowest length is suggested to be the best one, while the priority is set on the first factor. This quality value is not only important to find the best of the possible paths, but also to filter the ones that are absolutely not acceptable due to their length, the up-to-dateness or emotional costs
All other but the finally selected path are being deleted. The chosen path is kept within the controller class until it has become obsolete. This can be because it has been noticed that the path is not being followed (see PathGetDirection function in Section 4.2) or when a new path has been calculated. The steps within the path however are not being deleted but are kept within the episodic memory.

4.3.4 Assumption System

As can be seen from Figure 4.11, the connection between the controller and the assumption subsystem is the function call to make an assumption. For this purpose, similar to the path planning, a step history list of clsStep containers is created from the episodic memory stream. According to Section 3.1.3 it then performs a comparison of the current step data and the history

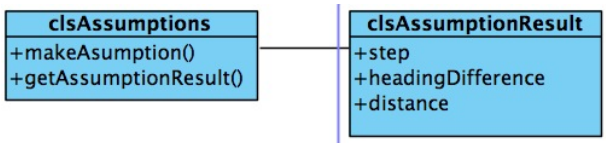


Figure 4.11: Assumption system Class

information. Therefore the dead reckoning, which calculates the current step data, is performed prior to the assumption after every recognized border crossing.
When an assumption process is successful, meaning the system is able to make a prediction about the transition outcome, the result is saved in a variable that can be called by the controller class. This result is a list of potential transitions that are similar to the current one. The best of these results is calculated and returned by calling the getAssumptionResult function.

4.4 Episodic Memory extension

The episodic memory system as described in Section 2.4.3 was designed with triggering functions for extreme emotions, drives or special actions. Every one of these three kinds of states was given a triggering function that can cause a recording. These trigger functions are explained in [Gru07], pp 75. To additionally add the ability of saving transition data, it is necessary to add

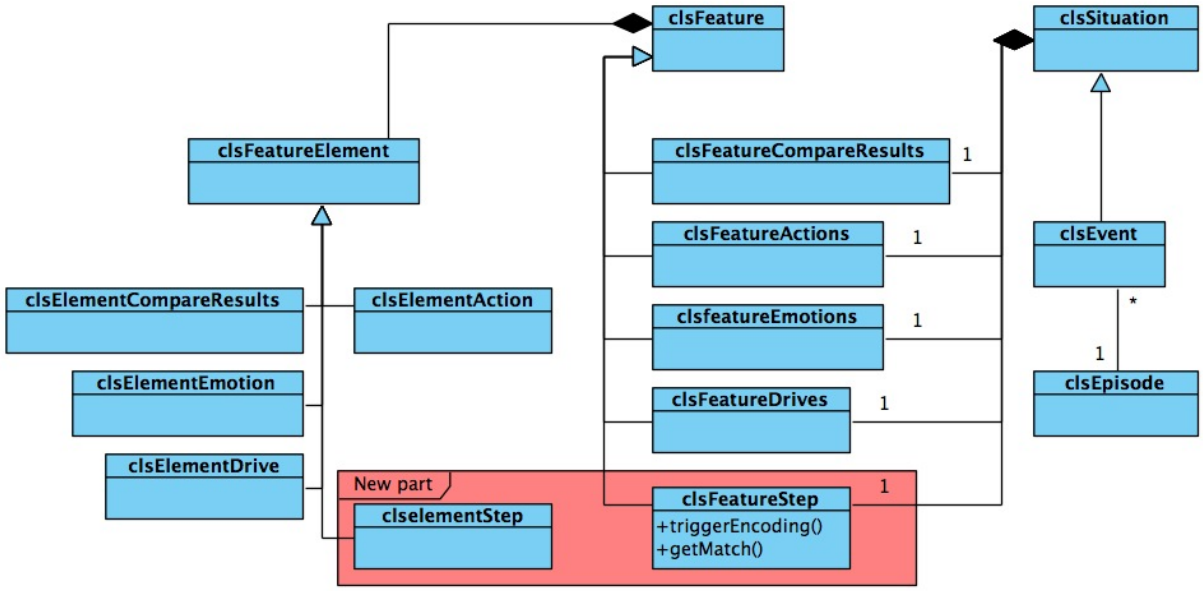


Figure 4.12: Class diagram of episodic memory class structure with extension for step recording

an appropriate container for the clsStep objects. These have to be equipped with the necessary triggering and comparison functions. This makes it possible to encode as well as to remember certain events according to the transition occurrence. In Figure 4.12 the necessary extensions are shown in red. Two additional classes are added that contain the clsStep instance and the necessary functions. The knowledge about the current area an event is recorded in, is taken from the saved step. With every recorded event, also if it was not triggered by a transition, the current clsStep instance is gathered and also saved. In case there was no transition when the event was recoded the fromArea and the toArea both contain the same entry since the agent traveled only within one area. In this case both entries represent the current area. This information is used for the encoding triggering function within the clsFeatureStep class. In case

the previously recorded event occurred within an other area then the current one, meaning the fromArea differs from the toArea, the trigger function causes an encoding.

Besides these mentioned classes, the initial update function of the memory system is modified to except step data as additional input besides the action, emotion and drive data.
To create a connection between the orientation system and the episodic memory transfer functions that hand over the memory stream are added. This function can then be called using a pointer to the memory system which is stored within the orientation system at the initialization step. With this connection the necessary historical data can be retrieved that is necessary for the path planning as well as for the assumptions.

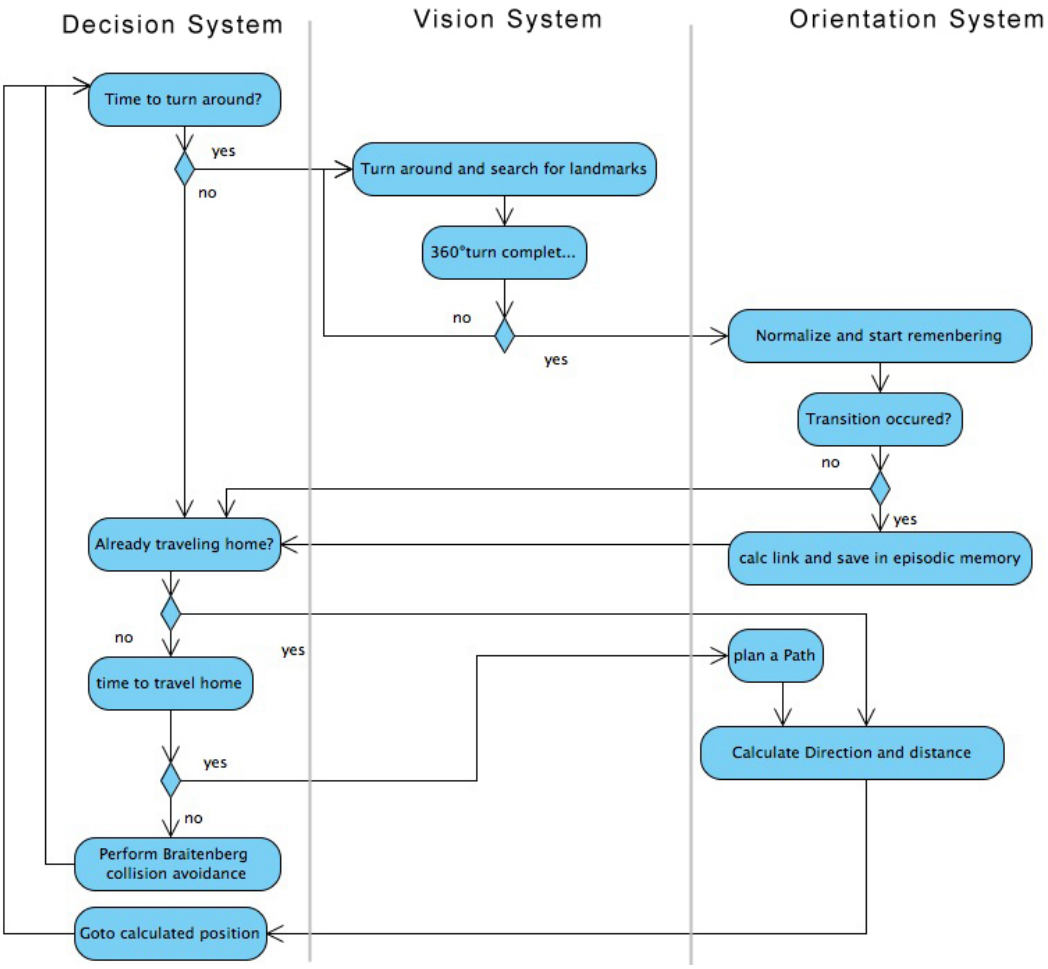


Figure 4.13: Decision chart of the e-Puck implementation

4.5 E-Puck Implementation

Testing and using a system in a virtual environment is one thing, but confronting it with real world conditions can be very different. Still, as explained in Section 2.6, to check the basic system functionality a virtual environment is sufficient. But to test the influence of sensor error, a real world implementation is favorable. Therefore such a real world implementation is done within an e-Puck educational robot (see Section 2.6). The goal is to explore how applicable the system is when confronted with data that contains much more uncertainty. This imprecise data like bearing error, distance error, frequently false landmark recognition and many more, can

have a huge impact on actual orientation performance. Although these problems are caused by a poor vision system, they should still partly be compensated by the orientation algorithms.

Since this orientation system is just a kind of add on module that requires other parts to function, these parts have to be designed first. These depending parts are a basic decision unit that uses the orientation module for finding its path, and a vision system that provides the necessary input data. As both are not really part of this work, only very basic versions of both are used. Basic by the means of, only containing the functionalities that are needed for testing the orientation system. This includes a decision system that travels around, performing collision avoidance and frequent orientation, and having the urge of returning to a certain base area periodically. In case of the vision system it is necessary to recognize a limited amount of landmarks, their distance to the robot and their relative bearing to each other. Figure 4.13 shows an abstract decision chart of the whole implementation. In analogy to the java subsystems described in Section 4.3 several functions can be segmented according to their purpose. More detailed information about these separated systems are shown in the following.
The whole implementation is converted from the java version into C code to make it leaner and to raise the compatibility with the Webots framework (see Section 2.6) and the new vision system.

It has to be pointed out that the e-Puck robot is picked on purpose. As can be seen from the description in Section 2.6 it is a very low level robot with very limited resources. This refers to the controller, in terms of available memory and processing power, as well as to the available sensors. The reason for using such a robot, is to show that the proposed localization and orientation system requires only very little resources to operate and to guide.

4.5.1 Decision Unit

As mentioned before, the decision unit for the testing purpose need not to be very complex. Thus it mainly consists of a basic traveling algorithm that causes the robot to travel around and to avoid obstacles. Besides this traveling function the decision unit periodically performs a vision update and an update process where the self localization and the transition savings within the episodic memory are performed. To test the orientation ability, a $goHome()$ function is defined that causes the robot to start a path planning to a defined area and to travel according to the resulting path until the goal is reached. Of course this is a simple decision unit as it follows a pre defined pattern and has only very limited data to base its decision on. But since in this case it replaces the ARS decision unit it is named similarly.

Because vision and orientation update are basically function calls within the subsystems, the main part of the implemented decision and control unit is the traveling. Therefore a collision avoidance based on the Braitenberg aggression theory [RS05] is implemented. This theory links the modified sum of the distance sensor values to the wheel speed of the left and right wheel sets, causing them to accelerate or slow down. Doing so results for example in a lower left wheel speed in case an obstacle appears on the robots right hand side, and causes a turn to left side, avoiding the object.

$$V_L = \sum_{i=0}^{7} B_{L,i} \times S_i \tag{4.1}$$

$$V_R = \sum_{i=0}^{7} B_{R,i} \times S_i \tag{4.2}$$

4.1 and 4.2 show the equation for this method. The left and right wheel speed (V_L, V_R) is defined by a set of Braitenberg coefficients for the left and the right side $(B_{L,i}, B_{R,i})$ and the sensor values (S_i) of the 8 available proximity sensors. The set of used coefficients is shown in table 4.1.

i	1	2	3	4	5	6	7	8
B_R	150	100	80	−10	−10	−10	−15	−35
B_L	−35	−15	−10	−10	−10	80	100	150

Table 4.1: Table of Braitenberg coefficients used for implementation

In case no obstacle is in range this function causes the robot to travel straight on.

4.5.2 Vision System

The vision system is the most important part since the quality of the information it provides is directly related to the performance of the orientation system itself. As described in Section 3.1.1 the necessary informations are the bearing and the distance of recognized landmarks. Since the range of the proximity sensors is too low to be used for the distance recognition, the only alternative is the CMOS color camera module. The recognition and the calculation of the bearing and distance is described in the following

Color detection is used to identify the red, blue and green landmarks as such. For example image areas with a relative green intensity, meaning $intensity_{green}/(intensity_{blue}+intensity_{red})$, higher than a defined threshold are recognized as green landmark. Moreover the dimensions are being stored for further calculations like the distance estimation.
For calculating their relative bearing to each other, odometry data is taken. As described in Section 2.1.2, odometry causes an accumulating error during usage over time. For short time usage, like a turn around the robots' z axis, the error stays very low. Because no omnidirectional vision is available, such a 360° turn is necessary to receive a full image of the surrounding. During the turn, the bearing is constantly monitored and the value is saved in case a landmark is recognized. So in one full turn all landmarks, their bearing values and their observed widths are obtained.
As for the distance detection, which has to be done also by using the camera, the recognized width of the landmark is used for calculation. Due to the similar cylindric shape of all the landmarks the distance calculation is the same for all of them. Tests showed however that due to the low resolution and the update rate of the camera, the relation between distance and object width is not linear as it should be. This data is shown in Figure 4.14. The blue circles indicate the taken landmark width measure points from 70cm to 15cm distance. The closer the robot is to the landmark the more fluctuation is in the width data, and the error rises exponentially when coming closer than 15cm. This leads to a minimum distance of 15cm for accurate distance estimation. Due to the non linearity an approximation function that fits the measured data is used. Here it is an exponential function shown in equation 4.3 and displayed as black line in Figure 4.14. With this function the distance can be calculated with an average deviation of 4cm at 15cm to 70cm with an average deviation of 2cm.

$$Distance = 130 \times e^{-0.11 * width} + 16 \tag{4.3}$$

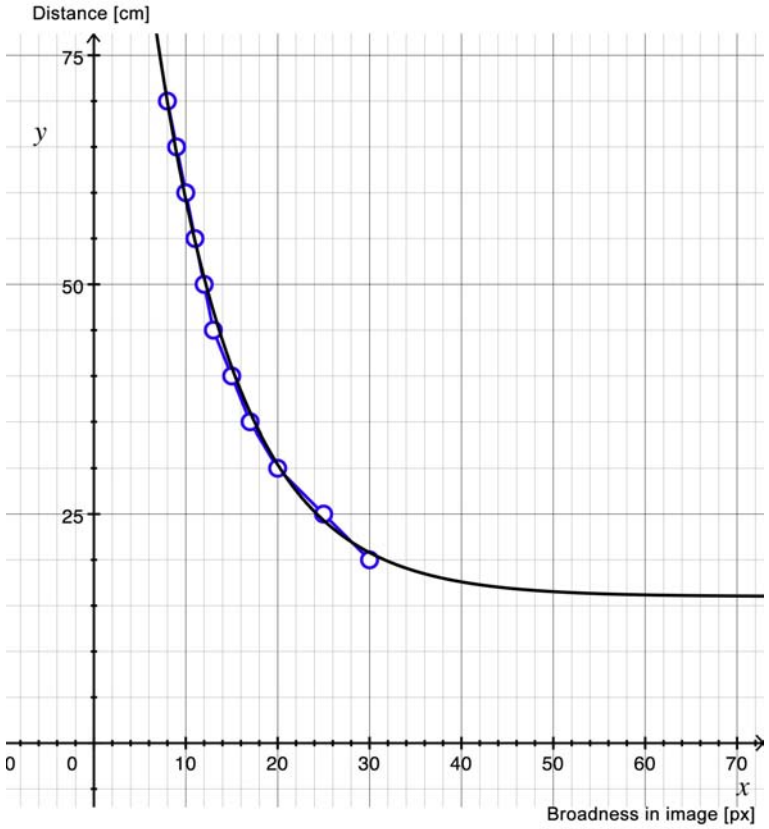


Figure 4.14: Function generation for e-Puck distance calculation

4.5.3 Orientation System

The orientation system for the testing within the e-Puck robot holds the most essential parts of the Java implementation. It consists of the episodic and semantic memory system that are responsible for saving the area and transition data. Therefore special data types for areas and transitions are defined. The maximum amount of recordable areas is set as 255 and the predefined maximum count of landmarks is 5. Doing so resulted in lower memory consumption due to the fact that variables like areaID in an area type or fromArea in a transition type, could be set to an 8 Bit number. This limitation causes trouble only in very large environment with many landmarks. A usually used area for testing, representing a large room which contains 4 to 5 landmarks, contains less than 100 different areas. An example for this will be shown later in the evaluation Chapter 5. When using these values however, a saved area requires only 34 bytes and a transition 22 bytes. Counting this in an environment that results in 100 potential areas, the requirement is 3,4 kByte. The episodic memory for the transitions is organized in a ring buffer which causes old transitions to be overwritten which has the simple effect of forgetting the path over time. Depending on the available memory, this ring buffer can be defined as needed.

Besides the memory, a path planning unit similar to Section 4.3.3 and a controlling orientation unit like in 4.2 are present. So, despite the missing assumption system, the implementation within the e-Puck has the same functionality as the java implementation for the virtual environment. Since the e-Puck implementation is mainly used to test the effect of real time sensor error on the self localization and the path following, no assumption system is required.

Chapter 5

Evaluation

The system evaluation is done in 3 steps. Each of them within the virtual environment, using the ARS framework, as well as in the real world environment, with the e-Puck implementation. The first step is a static self localization which means that the agent tries to locate itself within an environment. Therefore the environment is already known due to a prior fictional exploration. It intends to simulate a situation where the agent awakes and has to reorientate itself before starting to go anywhere.
The second test is a dynamic self localization with path recording. Here the agent has to learn the areas and to record transitions while it travels around. This test intends to check the recording ability of both the semantic and the episodic memory system.
In a third and last test, the full system becomes active. Not only are areas and paths learned during traveling, but after a periodic time, the agent is forced to try to return home to an encountered food area. Therefore the path calculation and a path following are in action.

In the following, more detailed descriptions about the several tests are shown, followed by the results and discussions about some data. For each test the results within the virtual and the real world are shown separately.

5.1 Static Self Localization

The static self localization simulates an awakening agent within a known environment. This means that the semantic memory system is already filled with the knowledge about all the possible areas. The remembering function then tries to retrieve the correct area from the memory. So the self localization is the essential part whose performance is being tested here. The initial segmentation is done by performing an area calculation for every location on the map. The different areas that result from this initial step are then saved within the semantic memory.

An important thing to mention is, that the landmarks are given stationary property values. Since the most stable object is to be used as main object (see Section 3.1.1), it is ensured that the areas contain the similar main object. This is very important for the testing purpose. Without it the area segmentation would always look different, depending on the exploration path. This means that different paths can cause a different list arrangements of the same landmarks and thus different main objects since the first entry is used as such. Now a sorting process is performed during the normalization function, to find the most stable object.

Test in Virtual Environment

The test within the virtual environment is going to show effects of several error sources. The reaction to uncertainty in the landmark recognition, the results when the area contains several

similar landmarks and the output due to an environmental change. Figure 5.1 shows the output of the initial segmentation process. In this environment, that contains 4 landmarks, 30 different areas are recognized when using a 40 % object similarity tolerance, a 0 % position similarity tolerance and an angular-tolerance of 90°. Since the tests simulate orientation in a single but large quadratic room, the range of sight is set to be 80% of the room length. These values are also used in the following test runs. The found areas are displayed in different colors. We see here, that from such a landmark arrangement nearly no scattered area distribution occurs. The number within the landmarks identifies their ID as they are recognized by the object recognition system. In this case with perfect conditions, all the available landmarks still exist in the correct place and are recognized correctly. Since the uncertainty in the landmark position estimation within the virtual environment is negligible, 100 % of the awakening processes are correct. This result is expected because the initial segmentation itself is also performed with the same perfect data and the same algorithm.

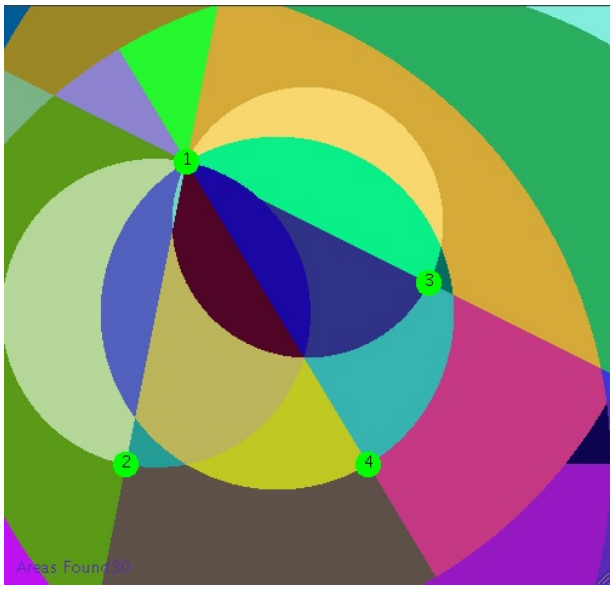
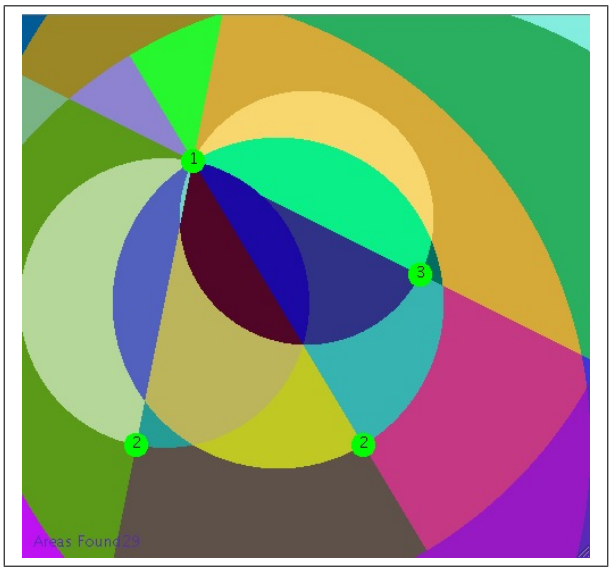


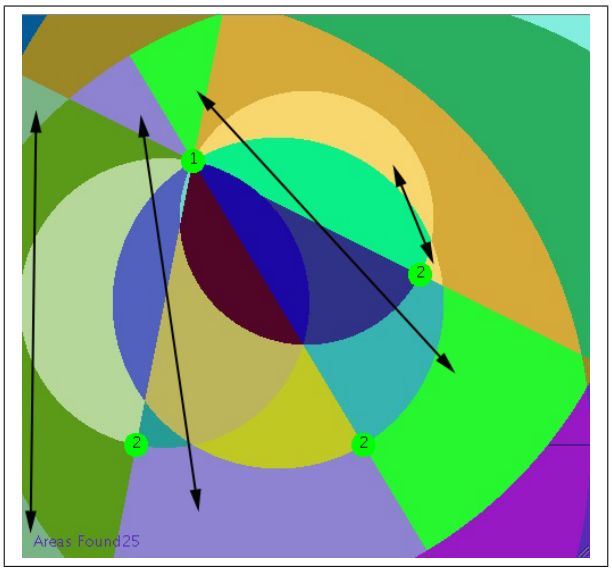
Figure 5.1: Initial segmentation output

Similar Objects The output becomes much more interesting when a certain degree of error is introduced. The relative landmark location data still remains perfect, meaning with an error of only a few degree which might result from type conversions within the java code. It makes no sense to add uncertainty to the sensors for testing purpose since this is done later by using real world data within the e-Puck robot.
The landmark recognition however is very important to test in this environment since it might always occur that several similar landmarks exist. Due to the limited amount of properties of the landmarks it might happen that the object recognition recognizes them as similar. Therefore the same scenery that was segmented with perfect conditions in Figure 5.1 is now used again with different amounts of similar objects.

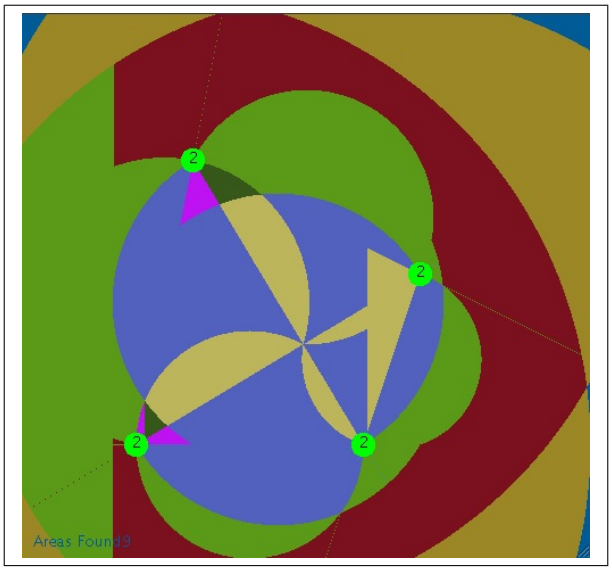
Figure 5.2 shows the outcome of this test. Again the various colored regions represent the different recognized areas. The first thing that can be noticed is that the amount of recognized areas is lowered from 30 in the initial segmentation to 29 with 2 of 4 objects looking similar (see Table 5.1). When having only one unique object the count goes down to 25. This lowering is mainly caused by scattered areas highlighted by the arrows in Figure 5.2(b) and mentioned later. In only one case it is due to merging of neighbored areas. This means that even though landmarks look alike the area border distribution stays



(a)



(b)



(c)

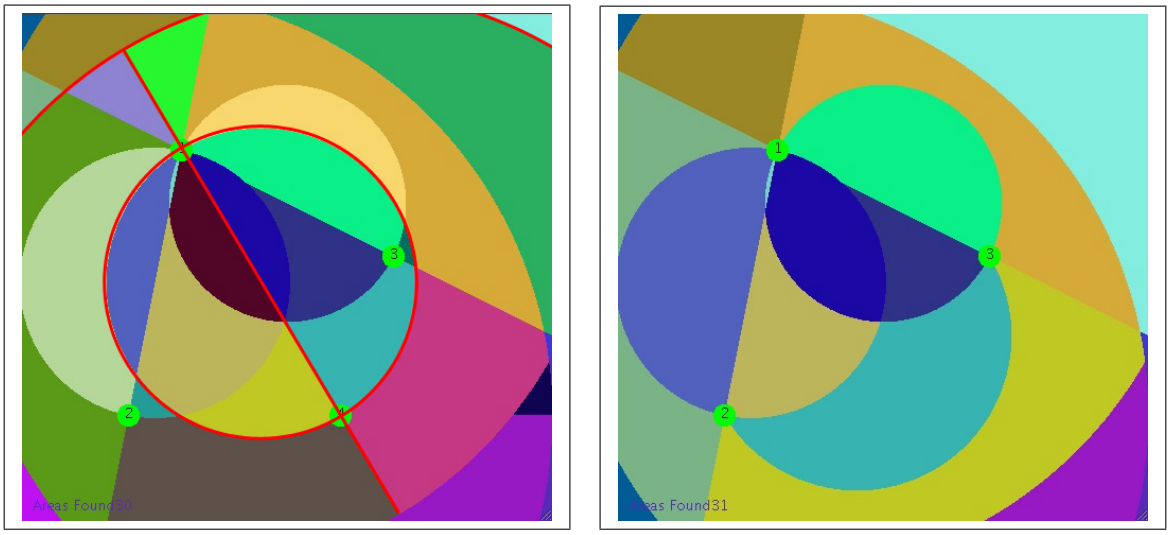
Figure 5.2: Self localization test in environment with different amounts of similar recognized objects

nearly the same, as if they looked different. The important factor is that until one of the objects stays unique and still can be referred to as the main object the area segmentation stays almost stable. If however this main object begins to float, the system breaks down as can be seen in Figure 5.2(c). Only 9 areas are found and no localization is possible any more.

unique landmarks	4	2	1	0
found areas	30	29	25	9

Table 5.1: Found areas depending on the amount of unique landmarks.

The second problem that results from similar looking landmarks is a scattered area distribution. As mentioned before this refers to two sums of locations that belong to the same area but that have no direct neighborhood connection to each other. While in 5.2(a) no such scattering is included, it occurs in 5.2(b) in 4 different areas. This means 13,7 % error probability. The same test is performed with an amount of 5 landmarks within the environment whereas in the ideal situation 54 Areas are identified. It results in an error probability of 14,8% in the extreme situation where only one landmark is unique and the other 4 are identified as similar. This percentage results from 4 areas that have a scattered counterpart.



(a) (b)

Figure 5.3: Self localization test in environment with disappearing landmarks; Settings: angular-tolerance 90°, Similar Object Tolerance: 40%, Similar angular-tolerance 0%

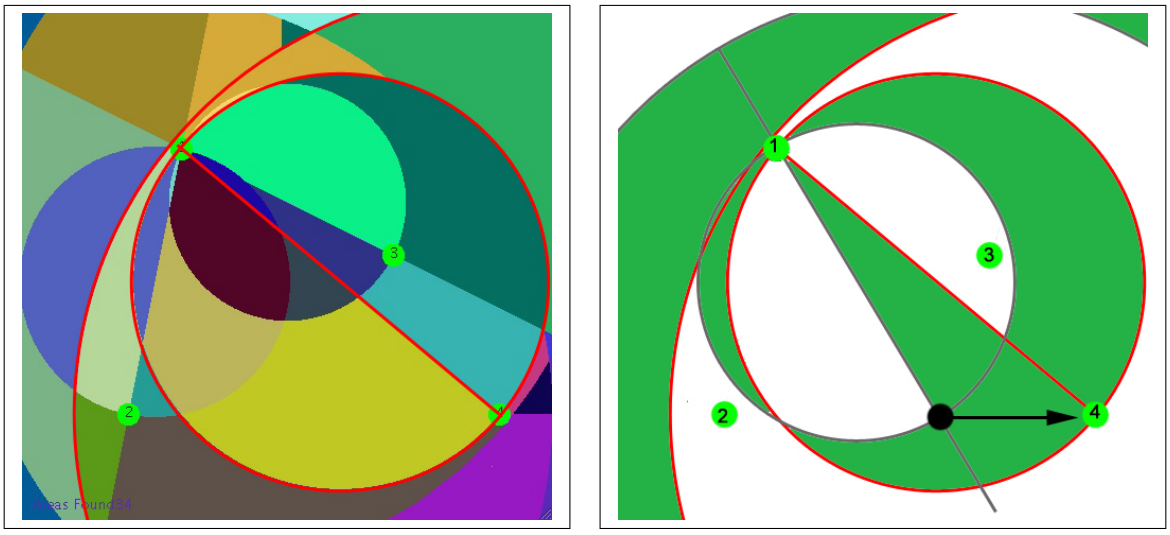
Disappearing Landmarks The next possible and much more severe error source is an environmental change where objects used as landmarks disappear. The result from such a situation, where one of 4 Landmarks vanishes is shown in Figure 5.3. This can be seen as a 25 % change in the environment which is in the tolerated range of 40 % similar object tolerance that is defined for the tests. This change results in loosing all the borders that are defined due to this landmark. Because it is in sight or because of its relative position to the main object. These borders are highlighted in red in Figure 5.3(a). With full knowledge about all the possible areas, it happens that after the environmental change, neighbored areas at this borders can be mixed up. The reason for this to happen is, that after the similarity ranking explained in Section 3.1.2, the both areas have the same like-

liness of being the current one, since the difference lies in the non present landmark. This leads to 11 areas merging with their neighbor and thus the inability of identifying them precisely. In numbers, this means a potential error of 36.3%. Table 5.2 shows the potential error with different values of environmental change. This relation between environmental change and potential error shows an almost linear behavior with 1,5 times the potential error as there is environmental change.

Environmental Change	Potential error
16.6%	23%
20%	29.8%
25%	36.3%
33.3%	53.2%

Table 5.2: Amount of potential error depending on the amount of environmental change

Again the most fragile part is the main object. In case it disappears the whole remembering system fails because it is required as reference object for the self localization. In future improvements (see Section 6) this importance and thus the fragility can be lowered by not only referencing a landmark bearings to the main object but to all others defining the area.

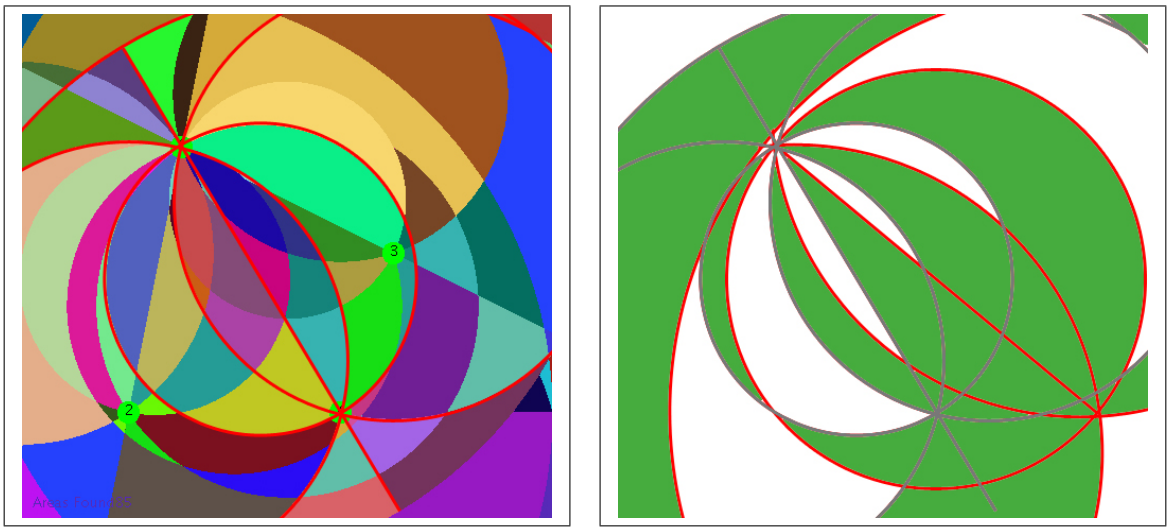


(a) (b)

Figure 5.4: Self localization test in environment with moving landmarks; Settings: angular-tolerance 90°, Similar Object Tolerance: 0%, Similar angular-tolerance 40%

Moved Landmarks In this case, all the borders that are caused by the moved landmark also change place. This leads to a different distribution of the known areas as well as the creation of new ones if the misplacement gets too large. A long range of tests shows that the self localization errors occur in very specific regions. In those the borders pass during the movement, due to the moving landmark. The same example setup (see Figure 5.3(a)) as for the disappearing object test is used. The amount of landmarks and the tolerance settings indirectly influences this error rate. Landmark count and tolerance settings change the amount of possible areas within the environment and thus the error rate varies. The reason for this error however is better visible in such a simple setup.
Figure 5.3(a) shows the initial setup of the landmarks. Landmark 4 then changes its

position for some reason. It is to be pointed out again that landmarks should always be objects that are most unlikely to move. A person who is explaining a path to someone would use street signs or buildings as waypoints but no parking cars that might already be gone or have moved. The result of such an unwanted landmark location change is shown in Figure 5.4(a). It is easy to notice several changes in the area distribution. In both figures the area borders caused by landmark 4 are highlighted in red. If the landmark moves to its new position these borders also move with it. The locations they cover during this movement are shown in Figure 5.4(b). The old border position is marked in grey while the new is in red. The green regions in between are the described touched locations in which the area distribution changes occur. In positions that are within this region the self localization result contains several potential areas because of the defined tolerances. Due to the moved borders the best fitting of these results is in almost all cases an other one than before the movement. This "correct" location is among the results but its similarity value is not the highest. This effect could be lowered by adding an other information source like the distance to the landmark (see Section 6). Based on the position change and the known shape of the caused borders, the error area can be calculated. This error area usually gets larger with a lower angular-tolerance, like 45 °for example. The border shape and the change in this case is shown in Figure 5.5. Again in Figure 5.5(a) the initial setup before the change. Figure 5.5(b) displays the initial borders in grey and the new border setting due to the moved landmark in red. The green regions are the places the borders cross during this move and which cause error.



(a) (b)

Figure 5.5: Border shape and Error area with settings: angular-tolerance 45°, Similar Object Tolerance: 0%, Similar angular-tolerance 40%

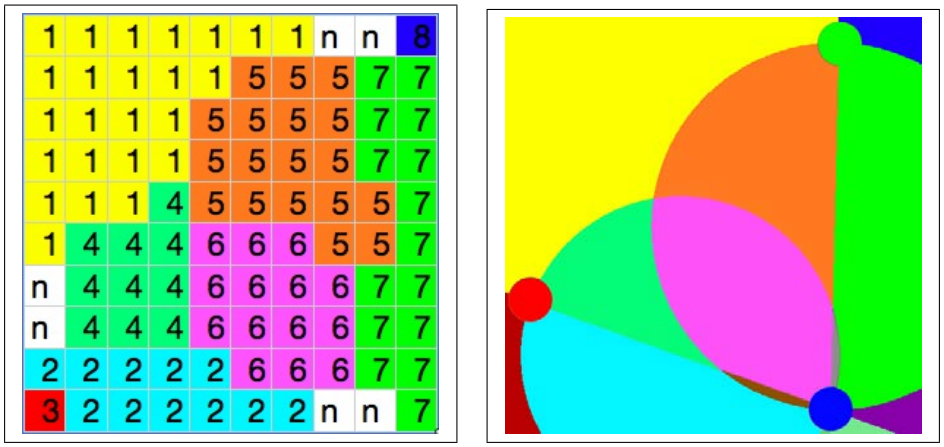
Again the borders before the move are shown in grey and the final ones are red. It can be seen that in this case there exist more borders and because of this the region they cover during the movement is also larger. As a consequence a greater number of locations is misinterpreted.

Test in e-Puck Robot

The real world introduces uncertainty within the sensor values because of which, errors within the self localization might occur. The basic behavior in case of similar, disappearing or moved objects is the same and is therefore not tested again.

For the purpose of testing the impact of imprecise sensor values, a testing area of 700 x 700mm containing 3 cylindrical landmarks is defined. Since the e-puck itself has a form of a cylinder with a diameter of 70mm, this again simulates a large room. The vision range is set to be 1000mm to make sure every landmark is visible from every location within the environment. The angular-tolerance is set to 90 °, Similar Object Tolerance to 40% and Similar angular-tolerance to 0%.

With this settings the epuck is positioned on 94 locations (100 - the locations of the landmarks) to perform a self localization. The result of this test is shown in Figure 5.6. In 5.6(a) the actual results of the several locations in form of the recognized area ID is shown. The locations labeled "n" are the landmark locations. To make the accuracy more visible, the calculation within a virtual environment with the same landmarks is displayed in 5.6(b).



(a) (b)

Figure 5.6: Self localization test under real world conditions using e-Puck robot and calculation of the same environment within simulation

Most of the observed uncertainty is the result of the heavy discretization in the test locations. However it can be seen, that not only due to the discreet locations the borders are imprecise but also due to obviously wrong sensor readings. A very good example is the border between area 5 and 7. Taking this location and considering the degree of discretization, the bearing error can be defined as $\pm 5°$. This error results mainly from the imprecise odometry that is used when making a full 360 degree turn to perform a look around. Since this value is used to define the angle in which the landmark is seen, it also causes an error. This error is not constant but can differ from location to location depending on the wheel grip on the ground.
A few very small areas are also not even recognized in the process, but in this case it can be seen mainly as a result of the discretization. On the whole however only 3 of 94 locations can clearly be interpreted to be misidentified due to sensor error which would be 3,2 % of error.

5.2 Dynamic Self Localization and Path Recording

In the second step of evaluation, the agent is exploring the environment by traveling around. During this journey it acquires knowledge about areas as well as the transition spots that connect them. The basic questions to be answered here are how precise the transition informations are, how is the behavior in case of an environmental change and how does the system react to this change. Therefore both, semantic and episodic memory systems are under monitoring as well as the assumption system. Again most of the testing is done in the simulation and the influence of the real world conditions is tested using the e-Puck robot.

Test in Virtual Environment

For evaluations within the virtual test set the agent is told to travel a fixed exploration path which is shown as a white line in Figure 5.7. Since environmental change is not noticed when exploring unknown areas, a part of the path is defined to lead back via known regions. Since this slightly overlapping section also has very close transitions (marked in green), the reaction on environmental change can be evaluated. To get a better impression of the results, the shown images all have a pre calculated area segmentation in the background. This way the results can be compared with the more precise calculation.

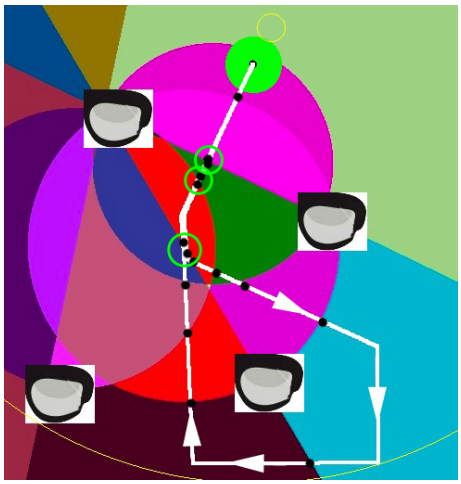


Figure 5.7: Initial path for dynamic self localization test

Static environment: First the area and transition learning as well as the assumption making in a constant environment is tested. Under the present condition that there is no environmental change that would alter the area distribution, the area and connection recognition is almost perfect. This can be seen from comparing the actual area borders with the locations where a transition is recognized. Very small variations still occur within the connection data due to the limited update rate of the vision and orientation system, and little error in the link calculation due to type conversion in the code.

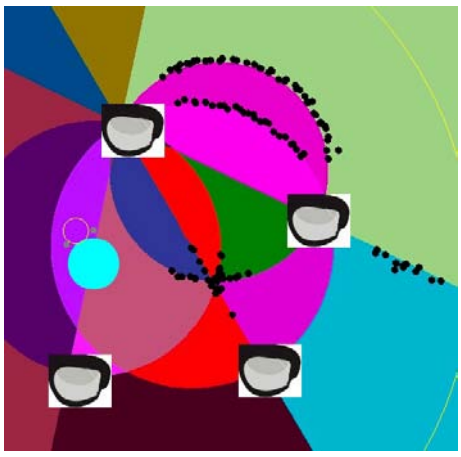
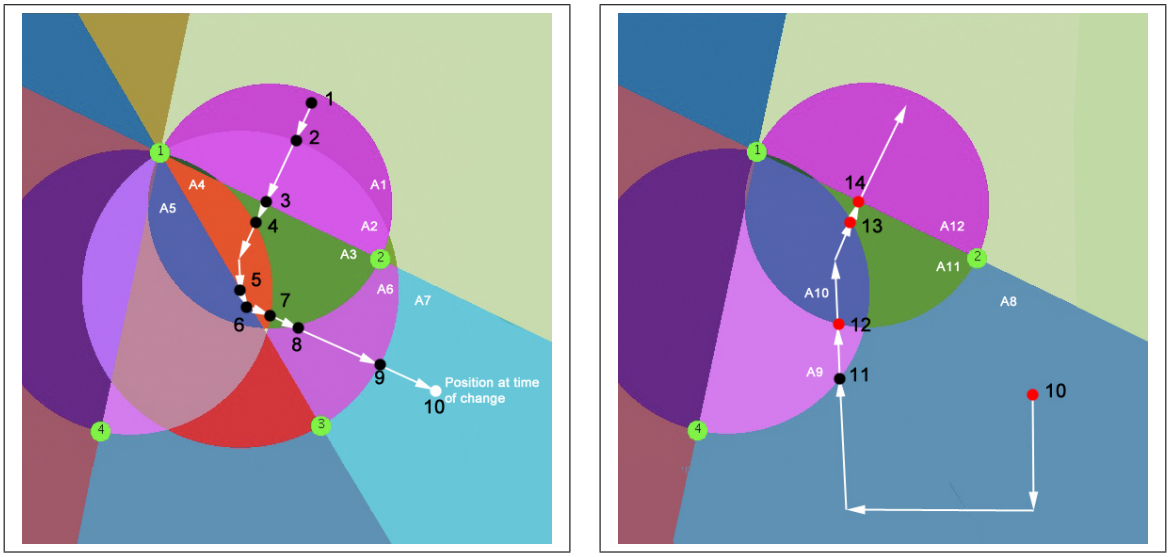


Figure 5.8: Learned transitions at 3 different kinds of borders (curved, straight, star).

Figure 5.7 and longer test runs (see Figure 5.8) show that the soft border (border with a parallel variation) that results from this has an average variation of 10 pixel. Using the

size of the virtual environment in relation to the e-Puck test range (70 x 70 cm), this means a 1,2cm variation. In the test run shown in Figure 5.8, 3 series of transitions are recorded in order to check the outcome of the assumption depending on the nature of the border. Therefore the borders are crossed 130 times on different spots. The result shows that, as expected, no wrong assumption is made on the curved and the straight border. At the star formed one however, at which several areas cross, there is a drastic increase of errors when circling near the crossing point. From the 43 random transitions in this area, 5 cause a false assumption result. This means that the system expects a change to have happened while there was none. Depending on the defined tolerances of the assumption system (see Section 3.1.3) the distance of the star center where this effect occurs, differs. In this test the tolerated angle is set to 30°and the tolerated distance to 15 pixel.

Disappearing Landmark In the following the same parts as before, area learning, transition learning and assumption system, are tested with the situation of an environmental change in form of a disappearing landmark. Therefore the defined path shown in Figure 5.7 is followed and during this travel the change is introduced. It is then observed how the system reacts and if this change is recognized as such.



(a) (b)

Figure 5.9: Agents experience when traveling a path before and after an environmental change

Figure 5.9 shows the result on the defined path. In 5.9(a) the segmentation before the environmental change is shown including the recognized transitions as black spots. In Figure 5.9(a) the traveling path after landmark 3 has disappeared, is shown. The red spots indicate transitions at which an up-to-dateness of an area is modified because the system assumes that an area has changed due to an environmental alteration. Table 5.3 shows the system output and the assumptions. In the figures the step numbers are printed in black and the area ID in white. The lines in the table are marked according to their correctness. It shows, that all the changed areas are recognized as such, and their changed appearance is saved as a new entry. The old ones still remained but have not the same similarity value and are thus not the best result. These values however are still available which is important when following a path. The only false action is done when traveling from area 7 to area 8 in the moment the landmark disappeared. In this case the wrong area is reevaluated. During the later travel in the changed environment, every transition that leading to a previously known area, that now looks different,

Step Nr	System Action	Potential Current Area (Similarity)
1	New Area Defined: Area 1	1 (100%)
2	New Area Defined: Area 2	2 (100%)
3	New Area Defined: Area 3	3 (100%)
4	New Area Defined: Area 4	4 (100%)
5	New Area Defined: Area 5	5 (100%)
6	Traveled to Area 4	4 (100%)
7	Traveled to Area 3	3 (100%)
8	New Area Defined: Area 6	6 (100%)
9	New Area Defined: Area 7	7 (100%)
10	New Area Defined: Area 8	8 (100%),6 (75%),7 (75%)
	Lowering up-to-dateness for area 6; Memory System	
11	New Area Defined: Area 9	9 (100%)
12	New Area Defined: Area 10	10 (100%),5 (75%),4 (75%)
	Lowering up-to-dateness for area 5; Memory System	
13	New Area Defined: Area 11	11 (100%),3 (75%)
	Assuming Path between Areas 3 and 4	
	Lowering up-to-dateness for area 4: Assumption System	
	Lowering up-to-dateness for area 3; Memory System	
14	New Area Defined: Area 12	12 (100%),1 (75%),2 (75%)
	Assuming Path between Areas 2 and 3	
	Lowering up-to-dateness for area 3: Assumption System	
	Lowering up-to-dateness for area 2; Memory System	

Table 5.3: System output during evaluation scenario in Figure 5.9

causes an alteration of the old entries up-to-dateness. Also in the case of the transitions where an assumption process is possible, which are step 13 and 14, the change is recognized correctly. After this short travel, all of the known and now changed areas are identified and modified with a lower up-to-dateness.

Test in e-Puck Robot

Again, the error within the sensor data due to real world conditions is to be tested within the e-Puck robot. The test arena remains the same as in the area recognition test, but this time the robot is traveling around by its own. During its travel the robot learns several transitions which are then evaluated in terms of their accuracy.

Two factors cause an error within the memorized transition precision. For one the look around process and the orientation system update is performed periodically. The length of this period determines how far the robot travels during 2 update steps. At worst this whole distance results in an error. The second problem is the error of bearing and distance sensor. Since both error sources are independent from each other, the result of the sensor error is evaluated separately. Therefore a series of 20 link data calculations are forced at different locations in the arena. The distance from the transition point, calculated using the sensor values, to the real current location is then used as a measurement for accuracy.

The resulting data shows very clearly that there exist regions that cause very high error. In Figure 5.10 a visual representation of the found data is shown. The tested transition locations

are marked as white spots. It can be seen that close to the main and secondary object, which are important for the link calculation, the error gets very high and very inconsistent. To be more precise, due to the error in the distance measurement when located closer than 15cm to the landmark, a transition position error between 13 and 44cm is measured. With greater distance the average error is 2cm with a maximum of 3,7cm in one location. The reason that these error zones are only close to 2 of the landmarks is, that due to the defined priorities, these landmarks always act as main and secondary objects. Since these are the only ones used for the link calculation, only in their close region the distance error has effect on the link data.

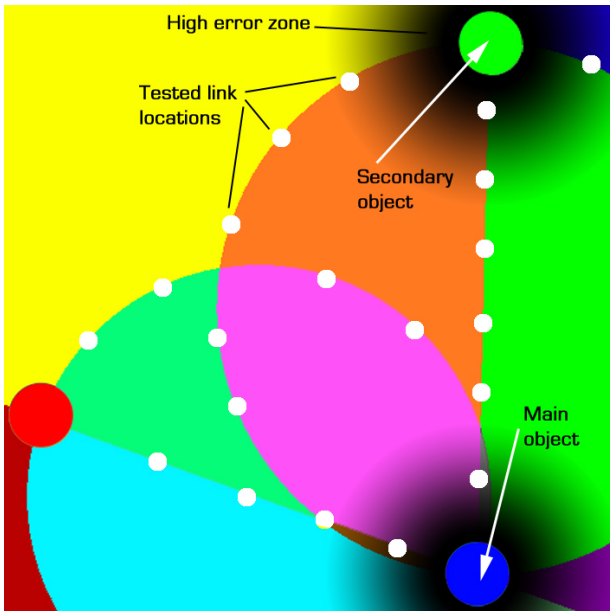


Figure 5.10: Regions with high error potential for link calculation.

5.3 Full Orientation

The full orientation test is the most important part of the evaluation process, because it shows how the system acts as a whole and how qualitative the actual output for the user is. The user in this case is the ARS decision unit.
For testing purpose, the decision unit has the behavior to travel around, to avoid collisions with obstacles and to acquire a topological network map of the environment. The agent is given a natural need for food that grows with time. This need can be satisfied by traveling to an area that contains a food source within a defined range. The area also has to be learned first. When the hunger gets too high the orientation system planes a path to the closest area that contains a food source. This is done by a spontaneous path planing (see Section 3.2.1). The quality of the system is measured by the amount of successful feeding processes and the amount of starvations because the energy source is not reached in time.

Test in Virtual Environment

The environment here is defined similar as in the previous tests within the virtual environment. The difference between the vision range and the range the food source has to be located to be recognized, can be interpreted as a result of the different size. The landmarks are large and visible from every location within the environment. The food source however is small and not visible over large distances. Figure 5.11 shows the environment setup including the energy

source. The red circle indicates the range this object is visible in and the yellow circle the usual vision range of the agent.

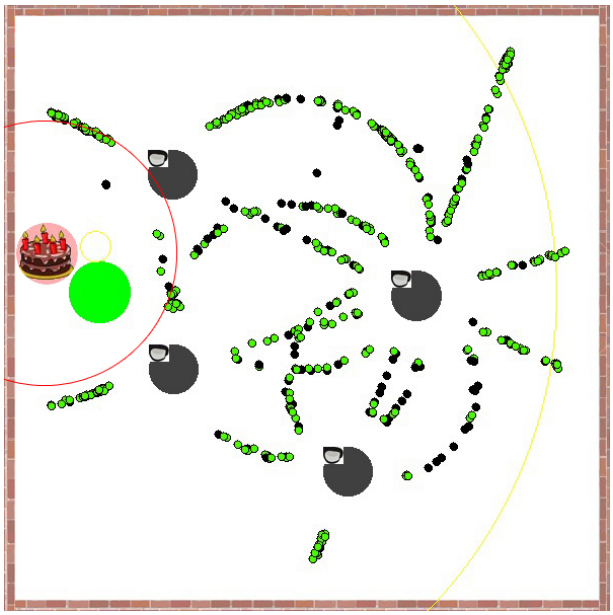


Figure 5.11: Result of a test run with the full system active

The test is performed several times under different conditions. Results are shown in Table 5.4. The most important data values are the amount of times the agent has EATEN and DIED as they indicate the result of the starving situations. These values show how well the system is adapting to the environment and change. Further significant informations are the amount of found areas, the amount of recorded transitions and the actions of the assumption and area evaluation system.

RunNr	*Eaten*	*Died*	*Areas*	*Transitions*	*Assumptions*	$Eval_{Mem}$	$Eval_{Ass}$
1	36	0	14	181	138	0	1
2	57	0	23	634	475	3	18
3	62	0	29	623	460	3	19
4	65	0	21	619	478	2	18
5	68	0	20	609	473	1	22

Table 5.4: Results of full system test

Static environment The system is first tested within a static environment as the standard situation is expected to be. In this case the landmark setup is changed after every trial to eliminate potential influences of the landmark placement. One of the setups is shown in Figure 5.11. Here the transition locations are displayed in black and green, depending on whether the assumption system causes an evaluation action or not. The locations of these shown transitions match the area border distribution. The numeric data of this specific test run is shown in Table 5.4 under Run 5.

The results show clearly that the food search is successful in every case. It means that every time the agent finds its way to the food source before starving and dying. The different amount of areas are a result from the different landmark arrangements. The data also indicates errors the system makes in this test. The evaluation processes that update

an areas up-to-dateness are meant only to occur in case of environmental change. In this case however is still happened without such change. The reasons lie within the defined tolerances. As for the evaluations due to the memory system ($Eval_{mem}$), the similar object tolerance and similar angular-tolerance are responsible. Due to these values two different areas can be interpreted to be alike if they only differ in a degree that is accepted by the tolerance.
These memory system tolerances are also partly responsible for the amount of evaluations due to the assumption system. Because the transitions to or from the areas that are calculated with these tolerances still are within the episodic memory and are used for future assumptions. Since the misinterpretation from the assumption system can happen at every transition but the ones from the memory system only when entering a new area, the occurrence count of the later is correspondingly lower.
From the orientation system point of view it seems as if a change occurred. However these misevaluations seem to have no impact on the system behavior.

Environmental change The second part of the full system test is the situation of an environmental change. Vanishing and appearing landmarks are integrated in the environment. The results show, that depending on which object disappears, the impact differs.
4 different situations are tested, all with the initial environmental layout shown in Figure 5.11. The result data for each run is shown in Table 5.5.

RunNr	*Eaten*	*Died*	*Areas*	*Transitions*	*Assumptions*	$Eval_{Mem}$	$Eval_{Ass}$
1	47	3	30	464	354	8	36
2	42	1	44	585	370	24	58
3	30	2	43	521	330	19	41
4	19	6	46	438	243	14	9

Table 5.5: Results of full system test with environmental change

RUN 1: One disappearing landmark after 1/4 of test time

RUN 2: One disappearing landmark after 1/4 of test time; new landmark appearing after 1/2 of test time

RUN 3: One disappearing landmark after 1/4 of test time; new landmark appearing after 1/2 of test time and vanishing after 3/4 of test time

RUN 4: Main object vanishing after 1/2 of test time

The results show that a vanishing landmark causes a short time disturbance that results in a few deaths of the agent. This confusion due to the missing are borders is however overcome very fast and the EATEN to DIED ratio stays very low. Depending on the amount of change, this ratio varies. The missing main object in run 4 is the most fatal case and causes the system to fail more often even some time after the change occurred. Reason therefore is, that the whole environment has to be learned again.
An other thing the values show very clear, is that much more evaluation processes are being triggered. This time due to a real environmental change.

Test in e-Puck Robot

Again the tests within the e-Puck robot intend to show how the real world conditions affect the functionality of the system. In this case the whole system. For the test, an area is defined previously as a base station. Starting from that area the robot travels through the environment.

After a defined time it is forced to return home to the base station. The quality of the system here is measure in the amount of runs the agent manages to return as planed.

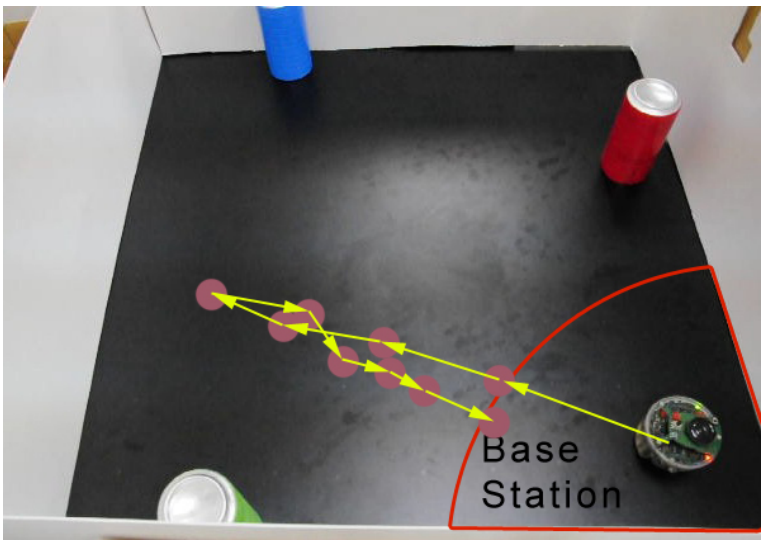


Figure 5.12: Result of a test run with the full system active

The whole test is performed over 10 returning situations. One of those is displayed in Figure 5.12. The robot starts at the base station and travels through the environment. The light red spots indicate the locations where it performs a look around action (see Section 4.5.2) and an orientation update. The yellow arrows describe the traveling path. This particular example shows how the sensor error influences the path. If there is no error the arrows are overlapping. Since the error is in an acceptable range the robot reaches the base station successfully. The results including the amount of areas the agent crossed before returning, can be found in Table 5.6.

Run	*Crossedareas*	*Result*
1	4	*fault*
2	3	*success*
3	4	*success*
4	2	*fault*
5	4	*success*
6	4	*fault*
7	5	*success*
8	4	*fault*
9	5	*fault*
10	3	*fault*

Table 5.6: Results of full system test within e-Puck robot

As it can be seen from the table the sensor error still causes problems when using the system to follow a path. As already shown in Section 5.2, the transition calculation contains grave errors when located in close range to the main or secondary object. This is caused by the error within the distance sensor in shorter distances. Because in some journeys the robot comes very close to the landmarks the transition recording and the traveling to the transition points caused a false path. In this situation a save return to the base station on purpose is not possible. Since all the fault situations result from that error a better distance sensor would be required here.

Chapter 6

Conclusions and Considerations for Future Work

A self localization and orientation system for autonomous agents, inspired by the natural human spatial abilities, is designed in this work. The intention is to create a robust and human like navigation system which can be used for orientation within natural environment. An important point is to tolerate a certain amount of errors in the sensor data as well as limited degree of environmental change. Since the system is designed and implemented within the ARS project (see Chapter 2.5) as an addable module, it is modeled to apply to the basic rules of phrenology (see Section 1.3). Within the ARS project the system allows the agents to find certain goal locations faster to enlarge their life time and shorten their time for task accomplishment. Further on, a real world implementation is done within an e-Puck robot to test the influences of sensor error on the system behavior and the system usability within the real world. During the implementation and testing, considerations for additional enhancements are made which are also explained here.

6.1 Conclusion

The Artificial Recognition System has the goal of creating a computational implementation of a human mind model based on the principles of psychoanalysis. Since the orientation system designed here is going to be a part of this implementation, it is also based on models of the human orientation system. Therefore the topological network map model defined by Byrne (see Section 2.2.1) is used as basic structure. The reason for this selection is, that it forms a widely accepted and most complete theory about the human spatial memory. Due to the usage of memory models described in Solms neuropsychoanalytic approach, it is also in consensus with the ARS project (see Section 2.4).

The used semantic and episodic memory concepts store and retrieve the information of the topological network map. Not only can the retrieved knowledge be used by the agent to localize himself within the environment but also to travel from one location to an other. These are also the 2 main parts of this model, the self localization and the orientation. Since the valuable information for the decision unit is the output of the orientation part and the orientation system is based on the data from the localization system, the whole model is referred to as orientation system.

In his theory Byrne concluded a segmentation of the environment into smaller areas that are defined by the surrounding elements. In the special case of this model, they are defined by the location of landmarks. In Byrne's theory these are no punctual areas but a summation of locations that have a certain alikeness in their environment. The model described here uses the relative bearings between the landmarks to define a certain area. The alikeness is defined by the landmarks in view and a discretization in the relative bearing as described in Section 3.1.1. With these defined alikeness evaluations, natural borders between the areas emerge. The form

and size of the areas formed by these borders depend on the amount and the location of the landmarks as well as the user's editable bearing discretization. By extending these alikeness factors with additional tolerance, a change within the environment can be recognized. An area then is also recognized even though it has changed over time.
With this basic knowledge about the areas, the second part of the topological network map becomes relevant, which is the information about the connections between these border areas. Using that data, one can travel from one area to an other via the link. In the model described in this work, these connections are defined by their location relative to two of the landmarks in sight. For the exact calculation see Section 3.1.1. This however causes the requirement of always having two landmarks available for calculating a link. Moreover, the correct two landmarks need to be existent when using a specific transition again. Problems are arising from this requirement in case the environment is equipped with few landmarks or when one landmark vanishes for some reason. Compared to the human situation where there exist plenty of landmarks to use, like windows, doors, buildings , etc., the problem becomes irrelevant. With continuously recording the area link data, not only a topological network map is created but also the agents' journey is saved. It can be used in future for the reconstruction of a used path and for recognizing environmental change.
Transitions that are used again are recognized by an assumption system and the result of the transition can thus be predicted. A result that does not mach the prediction is interpreted as a change and appropriate reactions are set. Such areas that are known to have changed and transitions that lead to them are then less often used for paths. These paths, or the direction to follow the paths, are the actual output of the whole system and the valuable information the decision unit receives. These paths are planned either in case of strong emotions and drives to lead to locations that satisfy these drives and emotions or consciously due to a request from the decision unit. The only informations however given to the decision unit is the direction in which to go, to follow the path to the planned goal like a food source in case if strong hunger.

Chapter 4 describes the implementation of the main parts of the system for usage within the ARS framework. Primarily the interface to the decision unit is described which is located in a controller unit that manages all the other parts. Further on, the implementation of the area semantic memory unit for managing the storage and retrieval of the encountered areas, as well as the assumption system and the path planning is shown. These units are all realized separately. Besides these modules, extensions to the already available episodic memory system (see [Gru07]) are described briefly. The reason for these modifications is to allow the saving of transition and location data within the recorded events. This way knowledge is available about where certain events happened. Finally the designed orientation system is implemented within a real world e-Puck robot which is described in Section 2.6. Therefore a vision system as well as a low level decision unit are designed which are shown in Section 4.5.1 and 4.5.2.

Finally the model is tested in both environments, in the virtual simulation of the Bubble World (see Section 2.5.3) to examine the basic behavior, and in the e-Puck robot to test the performance under real world conditions. The results of three kinds of tests are shown in Chapter 5. One solely for the self localization to examine the effects that different environmental errors and change have on the segmentation. The second one dedicated to the recording of transitions within the episodic memory system and the correct recognition of alike border crossings by the assumption system. The final one where the whole system is active and used by the ARS decision unit to travel to a close food area according to a planed path caused by an up coming hunger feeling. All test results indicated that the system not only does help the virtual agent to find a food source before starving to death, but also that a certain degree of environmental

change has little to no effect on the outcome. Some changes, like removing or moving many objects or removing or moving important objects, show much more severe consequences. This simply underlines the importance of the selection process of the used landmarks as main objects (see Section 3.1.1 and 3.1.1) based on their basic properties like mass, size or movability. The real world tests within the e-Puck robot are mainly performed to examine the influence of sensor errors on the output. They show very clearly how the influence of sensor errors, even if it is only within a very small range, can influence system performance.

6.2 Future Work

Some consideration for future work are made that are supposed to enhance the performance and reliability of the described system. Also the spatial knowledge exchange is considered. In the following, these enhancements are explained shortly. To make the system applicable under real world conditions, without defined testing environment, a lot of work still has to be done in the field of object recognition. Because correctly identifying the landmarks and picking the most solid objects as landmarks is a very important criteria for the usability. However since this is a different research topic it is not mentioned here as future work.

Short Time Environmental Imprint

Partly merging the theory of topological network maps with the Cognitive map theory mentioned in Section 2.2 would explain several effect in the human spatial abilities. Partly means in this case it could function as a short term exact knowledge of environment. This way the problem of having no continuous omnidirectional vision would be compensated and the orientation process would be much faster. Therefore an additional map based representation of the current environment is saved. If the environment changes the information is still saved for a defined time.

Change Determination

In the current system a change within the environment is tolerated to a certain degree but not exactly identified. It means that the nature of the change is not exactly known. With that knowledge the memory could be completely modified to fit to the new environment. This can be achieved in various ways. An example would be an observed change. When an agent recognizes an area transition that has not been caused due the agent's own movement (see Section 3.1.1). This can be interpreted as a disappearing or moving landmark. The knowledge can than be added to all the other areas and transitions which also depend on this landmark. Another possibility is to analyze all the situations in which the assumption system indicate a change. If several of these situations indicate the missing of a specific landmark, again the memory can be updated.

Knowledge Exchange with other Agents

A very important channel for gaining knowledge about new areas and occurred changes is the communication with other exploring agents. Since every agent travels its own path they all have different knowledge about the environment. Some might even have recognized and identified a change in the environment as explained before as "Change determination". The exchange process could be similar to the human way of communicating a path. Agent A tells agent B about all its known areas by explaining him how it they look like according to the entries in its area semantic memory. Agent B uses the remembering functionality as usual to find out whether it already has knowledge about these locations. After that the same is performed with

the saved transitions.
Another communication strategy would be explaining the path to a location, that for example contains food, to an other agent. Therefore they try to find a common starting area they both know. The explaining agent A could start with the actual goal area with the food, if this is unknown to agent B it tries the surrounding connected areas. This way an area is found both agents know. Starting from this area agent A can plan a path to the goal area and communicate all the intermediate areas as well as the necessary transitions for this path to agent B. Now agent B is able to plan a path from its current location to the goal via the commonly known one. its

Path Planing Algorithm

This has been mentioned several times before. The currently implemented path planning does not have the best performance but on the other hand is consistent with Ho's definition of the autobiographic memory [HDN03]. It has to be examined whether also more sophisticated path planning algorithms like the Djikstra or the D* algorithm also have this property. For an implementation in which the consistency with the human mind is less important, one of the mentioned methods can easily be implemented and would drastically increase the performance.

Extending the Saved Set of Information

The test showed that the system is still vulnerable to some cases on environmental change. To prevent this the information base could be extended. For example by saving the information about all the landmarks in sight when encoding the link location or by adding distance information to the area definition. To stay conform with the human orientation this distance information must be strongly discretized. Therefore the human distance estimation has do be researched more deeply.
An other fact that could be added to the link data is how a transition is passable. In this wok it is assumed that a transition is always passable in both directions. In reality this might not be true in all cases, like at a door that can only be opened from one side.

Bibliography

[AJL02] M. Artac, M. Jogan, and A. Leonardis. Mobile robot localization using an incremental eigenspace model. In *International Conference on Robotics and Automation, Proceedings*, volume 1, pages 1025 – 1030. ICRA, May 2002 2002.

[Bad97] A. Baddeley. *Human Memory: Theory and Practice*. Psychology Press, 1997.

[Bug97] G. Bugmann. *A Connectionist Approach to Spatial Memory and Planning*, chapter 5. Springer-Verlag New York, LLC, 1997.

[BW94a] I. A. Bachelder and A. M. Waxman. Mobile robot visual mapping and localization: A view-based neurocomputational architecture that emulates hippocampal place learning. *Neural Networks*, 7:1083 – 1099, 1994.

[BW94b] I. A. Bachelder and A. M. Waxman. A neural system for qualitative mapping and navigation in visual environments. In *From Perception to Action Conference,*, pages 266 – 277, September 1994.

[BW95] I. A. Bachelder and A. M. Waxman. A view-based neurocomputational system for relational map-making and navigation in visual environments. *Robotics and Autonomous Systems*, 16:267 – 289, 1995.

[Byr79] R.W. Byrne. Memory for urban geography. *Quaterly Journal of Experimental Psychology*, 31:147 – 154, 1979.

[CLRS09] T.H. Cormen, C.E. Leiserson, R. L. Rivest, and C. Stein. *Introduction to Algorithms*. MIT Press, 3 edition, 2009.

[CMM06] Y. Cheng, M. Maimone, and L. Matthies. Visual odometry on the mars exploration rovers. *IEEE Robotics & Autmation Magazine*, pages 54 – 62, June 2006.

[DDBT99] F. Dellaert, D.F., W. Burgard, and S. Thrun. Monte carlo localization for mobile robots. In *IEEE International Conference on Robotics and Automation*. IEEE, May 1999.

[DFZB09] D. Dietrich, G. Fodor, G. Zucker, and D. Bruckner. *Simulating the Mind - A Technical Neuropsychoanalytical Approach*. Springer-Verlag Wien, 1 edition, 2009.

[DGLV08] T. Deutsch, A. Gruber, R. Lang, and R. Velik. Episodic memory for autonomous agents. In *Conference on Human System Interactions*, pages 621 – 626, May 2008.

[Dij59] E. W. Dijkstra. A note on two problems in connexion with graphs. *Numerische Mathematik*, 1:269 – 271, 1959.

[DLP+06] T. Deutsch, R. Lang, G. Pratl, E. Brainin, and S. Teicher. Applying psychoanalytic and neuro-scientific models to automation. In *The 2nd IET International Conference on Intel ligent Environments*, pages 111 – 118, 2006.

[DZL07] T. Deutsch, H. Zeilinger, and R. Lang. Simulation results for the ars-pa model. In *5th IEEE International Conference on Industrial Informatics*, volume 2, pages 995 – 1000. IEEE, 2007.

[FAW07] S. Fitting, G. L. Allen, and D. H. Wedell. Remembering places in space: A human analog study of the morris water maze. *Spatial Cognition*, 5:59 – 75, 2007.

[FG97] S. Franklin and A. Graesser. Is it an agent, or just a program?: A taxonomy for autonomous agents. In *Proceedings of the Workshop on Intelligent Agents III, Agent Theories, Architectures, and Languages*, pages 21 – 35. ECAI, 1997.

[FH94] E.L. Fergusin and M. Hegerty. Properties of cognitive maps constructed from texts. *Memory and Cognition*, 22:455 – 473, 1994.

[Flo05] S. Florczyk. *Robot Vision*. Wiley-VCH Verlag GMBH & CO, 2005.

[FSMB98] M. Franz, B. Schölkopf, H. A. Mallot, and H. H. Bülthoff. Learning view graphs for robotic navigation. *Autonomous Robots*, 5:111 – 125, 1998.

[GL06] S. Sam Ge and F. L. Lewis. *Autonomous Mobile Robots*. CRC Press Taylor and Francis Group, 2006.

[Gru07] A. Gruber. Neuro-psychoanalytically inspired episodic memory for autonomous agents. Master's thesis, University of Technology, Vienna, May 2007.

[HDN03] Wan Ching Ho, K. Dautenhahn, and C. L. Nehaniv. Comparing different control architectures for autobiographic agents in static virtual environments. In *Intelligent virtual agents. International workshop No4*, volume 2792, pages 182 – 191, September 2003.

[Jef07] K. J. Jeffery. Self-localization and the entorhinal–hippocampal system. *Current Opinion in Neurobiology*, 17:684 – 691, 2007.

[Kal60] E. Kalman. A new approach to linear filtering and prediction problems. *Transactions of the ASME–Journal of Basic Engineering*, 82:35 – 45, 1960.

[LL92] A. Lazanas and J.C. Latombe. Landmark-based robot navigation. In *Proceedings of the Tenth National Conference on AI*, 1992.

[MBR+09] F. Mondada, M. Bonani, X. Raemy, J. Pugh, C. Cianci, A. Klaptocz, S. Magnenat, J.C. Zufferey, D. Floreano, and A. Martinoli. The e-puck, a robot designed for education in engineering. In *Proceedings of the 9th Conference on Autonomous Robot Systems and Competitions*, volume 1, pages 59 – 65, 2009.

[MC82] I. Moar and L.R. Carleton. Memory fo routes. *Quarterly Journal of Experimental Psychology*, 34A:381 – 394, 1982.

[MTBF03] R. Moratz, T. Tenbrink, J. Bateman, and K. Fischer. Spatial knowledge representation for human-robot interaction. In *Spatial Cognition: Routes and Navigation, Human Memory and Learning, Spatial Representation and Spatial Learning*, volume 3, 2003.

[NS00] M. Noto and H. Sato. A method for the shortest path search by extended dijkstra algorithm. In *IEEE International Conference on Systems, Man, and Cybernetics*, volume 3, pages 2316 – 2320. IEEE, 2000.

[OD71] J. O'Keefe and J. Dostrovsky. The hippocampus as a spatial map: preliminary evidence from unit activity in the freely-moving rat. *Brain Res.*, 34:171 – 175, 1971.

[Pfe99] R. Pfeifer. *Understanding Intelligence*. MIT Press, 1999.

[RF96] P.N. Rao and O. Fuentes. Learning navigational behavior using a predictive sparse distributed memory. In *The Fourth International Conference on Simulation and Adaptive Behavior*. MIT Press, 1996.

[RH96] M. Recce and K. D. Harris. Memory for places; a navigation model in support o marr's theory of hippocampal function. *Hippocampus*, 6:735 – 748, 1996.

[RLVF06] C. Roesner, B. Lorenz, K. Vock, and G. Fodor. Emotional behavior arbitration for automation and robotic systems. In *Proceedings of 2006 IEEE International Conference of Industrial Informatics*, pages 423 – 428, 2006.

[RS05] I. Rañó and T. Smithers. Obstacle avoidance through braitenberg's aggression behavior and motor fusion. In *Proceedings of the 2nd European Conference on Mobile Robots*, pages 98 – 103, 2005.

[SN04] R. Siegwart and I. R. Nourbakhsh. *Introduction to Autonomous Mobile Robots*. The MIT Press, 2004.

[SON96] K. T. Simsarian, T. J. Olson, and N. Nandhakumar. View-invariant regions and mobile robot self-localization. *IEEE Transactions on Robotics and Automation and Automation*, 12(5):810 – 816, Octobe 1996.

[Spe50] K.W. Spence. Cognitive versus stimulus-response theories o learning. *Psychological Review*, 57:159 – 172, 1950.

[ST94] D. L. Schacter and E. Tulving. *Memory Systems*. The MIT Press, 1994.

[ST02] M. Solms and O. Turnbull. *The Brain and the Inner World*. Karnac/Other Press, Cathy Miller Foreign Rights Agency, London, England,, 2002.

[Ste94] A. Stentz. Optimal and efficient path planning for partially-known environments. In *In Proceedings of the IEEE International Conference on Robotics and Automation*, pages 3310 – 3317. IEEE, 1994.

[Ste95] A. Stentz. The focussed d* algorithm for real-time replanning. In *In Proceedings of the International Joint Conference on Artificial Intelligence*, pages 1652 – 1659. IEEE, 1995.

[STG] N. H. Sleumer and N. Tschichold-Gürman. Exact cell decomposition of arrangements using path planing robotics.

[Tau07] J. S. Taube. The head direction signal: origins and sensory motor integration. *Annual Review on Neuroscience*, 30:181 – 207, 2007.

[Tol48] E.C. Tolman. Cognitive maps in rats and men. *Psychological Review*, 55:189–208, 1948.

[TSSE97] H. Tanila, P. Sipila, M. Shapiro, and H. Eichenbaum. Brain aging: impaired coding of novel environmental cues. *Neuroscience*, 17:5167 – 5174, 1997.

[Tul83] E. Tulving. *Elements of Episodic Memory*. Oxford: Clarendon Press, 1983.

[VB08] R. Velik and D. Bruckner. Neuro-symbolic networks: introduction to a new information processing principle. In *6th IEEE International Conference on Industrial Informatics, 2008. INDIN*, pages 1042 – 1047. IEEE, July 2008.

[Vel09] R. Velik. *A Bionic Model for Human-like Maschine Perception*. PhD thesis, University of Technology, Vienna, 2009.

[VLBD08] R. Velik, R. Lang, D. Bruckner, and T. Deutsch. Emulating the perceptual system of the brain for the purpose of sensor fusion. In *IEEE conference on Human System Interaction*, pages 657 – 662. IEEE, 2008.

[Web95] B. Webb. Using robots to model animals: a cricket test. *Robotics and Autonomous Systems*, 16:117 – 134, 1995.

[YJYQ03] W. Yimin, X. Jianmin, H. Yucong, and Y. Qinghong. A shortest path algorithm based on hierarchical graph model. *Intelligent Transportation Systems*, 2:1511 – 1514, October 2003.

[YSSB98a] A. Yahja, A. Stentz, S. Singh, and B. L. Brumitt. Framed-quadtree path planning for mobile robots operating in sparse environments. In *IEEE International Conference on Robotics and Automation*, volume 1, pages 650 – 655. IEEE, May 1998.

[YSSB98b] A. Yahja, A. Stentz, S. Singh, and B.L. Brumitt. Framed-quadtree path planning for mobile robots operating in sparse environments. In *IEEE Conference on Robotics and Automation*. IEEE, May 1998.

[ZDML08] H. Zeilinger, T. Deutsch, B. Müller, and R. Lang. Bionic inspired decision making unit model for autonomous agents. In *IEEE International Conference on Computational Cybernetics*, pages 259 – 264, November 2008.

Internet References

[ecl09] eclipse.org, June 2009. http://www.eclipse.org.

[GPS09] WIKIPEDIA GPS, June 2009. http://en.wikipedia.org/wiki/Global_Positionings_System.

[MAS09] MASON, June 2009. http://cs.gmu.edu/ eclab/projects/mason/.

[Phr09] WIKIPEDIA Phrenology, June 2009. http://en.wikipedia.org/wiki/Phrenology.

[Sex09] WIKIPEDIA Sextant, June 2009. http://en.wikipedia.org/wiki/Sextant.