

ROBOT POSITIONING BY SUPERVISED AND UNSUPERVISED ODOMETRY CORRECTION

THÈSE NO 1858 (1998)

PRÉSENTÉE AU DÉPARTEMENT D'INFORMATIQUE

ÉCOLE POLYTECHNIQUE FÉDÉRALE DE LAUSANNE

POUR L'OBTENTION DU GRADE DE DOCTEUR ÈS SCIENCES TECHNIQUE

PAR

Philip MÄCHLER

Ingénieur électrotechnique diplômé EPFZ
originaire de Küsnacht (ZH)

acceptée sur proposition du jury:

Prof. J.-D. Nicoud, EPFL (directeur de thèse)

Prof. R. Siegwart, EPFL, Switzerland

Prof. J. Crowley, I.N.P. Grenoble, France

Dr Marek Piasecki, Wroclaw UNI, Poland

Lausanne, EPFL

1998

Abstract

The aim of this thesis is to perform robot positioning, based on an odometry which is continuously corrected by different landmark detection systems demanding as less modifications as possible for the environment. Two independent correction systems (a supervised and an unsupervised) were implemented into two different experiences which represent the subject of this thesis.

The supervised experiment uses grid lines painted on the floor which are detected by a single light sensor underneath the robot which cannot distinguish between horizontal and vertical lines. The robot knows the geometry of the grid lines and its estimated position which is calculated by odometry. A new position probability model calculates the assumed robot position and transforms this single sensor information into a reliable position and orientation indication of the robot. The intended trajectory is slightly modified in order to optimize the correction algorithm by guiding the robot more efficiently over close grid lines. The theory was implemented and tested on a real Khepera robot. Investigation and modeling of the odometry error is the main subject of this first experiment.

The second experiment demonstrates continuous odometry correction by an unsupervised correction system. Different kind of unsupervised neural networks classify the robot's rough sensor signals. A statistical algorithm extracts "meaningful" classes from the generated pool of classes and uses these as reference points for odometry correction. The robot position will be calibrated to these reference points if the robot detects the same "meaningful" sensor information which generated earlier the corresponding reference point. What is going to be used as a reference point is initially unknown, and no extended pre-processing of the sensor signals is done. The localization method allows a robot to correct its position calculated by odometry without any prior information about its environment and its sensor configuration. This localization approach takes full advantage of the sensor abilities, because no model unsuitable for the sensor configuration was imposed.

In a higher level, sequences of reference points are grouped to *places* in order to establish a more reliable indication for a certain region in the environment. In opposite to reference points, places do not indicate an exact robot position, but more a fuzzy region which can be still detected even if the robot trajectory or the environment is slightly influenced by noise or other circumstances. Place recognition is done by Markov chain detecting reference point transitions. The result is a topological map of the environment representing these places and the common transitions between.

Résumé

On étudie dans cette thèse la manière dont un robot peut se localiser en se basant sur les résultats de son odométrie corrigés en continu par différents systèmes de détection de repères naturels. On y propose deux systèmes de correction distincts, l'un avec supervision, l'autre sans supervision. Ils font chacun l'objet d'une série d'expériences.

Dans les expériences supervisées, un réseau de lignes orthogonales est peint sur le sol. Un capteur de lumière placé sous le robot réagit à la présence des lignes, mais ne peut ni les identifier, ni distinguer leur orientation. Le robot sait toutefois que les lignes sont orthogonales et connaît leur espacement. Il connaît également sa position approximative par odométrie. A l'aide d'un algorithme original, il calcule la zone de ses emplacements possibles en y associant une probabilité. L'information rudimentaire du capteur unique lui permet alors de déterminer sa position et son orientation avec une grande précision. Lorsque, dans ses déplacements, le robot juge que sa marge d'incertitude devient trop grande, il modifie légèrement sa trajectoire pour franchir une ligne peinte selon un meilleur angle et recalculer ainsi sa position exacte. Cette méthode a été testée sur un robot Khepera. L'étude et la modélisation des erreurs d'odométrie sont au centre de ces premières expériences.

La seconde série d'expériences présente un système non supervisé de correction en continu de l'odométrie. Plusieurs réseaux neuromorphiques non supervisés de types différents participent à la classification des signaux bruts provenant des capteurs du robot. Un algorithme statistique extrait les classes significatives de l'ensemble des classes produites et les utilise comme points de référence pour la correction de l'odométrie. Lorsque le robot reconnaît dans les signaux provenant de ses capteurs une information significative déjà classifiée, il l'utilise pour recalibrer son odométrie. On ignore à l'avance ce qui, dans l'environnement, sera utilisée comme repère et, pour laisser le plus d'ouverture possible au robot, on limite rigoureusement le pré-traitement des signaux. Cette méthode permet donc à un robot de corriger ses erreurs d'odométrie sans rien savoir initialement de son environnement, ni de la nature et de la configuration de ses propres capteurs. En renonçant à imposer aux capteurs une certaine vision de leur environnement, elle leur permet de tirer au mieux profit de leurs caractéristiques.

Après l'identification de repères, le calcul se poursuit par la constitution d'"emplacements" définis par une séquence de repères. On obtient ainsi une caractérisation encore plus fiable de certaines régions de l'environnement. A l'inverse des repères, les emplacements ne permettent pas de préciser la position exacte du robot, mais plutôt de les situer dans une région circonscrite de manière floue mais clairement identifiable même lorsque la trajectoire du robots ou certains traits de l'environnement ont été légèrement modifiés par des perturbations de diverses natures. La reconnaissance des emplacements est assurée par une chaîne de Markov détectant les points de transition. Il en résulte une carte topologique de l'environnement représentant les emplacements et les transitions les reliant.



für meine Eltern

Table of contents

Abstract	iii
Résumé	iv
Table of contents	vii
Table of figures	xi
1 Introduction	1
1.1 Thesis inspiration and justification	2
1.2 Organization of this thesis.....	2
1.3 Sensor positioning systems in robotics	3
1.4 Data sensor fusion.....	4
1.5 The importance of odometry in robot positioning	6
1.6 The problem of modularized data processing	7
1.7 Related Work in the field of model-less learning	9
2 Supervised Passive Positioning System (SPPS)	11
2.1 Overview	12
2.2 Introduction to odometry correction	13
2.2.1 Properties of odometry errors	13
2.2.2 Combining of systematic and non-systematic odometry error	14
2.2.3 One-dimensional correction.....	14
2.2.4 Two-dimensional correction.....	16
2.2.5 The traditional ellipse approach to distinguish x and y correction	16
2.3 The PPD model	17
2.3.1 Introduction.....	17
2.3.2 Geometrical calculation of the PPD model	18
2.3.3 Calculation and display of the PPD shape	19
2.3.4 Differences between the circular and the PPD model	22
2.3.5 Deformation of the PPD by curves and corners	22
2.3.6 Examples.....	23
2.3.7 Conclusion	24
2.4 Statement of the problems.....	25
2.4.1 Angular drift, caused by inaccuracy robot direction	25
2.4.2 Information distribution of a grid	25
2.5 Strategy for PPD, robot position and direction correction.....	26
2.5.1 Passive correction, using the PPD	26
2.5.2 Correction of the position	26
2.5.3 Correction of the angular error	27
2.6 Integration of a path planner “grid fitter”	28
2.6.1 Including the grid fitter into a navigation system	29
2.6.2 Grid fitter implemented by potential fields	31
2.6.3 Other tasks of the grid fitter.....	32
2.6.4 Practical tests	33
2.7 Conclusion	34

3	Unsupervised Passive Positioning System (UPPS)	35
3.1	Unsupervised classification approaches	36
3.1.1	Statistical approach	36
3.1.2	Machine learning approach	36
3.1.3	Neural network approach	36
3.2	Experimental Set-up	38
3.2.1	Aim	38
3.2.2	Experience set-up	39
3.2.3	Algorithm overview	40
3.3	Normalization	41
3.3.1	Distance and ambient light sensors	41
3.3.2	Camera preprocessing	41
3.4	Classification	45
3.4.1	The Growing K-means Algorithm	46
3.4.2	Adaptive Resonance Theory (ART)	48
3.4.3	Simplified Fuzzy ART	50
3.5	Discussion of unsupervised classifiers	52
3.5.1	Significant versus entire number of classes	52
3.5.2	Quality versus quantity of significant classes	53
3.5.3	Class distribution in the input space	53
3.5.4	Representing the environment by classes	54
3.6	Classes become landmarks	55
3.7	Landmark comparison by Levenshtein Distance	56
3.7.1	Introduction	56
3.7.2	Recursive mathematical definition	56
3.7.3	Flat implementation of the LD	58
3.7.4	Code example for WLD	58
3.7.5	Graphical example	59
3.7.6	Calculation of substitution cost	59
3.7.7	Recognizing identical landmarks by WLD	60
3.7.8	Estimated robot position supports landmark recognition	61
3.8	Constraining robot movements	62
3.9	Introduction to concepts	63
3.9.1	Concept quality depending on frequency	64
3.9.2	Concept quality depending on the sensor ability	64
3.10	Landmarks become places	65
3.10.1	Recognizing places from the landmark stream	65
3.11	Place identification by Markov Chain	66
3.11.1	Concept of probability	66
3.11.2	A simple Markov model	67
3.11.3	Implementation of Place recognition by Markov Chain	68
3.12	What's the difference between Levenshtein and Markov?	70
3.13	Places become a map	71
3.14	Algorithm controlling	72
3.14.1	Interconnections between the different levels	73
3.14.2	Learning phases	74

3.15 Experimental results.....	75
3.15.1 Landmark distribution	75
3.15.2 Odometry error correction	76
3.15.3 Place production	76
3.15.4 Structured environment	77
4 Discussion and conclusion.....	79
4.1 Future work.....	81
5 Appendix	83
5.1 Different ways to calculate the odometry	84
5.1.1 Odometry calculation describing the direction of the robot by an angle.....	84
5.1.2 Odometry calculation describing the direction of the robot by a vector	85
5.2 Levenshtein simulators on the WEB.....	86
5.3 PPD simulation with Mathematica	87
5.4 Simplified Fuzzy ARTMAP	89
6 References.....	91
7 Remerciements aux dieux de l'Olympe	97
8 Publications	99
9 Curriculum Vitae.....	101

Table of figures

Fig. 1-1:	Robot PEMEX for antipersonnel mine detection.	2
Fig. 1-2:	Sensor fusion of related information into similar spatial place of the pit tectum	4
Fig. 1-3:	Foraging and return of a desert ant “Cataglyphis fortis” [Wehner R., 1992].	6
Fig. 1-4:	Simplified landmark recognition by several processing modules	7
Fig. 1-5:	Recognition of useful part in signals, independently of the sensor type	8
Fig. 1-6:	Rectangular robot environment with a significant exception.	8
Fig. 2-1:	Khepera performs odometry correction by detecting grid lines on the floor.	11
Fig. 2-2:	Combination of different positioning systems	12
Fig. 2-3:	Folding of the non-systematic and systematic gaussian error distribution	14
Fig. 2-4:	Example of the fundamental technique to correct odometry error.	15
Fig. 2-5:	Example: The closest grid line is not always the best	16
Fig. 2-6:	Passing grid lines change the shape of the ellipses	16
Fig. 2-7:	Minor wheel speed difference causes a strong change of the robot direction	17
Fig. 2-8:	Derivation of a PPD model	18
Fig. 2-9:	Gaussian distribution of the wheel speed.	20
Fig. 2-10:	PPD shape (right side). Path length 3m; straight ahead; wheel slip 1%	21
Fig. 2-11:	Difference between the PPD and traditional circular approach.	22
Fig. 2-12:	Example 1: PPD behavior in a curve	23
Fig. 2-13:	Example 2: PPD behavior after a turn back	23
Fig. 2-14:	Example 3: PPD decrease after a turn back	24
Fig. 2-15:	Spreading out and concentrating of the PPD	24
Fig. 2-16:	Wrong correction because of confused grid lines	25
Fig. 2-17:	Correction of the assumed position by matching the PPD with the grid	26
Fig. 2-18:	Each point of the PPD indicates the angular drift	27
Fig. 2-19:	The modified path (dotted line) leads to many more V-H transitions	28
Fig. 2-20:	Zone events, caused by the intended and corrected way	29
Fig. 2-21:	The grid fitter can easily be inserted into a standard path planner.	30
Fig. 2-22:	Potential fields guide the robot over V and H zones	31
Fig. 2-23:	Diagonal grid line crossing improves the recognition of angular errors	32
Fig. 2-24:	Angle between PPD and grid line is important (not PPD and robot path).	32
Fig. 2-25:	Experiment without and with odometry correction	33
Fig. 3-1:	Robot exploring an unknown environment	35
Fig. 3-2:	Three different representations of an artificial neuron.	37
Fig. 3-3:	Standard environment offers several stimuli	38
Fig. 3-4:	Equipment of the extended Khepera robot.	39
Fig. 3-5:	Sensor signal data processing and recognition of significant features.	40
Fig. 3-6:	Calculation of light intensity, first approximation	41
Fig. 3-7:	Two different images preprocessing to improve classification	42
Fig. 3-8:	Screen-shot of the camera image normalization	43
Fig. 3-9:	Effect of camera processing	44

Fig. 3-10: 2-D robot sensor space defined by a distance sensor and a compass.	45
Fig. 3-11: Network structure of the K-means algorithm.	47
Fig. 3-12: General ART architecture	48
Fig. 3-13: Sketch of a Fuzzy ART network	50
Fig. 3-14: Almost linear relationship between significant and used classes.	52
Fig. 3-15: The reactivation of significant classes is better for small neural networks	53
Fig. 3-16: Different cluster shapes for a 2D input space.	54
Fig. 3-17: Different environment representation by classes	54
Fig. 3-18: Combining three class streams into a landmark.	55
Fig. 3-19: A landmark is a significant class combined with the adjacent classes.	55
Fig. 3-20: Combination tree to compare two strings with each two characters	57
Fig. 3-21: Landmark comparison by analyzing all class streams	60
Fig. 3-22: Similarities of landmarks depend on Levenshtein distance and position error	61
Fig. 3-23: Robot path attracted by walls and contours	62
Fig. 3-24: Object's frequency influences the concept's quality	64
Fig. 3-25: Different concepts for the same object depending on the sensor ability	64
Fig. 3-26: Accumulations of landmarks become places	65
Fig. 3-27: Definition of First Order Markov Chain	66
Fig. 3-28: Transition probability matrix [T] of English inhabitants	67
Fig. 3-29: Typical distribution of the most important places	71
Fig. 3-30: Distorted representation of the environment	71
Fig. 3-31: Complete algorithm divided into levels and temporal phases	72
Fig. 3-32: Transition of the learning phases	74
Fig. 3-33: Robot environment with all created landmarks from all classes	75
Fig. 3-34: Stabilized error due to landmark synchronization	76
Fig. 3-35: Place evolution in time.	77
Fig. 3-36: Structured environment leads to little and less reliable class production.	77
Fig. 3-37: Landmarks synchronization is less stable and fails after a certain time.	78
Fig. 4-1: Some significant situations can only be recognized by sensor correlation	80
Fig. 5-1: Rough overview of the traditional approach for odometry calculation	84
Fig. 5-2: Displacement calculation based on the normalized direction vector of the robot	85

1 Introduction

Watching a running robot searching for its trajectory and avoiding obstacles at the same time is both entertaining and intriguing for human observers. Self-identification is tempting because the robot is executing a very familiar task. Of course, the robot cannot withstand a comparison with human navigation strategy. The differences are so extensive that it would not make sense to list them in this chapter.

The observer is often fascinated to see how a robot is at ease to perform a task considered difficult or even impossible for a human being (specially concerning manipulation speed and precision). On the other hand, the same observer is sometimes astonished to discover the clumsiness of autonomous robots moving in an apparently simple environment. Robot's skills do not match human ones, they sometimes even seem to be quite the opposite.

One of the reasons humans are astonished by robotical behavior is because they tend to judge them according to human *clichés*. The robot's behavior is considered "human" if its reaction matches the imposed human model otherwise its behavior is considered "awkward". As a result, scientists try to adapt the robot's behavior to human or animal. Different techniques were developed to analyze and to model human and animal behavior which led to a wide spectrum of models. One problem can be put down to the fact that robots do not relate to their environment the way humans or animal do: their sensors, their effectors and everything else in between have completely different characteristics. For instance, robots can easily perceive infrared sources, while human cannot without technical help, and humans can easily recognize a human face while robots cannot. Thus, even if robots and humans work in the same environment, the way they would perceive it is different. Why should a robot recognize a chair, if it can't sit on it? Why should a human care about magnetic field if it doesn't harm him at all?

Therefore the robot's and human's perception of the world is completely different and will probably never really be understood by humans. Hence the question is, are we actually able to create a model for a robot (agent) whose abilities we cannot really understand? The last sentence seems to enter the field of psychology, but the problem is also well known in that of robotics. Several scientists developed systems which are able to develop it's own model or at least which are able to select the best model of a set [Kuipers B., 1997] [Schmidhuber J., 1997].

A possible solution to this problem is letting the robots agent build its own model. This requires the ability to recognize the coherence and the context of the environment and not only to adjust some parameters of a model. It is obvious that to develop this idea will be very difficult and will hardly lead to an efficient result. Some initial models are still needed as a base to build on. But what is the limit? What kind of flexibility can we expect from systems that classify and conceptualize for themselves their own world?

1.1 Thesis inspiration and justification

The principal idea of this thesis was born during my first project in the lab. Prof. Nicoud developed a concept for a robot, able to detect anti-personnel mines and designed the robot PEMEX (PErsonnel Mine EXplorer) [Nicoud et al., 1995a] [Nicoud et al., 1995b]. My task was to realize the robot and to implement a navigation algorithm for the Pemex. The second task was much more difficult than expected. Almost every navigation algorithm meeting the imposed specifications depended on some special environment conditions or were based on artificial reference points which were difficult to put up due to the mine danger. However, as a human observer, the environment offered a great variety of reference points like trees, trails, horizon reliefs, surface conditions, etc. But there was no model and also no sensor fitting such reference points. A robot, placed in a natural environment, felt himself in another world and was completely lost. The problem was not only related to the inadequate sensors but also to the absence of a suitable model for information processing. The often confusing and noisy sensor signal made it difficult to develop an adequate model which necessitated good sensor quality. Observing the sometimes astonishing raw sensor values raised the following questions in my mind:

- Why not use the motor energy consumption to detect muddy ground or slope?
 - Why not use noisy ultrasonic information caused by a bush of grass?
 - Why not use a short drift in the compass every time the robot passed close the computer monitor?
- These questions were the “kickoff” to investigate a system which could identify and use any kind of sensor signals as a reference point, regardless, as far as possible, of any kind of model.



Fig. 1-1: Robot PEMEX for antipersonnel mine detection

1.2 Organization of this thesis

Odometry correction will be first tackled in chapter 2 in which an experiment shows the effect of odometry error and an original way to correct it by detecting grid lines on the floor. This experiment represents a *supervised* odometry correction because the dimension of the grid line is known. However, the supervised correction method does not use the ideas discussed above.

The second experiment in chapter 3 presents an approach for robot localization which is also based on odometry but corrected by an *unsupervised* system. Because the problematic of odometry error was already discussed in the previous section, this chapter treats exclusively the problem of unsupervised (or model-less) correction. The presented model-less correction method use neither knowledge about the environment nor about its own sensor configuration. This reduces the risk of *loss of information* by model based preprocessing, which are always created and therefore influenced by human models and therefore probably doesn't fit very well the robot's internal world. The problem is explained more detailed in section 1.6 on page 7.

1.3 Sensor positioning systems in robotics

Recent years have witnessed a tremendous development in robotical electronics. Most progress was based on the commercial availability of interesting new components. For instance, small and efficient electric motors and controllers were developed for the car industry. In twenty years, microcomputer power grew from simple Z80 processors with 64kByte address space to Pentium based system with a thousand times larger address space. But in the field of robot positioning sensors, nothing very spectacular was achieved. Positioning remains the weak spot of robot systems. Table 1-1 gives an overview of the most frequently used robotical positioning sensors. A very good survey of almost every kind of positioning sensors can be found in the Book “Sensors for Mobile Robots” [Everett H., 1995]:

Pos. system	Short description	Advantage	Drawbacks
Odometry	Position determination by integrating wheel speed information	• cheap	• distance drift
Inertial navigation	Sensing minor accelerations in all directional axes and integrating over time to derive velocity and position	• no ground contact needed	• time drift • expensive
Compass	Measures direction heading based on Earth's magnetic field	• cheap • no time drift	• sensitive to magnetic anomalies • limited accuracy
Gyroscope	Direction heading based on the inertial properties of a rapidly spinning motor or other techniques like optical gyroscope	• insensitive to magnetic anomalies	• time drift • expensive
Ultrasonic distance	Time of flight measurement for an ultrasonic chirp (pulse) to travel to a reflective object	• cheap	• no focusing • slow • limited range • artifact by echoes
Light/Laser triangulation	Reflection angle measurement of an emitted light beam	• cheap • fast	• limited range
Laser time of flight	Time of flight or phase shift measurement of an emitted light pulse	• fast	• expensive
Landmark recognition	Distance and/or angle measurement between the robot and an artificial landmark	• absolute position	• external set up
Stereo vision	Distance measurement by object recognition of two staggered images	• no external set up	• complex algorithm • not reliable
GPS	Global positioning system based on satellites	• absolute position	• free sky view

Table 1-1: Mostly used sensor system for robot positioning

Most other types of positioning systems can be reduced to one or several positioning system presented in table 1-1. However, generally no one of the enumerated systems can be used as a single positioning system because they are not enough reliable or are too expensive. A solution is to fuse multiple sensor information to obtain usable compromise solutions.

1.4 Data sensor fusion

Multisensor data fusion is mainly used in military applications. The Department of Defense (DoD) in the United States of America issues almost every definition in this field. Data fusion techniques combine data from multiple sensors to achieve improved accuracy and more specific inferences than could be achieved by the use of a single sensor alone. The Joint Directors of Laboratories Data Fusion Group, established in 1986 [Hall et al., 1990], has been active in standardizing the terminology used in data fusion. They have established the following definition of multisensor data fusion:

“A continuous process dealing with association, correlation, and combination of data and information from multiple sources to achieve refined entity position and identity estimates, and complete and timely assessments of resulting situations and threats, and their significance.”

Data sensor fusion is not tied to a specified type of data processing. Every kind of algorithm can be used that is suitable for the sensors used and the requirements. A data fusion system can therefore contain signal processing, pattern recognition, information theory, cognitive psychology, artificial intelligence and statistics.

The use of multisensor data fusion is hardly new. Humans and animals have both evolved and developed the capability to merge the outputs of different sensory modalities to improve their ability to survive. Let us take an example from [Abidi et al., 1992], originally published by [Newman, 1982]: Pit vipers and rattlesnakes have in their optic tectum (a midbrain structure found in vertebrates) neurons responding to both visual and infrared stimulus. Fig. 1-2 shows the left eye and the pit organ of a rattlesnake receiving information from the same region A in the environment. Both stimuli from region A are connected together on the surface of the optic tectum in a similar spatial orientation. Therefore each region of the optic tectum receives information from related regions of the environment. Neurons performing different kinds of fusion are able to detect different kinds of preys. For instance, “or” and “and”¹ neurons can be used to distinguish a warmblooded mouse from a cool-skinned frog. (The “and” neurons have been whimsically described as a mouse detector).

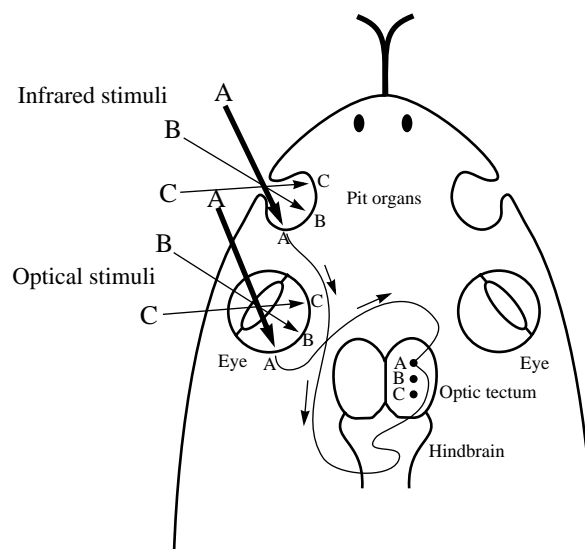


Fig. 1-2: Sensor fusion of related information into similar spatial place of the pit tectum

1. “and” and “or” in logical sense.

Fusion of different kinds of sensors

The way data fusion is implemented depends on the sensors used. We found it useful to divide sensors into two groups: *Continuous sensors* and *Event sensors*:

- *Continuous sensors* produce normally an uninterrupted flow of information of constant importance. This information can be integrated continuously into the position calculation process (for instance, position calculation by odometry which is continuously corrected by a compass). The accuracy of the values supplied by *continuous sensors* is normally constant. Therefore no particular action (like emergency stop) caused by this type of sensor has to be initiated. Fusing this kind of sensor can be achieved relatively easily, because the information of continuous sensors is always available and its importance (relative weight to other sensor inputs) is constant.
- *Event sensors* announce a special situation for the robot. The importance of a signal coming from an *event sensor* may be different and could trigger an immediate reaction (for example a touched bumper triggers an emergency stop or the lost contact with a laser reflector activates a search behavior). An *event sensor* signal can also be created from a *continuous sensor* signal. For instance a distance sensor (continuous sensor) can be used as an event sensor indicating objects getting too close to the robot. In such a case, the sensor can be seen as two sensors: As a continuous sensor delivering the distance to an object and as an event sensor supplying a signal if an object gets too close to the robot.

Fusing information coming from *continuous sensors* is normally done by probability calculations or other nonlinear functions. They deliver a continuous flow of results (for instance the updated robot position) and generally do not have to trigger special reactions for specific values of the signal. Therefore such fusion processes can be easily integrated into sensor data streams without affecting other modules like navigation or path planner modules.

As already mentioned, information coming from *event sensors* usually triggers a special behavior or an exceptional calculation. It usually initiates a decision which causes a change of some of the internal robot states. Fusing the information of event sensors requires usually less mathematical complexity but more expenditure for program structure. Thus, fusing event sensors is often less reliable because the problem cannot be modeled extensively and is therefore programmed in a more heuristic manner.

Both odometry correction algorithms presented in chapter 2 and 3 in this thesis belong to the second category, i.e. fusion of event sensors.

1.5 The importance of odometry in robot positioning

Odometry is the calculation of the actual position by integrating the traveled distance which is measured by the wheel rotation. It has not to be confused with *Dead Reckoning* which is derived from “deduced reckoning” of sailing days (see <http://www.teleplex.net/timonier/book18.html> for further details) and calculates the actual position by average speed and direction information.

The concept of *odometry* was first applied in 1910 for automobile navigation. The aim was to replace paper maps in order to eliminate the stress associated with route finding [Catling I., 1994]. This rather primitive but pioneering system counted the wheel rotation to derive longitudinal displacement and the steering wheel to calculate heading. Unfortunately the cumulative errors precluded its ultimate success.

The efficiency of odometry is proven by nature as well. Wehner [Wehner R., 1992] shows in the chapter 3 of the book *Animal Homing* that Saharan desert ants *Cataglyphis* are able to find their way home by using only direction and distance information. It seems that the distance information is gained from optical flow and compass direction from celestial cues, primarily from the polarization pattern of the blue sky. The example in fig. 1-3 shows that Saharan desert ants - during their search for food - cover a distance of 600 meters in about 19 minutes in a very accidental way. After that, they return straight home, in close to 6 minutes, 140 meters away. This is a very impressive performance considering the ant’s body length of 1.2 cm.

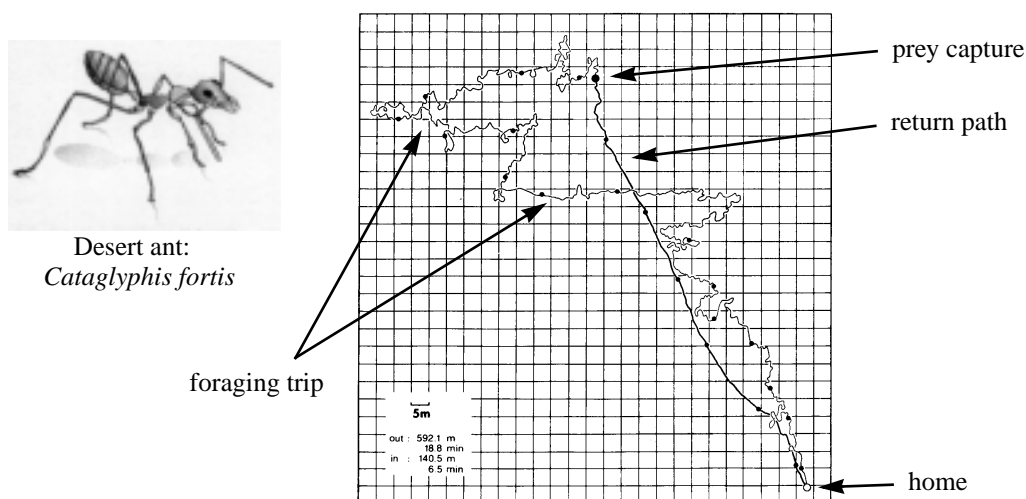


Fig. 1-3: Foraging and return of a desert ant “*Cataglyphis fortis*” [Wehner R., 1992]

Borenstein (see section 2.2) shows also that robots are able to perform astonishing results by navigating only with dead reckoning. He uses different methods to improve the odometry calculation.

However, odometry is very often used as robot positioning system because the needed wheel speed sensors are normally already implemented to stabilize the wheel speed (PID controller). This reduces the additional expenditure for odometry implementation to software. Therefore considerable effort is devoted to the correction of the cumulative odometry error using different systems. Kalman filter is a well-known method for data fusion and position estimation [Grewal et al., 1997] [Crowley et al., 1993] [Brown et al, 1992]. For example Cécile Durieu et al. [Durieu C., 1995] presented a fusion application for location of mobile robots using odometry and a panoramic laser telemeter. A similar approach is presented by Marek Piasecki [Piasecki M., 1995]. He uses multiple hypothesis tracking to determine the source of an observed signal in order to perform position estimation by comparing this source with corresponding map entries. However, in both papers the characteristics of the landmarks and their posi-

tion are known. On the other hand, H. Xu [Xu H. et al., 95] fuses a low cost gyroscope with dead reckoning signals to estimate the position. All three approaches use Kalman filter for position estimation.

1.6 The problem of modularized data processing

The usual approach to attack a general data processing problem is to break down the processes into modules, each of them with a well defined input/output interface. Thus modularized, a complex system is much easier to understand and develop. Each module performs a well specified data processing task and ignores every other kind of signal information which unconsciously could be of interest. Fig. 1-4 shows the typical data processing chain for a positioning algorithm. Sensor information is read and pre-processed independently. For instance, distance information is normalized into categories such as *close*, *middle* and *far away*. The image delivered by a camera is analyzed for vertical and horizontal edges and the direction information coming from a compass is divided into N, S, E and W. Afterwards, the pre-processed information is fused and abstracted in order to recognize some important situations, i.e. the edges are matched with the shape of landmarks which depends again on the direction of view (using a compass). Such recognized landmarks (marked by a cross) can be used to build a map and to navigate the robot.

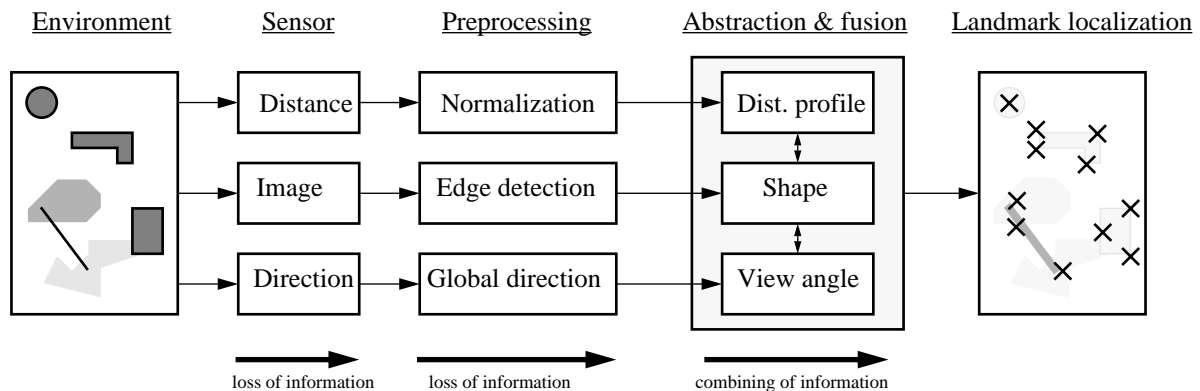


Fig. 1-4: Simplified landmark recognition by several processing modules

This could be called the Cartesian approach, following René Descartes (1596-1650) [Descartes R., 1637] who, in his “Discours de la méthode” explained his strategy: “to divide the problems I shall study as many parts as possible”. But, since we cannot really understand the robot’s world, we cannot be sure to have chosen the best possible modularization. Modularizing perception the way it is done in fig. 1-4 might very well entail a loss of information. Thus, the Cartesian way, one of the corner stones of modern science, should not be applied blindly.

Another strategy is to keep all the data available to all sensors and let the system discover by itself how to perceive the world. This is what is done with neural networks. After the learning phase, they know the relative attention to pay for different input vectors and store them into *weights*.

Nevertheless, such self learning systems are often trained to recognize foreseen features or they receive already preprocessed information. So the problem of modularized data processing is still existing.

The motivation for this thesis was to create a landmark recognition system which would avoid as much as possible such problems of modularized data processing. Hence:

- The landmarks should not be predefined in a way that specific signal preprocessing becomes necessary.
- The system should be capable of recognizing autonomously a combination (correlation) of signals which can be used as a landmark.
- This has to be done without any prior knowledge about the related sensors and even without any knowledge of the reason generating these signal events.

Fig. 1-5 shows in comparison to fig. 1-4 the structure of the system that include the changes mentioned. The preprocessing and abstraction module is replaced by different types of classifiers, which divide all sensor stimuli into classes. Every class (so every combination of stimuli) can be declared as a landmark, if it meets some later explained conditions. The way of classification can be slightly adjusted depending on the number and quality of classes. The resulting landmarks are unknown in advance but well adapted to the characteristics of the environment and to the sensor's abilities. The purpose of this thesis is to show how far such an adaptive system can be pursued.

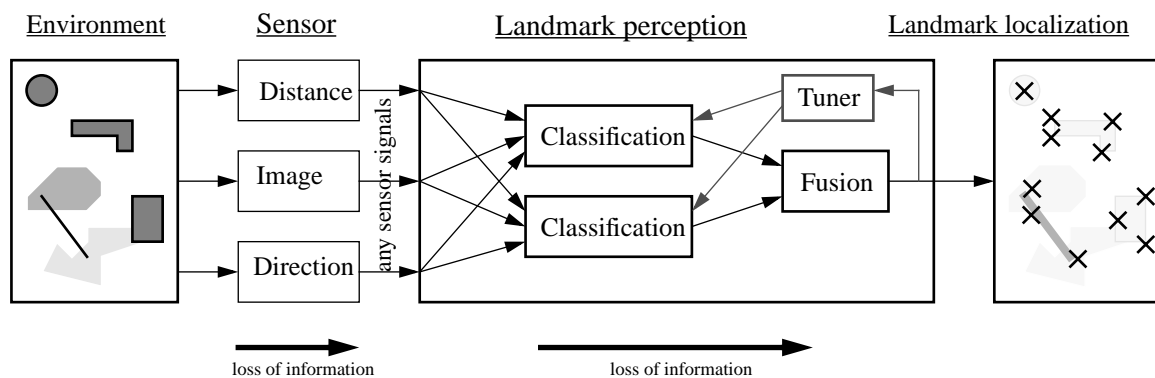


Fig. 1-5: Recognition of useful part in signals, independently of the sensor type

The example shown in fig. 1-6 shows the effect of such a system:

A room consisting of right-angled walls contains one diagonal wall as well. This significant exception (which was perhaps even not intended) will be automatically recognized by the mentioned classifier as a suitable landmark. Such a system could discover the irregularity of a natural environment and use them as landmarks. Localization systems as shown in fig. 1-4 cannot take advantage of such unintended situations.

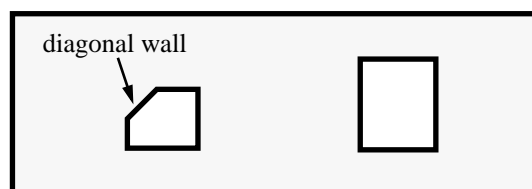


Fig. 1-6: Rectangular robot environment with a significant exception

1.7 Related Work in the field of model-less learning

The work mentioned in this section shows some different approaches for model-less learning of an environment.

Benjamin Kuipers and David Pierce presented in the paper *Map Learning with Uninterpreted Sensors and Effectors* [Kuipers B., 1997] an algorithm which finds the dependences between sensors and actors by applying and adapting models of a given set. The approach is divided into several levels. A rich set of models makes the algorithm very adaptive to a large range of sensor configuration. The result is a hierarchical model of the robot's world.

Lin and Hanson [Lin et al., 1993] demonstrate in the paper *On-line learning for indoor navigation: Preliminary results with RatBot* learning of a topological map of locally distinctive places. They use physical robots with 16 sonar sensors and 16 infrared sensors. The work is inspired by Kuipers and Byun [Kuipers B., 1991], but they use reinforcement learning¹ to train the local control strategies, rather than engineering them by hand. However, the target behavior like corridor following are specified by a human teacher, so the robot is rewarded if it moves along the corridor without bumping into obstacles.

Their approach is complementary to Kuipers and Pierce because they specify its own target behaviors, eliminating the need for the human teacher. On the other hand, Lin and Hanson specify the desirable behaviors by defining appropriate reward signals and then letting the robot learn on its own how to get the rewards.

An efficient approach to combine sensor events with a robot's position in order to perform robot navigation is shown by H. A. Mallot [Mallot H.A. et al, 1995]. He presented a view-based approach to map learning and navigation in a maze. The topological structure of a maze is defined from the sequence of images and perceptions experienced while exploring the maze.

Jürgen Schmidhuber [Schmidhuber J., 1997] goes one step further in his paper *What's Interesting*. He investigates the characteristic of interesting information. He says that interestingness depends on the observer's current knowledge and computational abilities. Information appears either trivial or random if there is either too much or too little known about it. He implemented this idea in a "curious" and "creative" explorer with two coevolving "brains".

Another interesting work, but in the domain of speech classification, is done by Virginia R. de Sa and Dana H. Ballard [de Sa, Ballard, 1998]. They use temporal correlation between sensations of different sensory modalities (lip motion and sound) to recognize "interesting events". However they want to avoid clustering of multimodal patterns which would prevent adequate performance in the individual modalities. So they describe an algorithm that avoids the intractable task of modeling cross-modal associations but uses this useful structure to derive its own internal target signals for classifiers in the individual modalities. The result is a network which classifies visual and auditory stimuli with performance comparable to supervised networks.

1. The reinforcement-learning algorithm is a neural-network version of Q-learning [Lin L.J., 1993]

2 Supervised Passive Positioning System (SPPS)

This chapter describes the origin and the propagation of odometry error and a supervised method for odometry correction. However, it is mainly focused on modeling a new odometry error model for two wheeled autonomous robots.

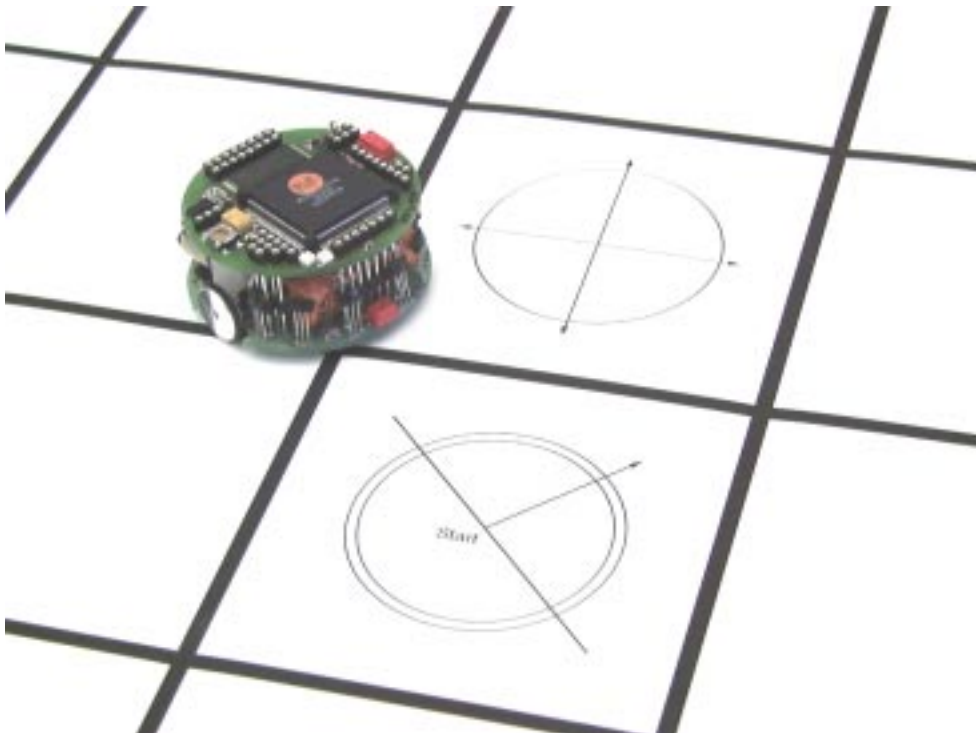


Fig. 2-1: Khepera performs odometry correction by detecting grid lines on the floor

Introduction

The experience presented performs odometry correction for an autonomous robot, based on black grid lines painted on a white floor (see fig 2-1). These grid lines are detected by a single light sensor underneath the robot, which cannot distinguish between horizontal and vertical lines. The robot knows the geometry of the grid lines (model) and its estimated position which is calculated by odometry (see section 2.2).

The robot's estimated position is needed to distinguish between the detection of a horizontal and a vertical grid line (both lines cause the same sensor signal). Thus, a detected grid line can be used to improve the estimated robot position.

2.1 Overview

For autonomous robots, navigating and interacting with their environment are fundamental skills. Depending on their task and the environment, robots need different techniques to work and survive in their environment. For example, autonomous vacuum cleaners¹ for swimming pools do their job perfectly well using a simple strategy: turning over when touching a wall.

On the other hand, an apparently similar and simple job like moving an autonomous robot inside an office becomes a technical challenge. The major difficulty for the robot is the spatial representation of the environment, and how to evolve a strategy to navigate in a safe and exhaustive manner, see [Knierim T., 1991] [Pau L. F., 90].

Various navigation systems can be found on the market to provide a robot with information about its position. They can be separated into *absolute positioning systems* such as Global Positioning System (GPS) and *relative positioning systems* such as odometry or acceleration sensors [Welch S., 92].

Absolute positioning systems deliver accurate position information without considerable drift in the long term, but they are relatively expensive. On the other hand, position delivered by a *relative positioning system* drift with time (see fig. 2-2).

The approach presented here tries to extract both advantages by using odometry as a cheap relative positioning system and grid lines on the floor as an absolute positioning system.

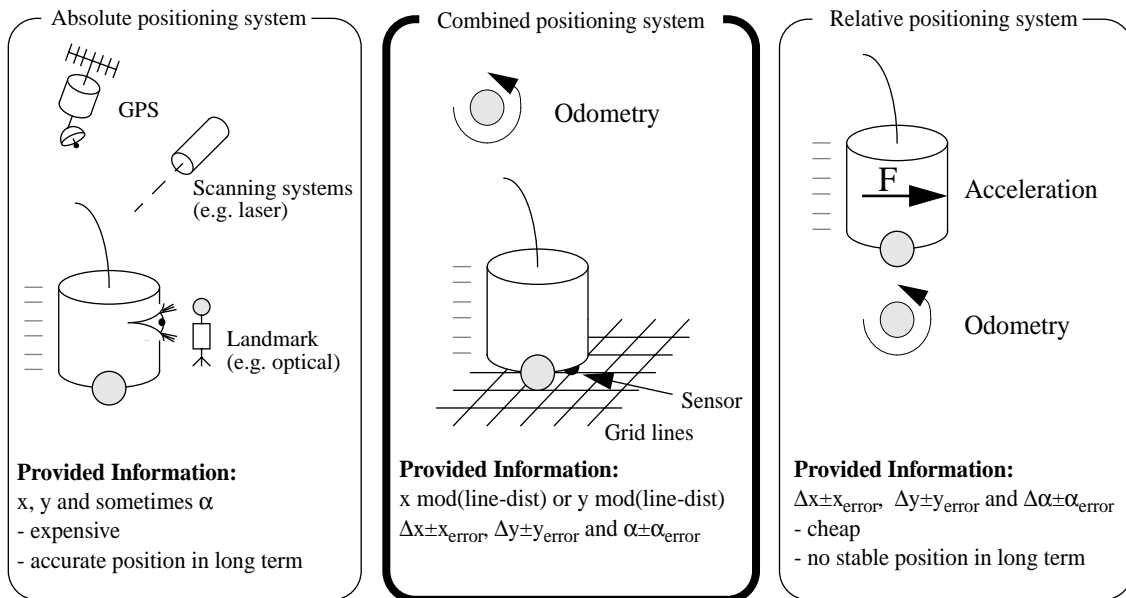


Fig. 2-2: Combination of different positioning systems

The position of the robot is represented by the position x , y and direction α . Odometry delivers the information \overline{x} , \overline{y} and $\overline{\alpha}$, but with a drift in the long term (therefore the variables are marked with an overline). The grid line delivers the absolute x or y position modul grid line distance. Thus:

$$(x, y, \alpha) = (\overline{x}, \overline{y}, \overline{\alpha}) \otimes [(x \bmod \text{gridlinedist}) \text{ OR } (y \bmod \text{gridlinedist})].$$

The presented fusion process (indicated by '*') is based on a new model of the position probability distribution and additionally supported by an interactive pilot system which influences the robot trajectory in order to take better advantage of the grid lines (see section 2.6 on page 28).

1. <http://www.electrolux.com.ru/robot/meny.html> or <http://diwww.epfl.ch/lami/robots/K-family/vacuum.html>

2.2 Introduction to odometry correction

The following sections introduce the problem of odometry error and grid line correction by comparing one and two dimensional cases. The difference makes it clear that a new position probability model is needed, which is presented in this section as well. The presented correction algorithm is designed for two wheel robots and has been tested on the Khepera^{®1}, which is often used in our laboratory.

Before correcting the estimated position of a robot, we have to know how to calculate this estimated position which consists of the position x , y and the robot direction α . Using the angle α to describe the robot direction can spoil the accuracy of the calculation and demands a special case treatment in the calculation program. The section "Different ways to calculate the odometry" in the Appendix on page 84 contains some practical hints on how to implement odometry calculation. A very comprehensive survey on several mobile robot positioning methods is given in [Borenstein et al, 1996].

2.2.1 Properties of odometry errors

The odometry error is the difference between the position on which the robot "thinks" to stay on (i.e. calculated by odometry) and its actual position (often measured by external devices). We divide the cause of the odometry error onto:

- *Systematic error*, which is caused by kinematic imperfections of the robot, as for instance unequal wheel diameters, misalignment of wheels or limited encoder resolution or sampling rate. The systematic error depends on the vehicle specification and stays almost constant over prolonged periods of time. Thus it can be examined and eliminated to a large extent. Borenstein and Feng describe in the paper *Measurement and Correction of Systematic Odometry Errors in Mobile Robots* [Borenstein, Feng, 1996] a method to investigate and compensate the systematic odometry error. The approach is simple but nevertheless very powerful and is mainly based on measuring the deviation of a robot running along a defined trajectory and compensating the odometry calculation by the measured parameters. However, the practical usefulness of this approach is questionable because the measured parameters can drastically change. For example, a different load distribution of the robot or worn wheels strongly influences the systematic error and requires frequent re-calibration of the systematic error.
- *Non-Systematic error*, which may be caused by wheel-slippage or irregularities of the floor. The non-systematic error is by definition unpredictable (if the environment is insufficiently known which is almost always the case). It is caused by interaction of the robot with unpredictable features of the environment. However Borenstein [Borenstein J., 1994] and [Borenstein J., 1995] shows that non-systematic error can be drastically reduced by using redundant encoder data. He uses several robots or only one robot with several redundant wheel encoders to compensate the odometry error. However, these approaches need special robot architectures and cannot be implemented on a simple two wheel robot like the Khepera.

1. The Khepera robot was designed at the LAMI EPFL Lausanne and is now marketed by K-Team SA Switzerland (<http://www.k-team.com/>)

2.2.2 Combining of systematic and non-systematic odometry error

In order to develop an odometry correction method, we need a model of the odometry error. The aim is to find a model which describes both kinds of error together (systematic and non-systematic). The influence of dynamical moments are not considered.

- Developing a model of the *non-systematic error* is very difficult because of its unpredictable characteristic. However, if we exclude “strong external disturbances” like hitting a wall or spinning wheels, we can model the difference between the real wheel speed V_{wheel} and the effective robot motion V_{motion} caused by this wheel by a Gaussian distribution $p_{\text{nonsys}}(V_{\text{dif}})$ (see eq. 2-1).

$$V_{\text{dif}} = V_{\text{motion}} - V_{\text{wheel}} \quad p_{\text{nonsys}}(V_{\text{dif}}) = \frac{1}{\sqrt{2\pi\sigma}} e^{-\left(\frac{(V_{\text{dif}}-\mu)^2}{2\sigma^2}\right)} \quad (\text{Eq. 2-1})$$

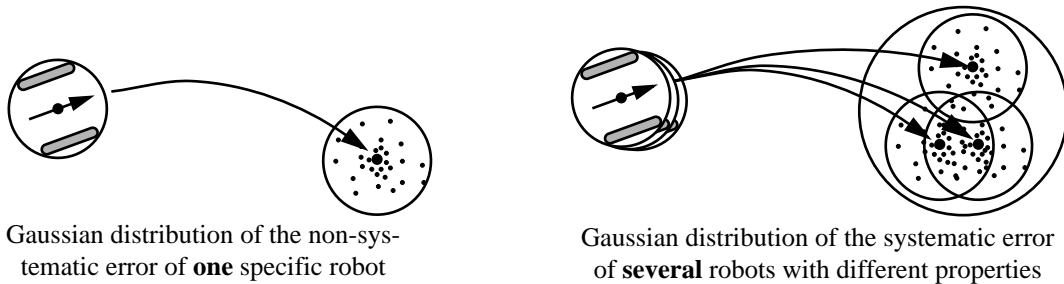
- The systematic error depends strongly on the specific robot. Therefore, $p_{\text{sys}}(V_{\text{dif}})$ is constant if the robot characteristic does not change at all. See case 1 in eq. 2-2.

Otherwise, we have to assume a gaussian distribution of the wheel diameter error, misalignment of wheels and other parameter errors in a population of robots of the same type. The model for *systematic error* is expressed again by a gaussian distribution of $p_{\text{sys}}(V_{\text{dif}})$ representing the difference between the real wheel speed V_{wheel} and the effective robot motion V_{motion} . See case 2 in eq. 2-2.

$$\begin{aligned} \text{case 1:} \quad p_{\text{sys}}(V_{\text{dif}}) &= \text{const} \\ \text{case 2:} \quad p_{\text{sys}}(V_{\text{dif}}) &= \frac{1}{\sqrt{2\pi\sigma}} e^{-\left(\frac{(V_{\text{dif}}-\mu)^2}{2\sigma^2}\right)} \end{aligned} \quad (\text{Eq. 2-2})$$

Fig. 2-3 shows that the cumulation of *non-systematic* and *systematic* error distribution is a convolution (folding) of two gaussian distributions. Because the convolution is as a multiplication in the frequency domain, we transform the gaussian function into the frequency domain to square it. Transforming the result back into the time domain results in a gaussian as well, but with different standard deviation σ .

Conclusion: Whether one considers the systematic error or not, the error distribution can be described by a gaussian distribution. Thus, all further calculations are based on a gaussian distribution.



$$\text{Convolution of two gaussian } f(t) \text{ and } g(t): f(t) \otimes g(t) = \int_{-\infty}^{\infty} (f(\tau) \cdot g(t-\tau)) dt = F^{-1}([F(v)] \cdot G(v))$$

$$F(v) = \int_{-\infty}^{\infty} e^{-ax^2} \cdot e^{ivx} dx = \underbrace{\int_{-\infty}^{\infty} e^{-ax^2} \cdot \cos(vx) dx}_{\text{Integration over a symmetrical range gives 0.}} + \int_{-\infty}^{\infty} e^{-ax^2} \cdot \sin(vx) dx = \underbrace{\sqrt{\frac{\pi}{a}} \cdot e^{-\frac{k^2}{4a}}}_{\text{Gaussian distribution}}$$

Fig. 2-3: Folding of the non-systematic and systematic gaussian error distribution

2.2.3 One-dimensional correction

The robot is equipped with a single light sensor underneath which detects the painted grid lines on the floor. We assume that the light sensor provides an ideal short pulse when crossing a grid line. This pulse will be used to synchronize the estimated position of the robot (calculated by odometry) with the real position (indicated by the grid lines on the floor).

The upper part of fig. 2-4 shows the robot passing some vertical grid lines. The robot is constantly calculating its estimated position by odometry. This estimated x position is indicated by a dotted line in the time-distance table in fig. 2-4. Between the detection of the grid lines, the dotted line (estimated x position of the robot) is rising constantly because the robot speed is assumed as being constant. The solid line in the time-distance diagram shows the real x position of the robot and is therefore rising with some irregularities due to the slipping wheels and other disturbances.

The estimated robot position (dotted line) is corrected to the x position of the closest grid line each time the light sensor detects a grid line. The gaussian error distribution surrounding the x position of each grid line (bottom diagram in fig. 2-4) indicated the odometry error if matched with the real position by a grid line detection. This can be used to recognize abnormal conditions and systematic drifts.

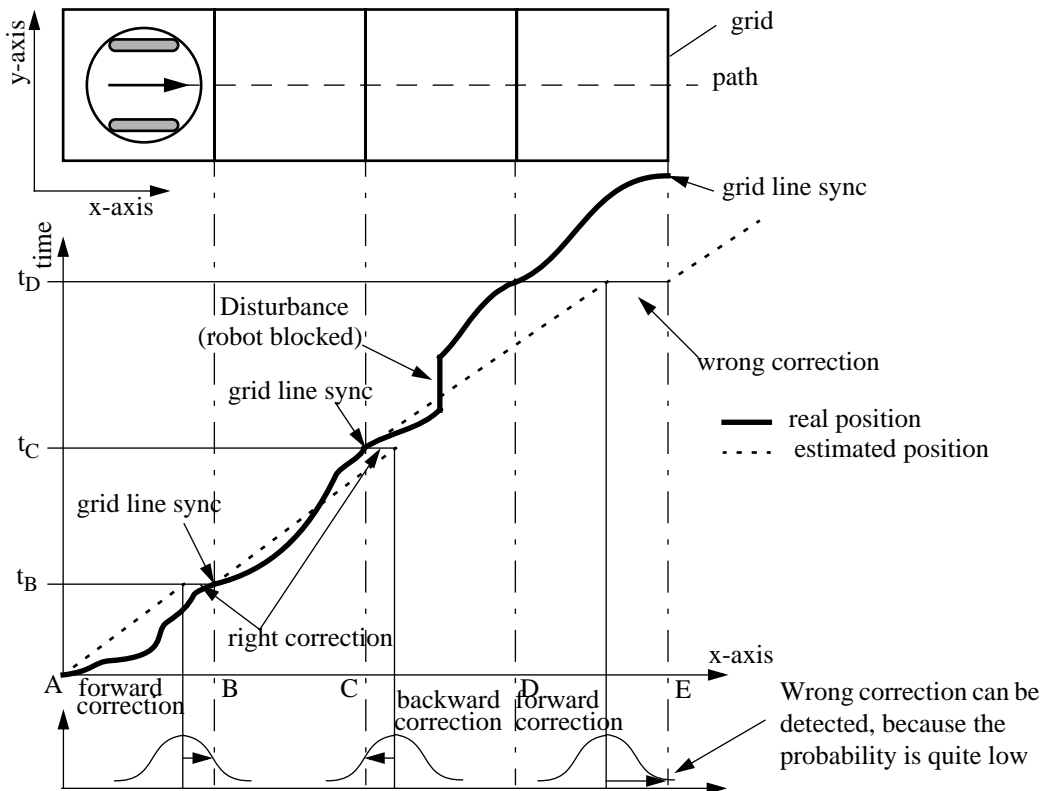


Fig. 2-4: Example of the fundamental technique to correct odometry error

The following paragraph explains fig. 2-4 in more details:

The robot starts at position A and is passing over the first grid line B at time t_B . At this time (t_B) the estimated position (dotted line) is a little bit behind the grid line B, which means that the robot moved faster than expected. The estimated position will therefore be corrected forwards to B. During the next sequence (B to C), the robot moves unintentionally slower and is therefore crossing the grid line C later than expected. The estimated position has to be corrected backwards, because point C is closer to the estimated position than point D. The last sequence shows a wrong correction due to a mechanical blockage of the robot. The light sensor indicates line D at time t_D . Unfortunately the estimated position (dotted line) is closer to point E than to point D, which results in a wrong correction. The robot lost its position because the odometry error was too big. This mistake can normally be detected, because the probability of the obtained point E is quite low (see bottom diagram in fig. 2-4).

Note: Counting the grid lines and using the number obtained as an indicator for the absolute robot position would solve the problem just described, but would cause more nuisance from missed or double counted lines.

2.2.4 Two-dimensional correction

In the previous one dimensional example, the position of the robot was corrected towards the position of the closest grid line. Such a technique cannot be used in a two dimensional environment with a right-angle grid network for synchronization. The reason appears clearly in fig. 2-5.

The robot is intended to drive straight forward which is marked by the dotted line (assumed trajectory). Each grid line passed generates an event which synchronizes the robot's x or y position to the x or y position of the closest grid line. Suddenly, the robot gets an event signal on the assumed position \times . The vertical grid line is closer, so the position marked by a empty circle \circ seems to be correct. However, it is much more probable that the robot followed the full line (real trajectory), because direction drift is more probable that distance drift (see section 2.3.1). This would explain the premature light sensor signal which was in fact generated by the horizontal grid line on position \bullet , even if this position is further away than the first assumed cross point.

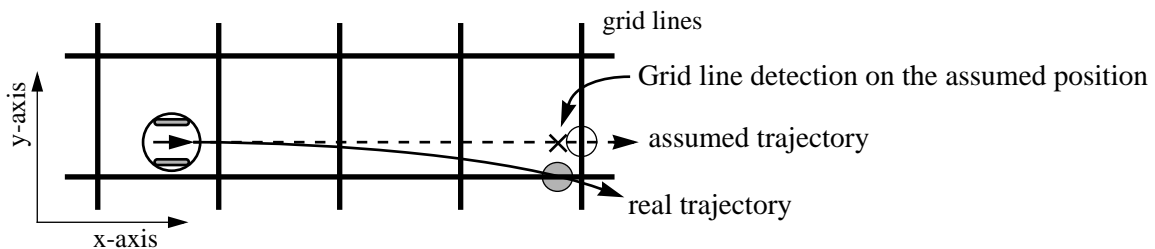


Fig. 2-5: Example: The closest grid line is not always the best

2.2.5 The traditional ellipse approach to distinguish x and y correction

The constantly growing positioning error can be described by a 3D bell curve, of which the z value indicates the probability for the robot to be found on the corresponding x, y position. The circles (or ellipses) in fig. 2-6 show the top view of a horizontal cut through the bell curves at a certain altitude (probability threshold). Circles are widely used to model the spatial position distribution [Piasecki M., 1995] [Pruski A., 96] of robots and for some robot architectures correspond very well to reality.

The ellipse is normally growing constantly in both dimensions. Each time the robot crosses a grid line, one dimension of the ellipse shrinks corresponding to the orientation of the crossed grid line. As already mentioned in the section before, the robot assumes the orientation of the grid line just crossed (thanks to odometry) and is therefore able to shrink the corresponding dimension of the ellipse.

Fig. 2-6 convincingly shows that alternately going over horizontal and vertical grid lines is important to keep the ellipse dimension small.

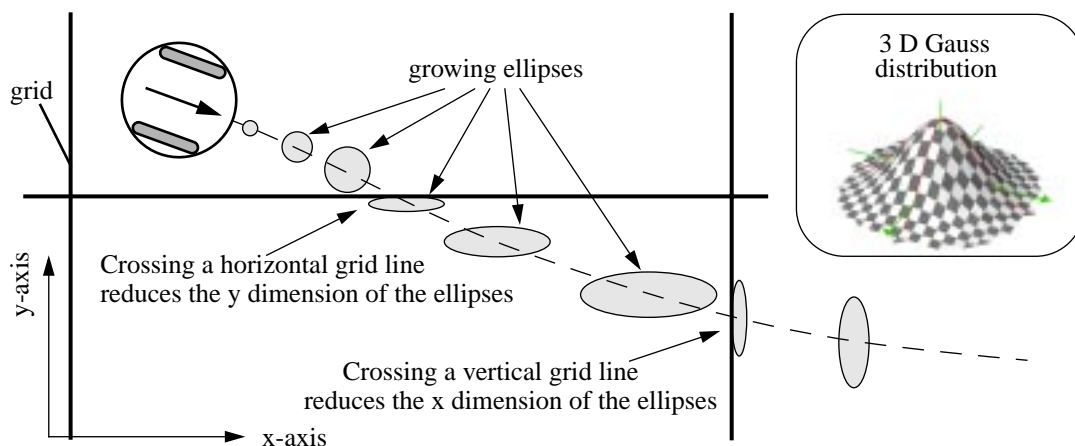


Fig. 2-6: Passing grid lines change the shape of the ellipses

2.3 The PPD model

The shape of the **Position Probability Distribution (PPD)** depends mainly on the robot's characteristics. This section examines the PPD shape for two wheeled robots like the Khepera[®]. A first idea of the distribution was given by an experiment which is presented in the introduction and later confirmed by mathematical derivation.

2.3.1 Introduction

Fig. 2-7 shows the result of an experiment in which a robot is launched several times from the same start position for a constant distance. The shape of all destination positions looks more like a "banana" than an ellipse. The result can be intuitively explained considering a constant wheel speed ω for both wheels but influenced by an individual speed error ΔV_x which represents the odometry error. The real speed can be described by $V_x = \omega \cdot r_{wheel} \pm \Delta V_x$ (x indicates the left 'L' or right 'R' wheel). An independent gaussian distribution for each wheel speed error ΔV_x results in the PPD shape showed in fig. 2-7.

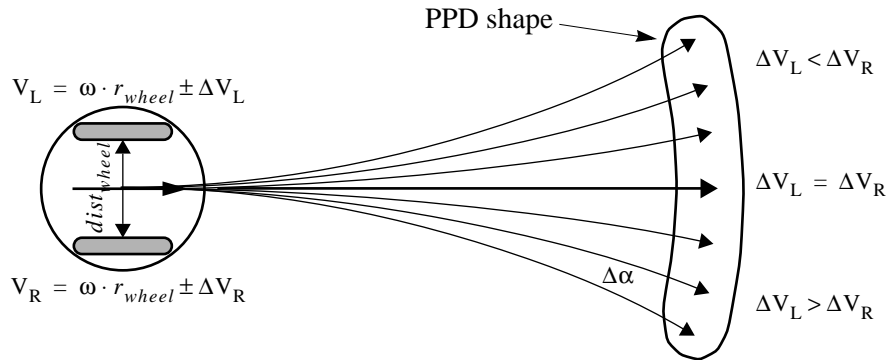


Fig. 2-7: Minor wheel speed difference causes a strong change of the robot direction

Due to minor wheel friction ΔV , the real speed V_x does not correspond to the driven wheel speed $\omega \cdot r_{wheel}$, hence $\omega \cdot r_{wheel} \approx V_x$. Experiments show that the average friction is about +/- 0.3%, so the distance covered is not affected in a crucial way. On the other hand the robot's direction is much more sensitive than the distance covered. The change in direction $\Delta\alpha$ can be calculated as follows:

$$\Delta\alpha = \frac{\Delta t \cdot (V_1 - V_2)}{dist_{wheel}} = \frac{\Delta t \cdot (\Delta V_1 - \Delta V_2)}{dist_{wheel}} \quad \begin{array}{l} \omega_x \cdot r_{wheel} + \Delta V_x = V_x \\ \omega_1 = \omega_2 \end{array}$$

The equation shows that the smaller the wheelbase the stronger the impact on the robot's direction due to different wheel friction. This lateral drift (change of direction) is much stronger than the drift in the direction of motion.

2.3.2 Geometrical calculation of the PPD model

We assume a rotation of the robot around a fixed point R. The deviation of the two wheels are not equal $\Delta V_L \neq \Delta V_R$ but constant. The radius r of the rotation depends therefore on the wheel speed difference $|\Delta V_L - \Delta V_R|$ and the wheelbase L . A lateral drift of the wheels will be ignored (see fig. 2-8).

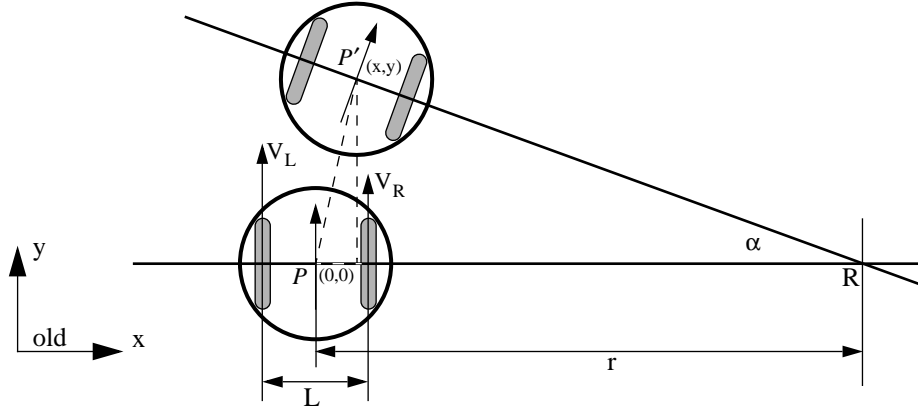


Fig. 2-8: Derivation of a PPD model

The following calculation derives an equation for x , y and α , depending only on V_L , V_R and L . The resulting x and y is the position and α the direction of the robot.

The relation between the wheel speed and the angle is described in eq. 2-1:

$$V_{\text{circumference}} t = r\alpha \longrightarrow \frac{V_L t}{r + \frac{L}{2}} = \frac{V_R t}{r - \frac{L}{2}} = \alpha \quad (\text{Eq. 2-1})$$

This double equation (eq. 2-1) allows the extraction of r and α as functions of V_L , V_R and L :

$$\begin{aligned} V_L t \left(r - \frac{L}{2} \right) &= V_R t \left(r + \frac{L}{2} \right) \longrightarrow \boxed{r = \frac{L(V_L + V_R)}{2(V_L - V_R)}} \\ \alpha &= \frac{V_R t}{r - \frac{L}{2}} = \frac{V_R t}{\frac{L(V_L + V_R)}{2(V_L - V_R)} - \frac{L}{2}} \\ &= \frac{V_R t \cdot 2(V_L - V_R)}{L(V_L + V_R) - L(V_L - V_R)} \longrightarrow \boxed{\alpha = \frac{V_L - V_R}{L} t} \end{aligned} \quad (\text{Eq. 2-2})$$

y and x can now be calculated by trigonometry:

$$\begin{aligned} y &= r \cdot \sin(\alpha) \longrightarrow \boxed{y = \left(\frac{L(V_L + V_R)}{2(V_L - V_R)} \right) \sin\left(\frac{V_L - V_R}{L} t \right)} \\ r - x &= r \cdot \cos(\alpha) \longrightarrow \boxed{x = \left[\frac{L(V_L + V_R)}{2(V_L - V_R)} \right] \left[1 - \cos\left(\frac{V_L - V_R}{L} t \right) \right]} \end{aligned} \quad (\text{Eq. 2-3})$$

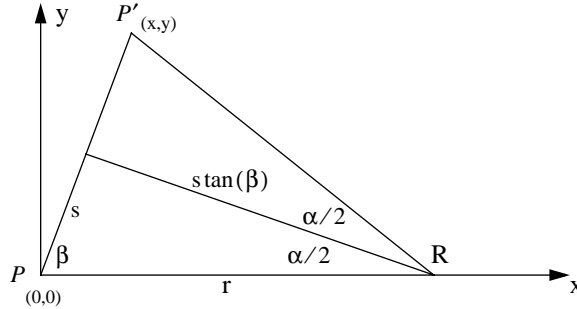
The resulting equations (eq. 2-3) supplies the position (x and y) of the robot, depending on the constant wheel speed V_L , V_R , the wheelbase L , and the time t .

2.3.3 Calculation and display of the PPD shape

In order to illustrate the distribution of the final position of the Khepera, we have to transform the eq. 2-3 into the following form:

$$\begin{aligned} x &= g(V_L, V_R) \\ y &= h(V_L, V_R) \end{aligned} \longrightarrow \begin{aligned} V_L &= k(x, y) \\ V_R &= l(x, y) \end{aligned} \quad (\text{Eq. 2-4})$$

We assume a constant wheel speed V_x and use a geometrical model to transform the variables:



The relation between (V_L, V_R) and (r, α) is already calculated (see eq. 2-2). We have now to find a geometrical relationship between (x, y) and (r, α) :

$$\begin{aligned} r &= \frac{s}{\cos(\beta)} = \sqrt{s^2 + s^2 \tan^2(\beta)} = s \sqrt{1 + \tan^2(\beta)} \\ \text{Replacement of } s \text{ and } \beta: \quad s &= \frac{\sqrt{x^2 + y^2}}{2} \quad \frac{y}{x} = \tan(\beta) \\ \frac{\sqrt{x^2 + y^2}}{2} \cdot \sqrt{1 + \frac{y^2}{x^2}} &= \frac{\sqrt{x^2 + y^2}}{2} \cdot \frac{1}{x} \sqrt{x^2 + y^2} = \boxed{\frac{x^2 + y^2}{2x} = r} \\ \alpha &= 2 \cdot \left(\frac{\pi}{2} - \beta \right) = \boxed{\pi - 2 \operatorname{atan}\left(\frac{y}{x}\right) = \alpha} \end{aligned} \quad (\text{Eq. 2-5})$$

Equation 2-6 calculates the velocity (V_L, V_R) as a function of the position (x, y) , this means the velocity needed for both wheels to reach the position (x, y) under the following conditions:

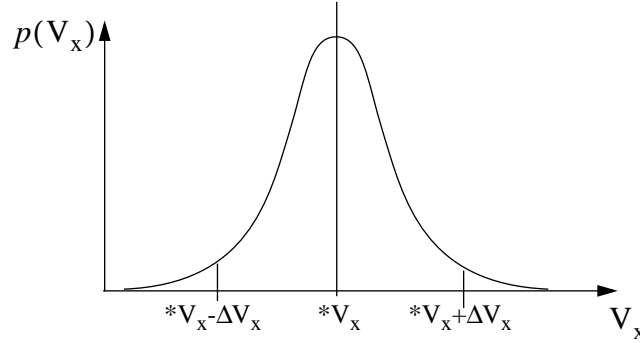
- The wheel speed is constant.
- All positions on the y-axis ($x=0$) result in a solution that is ∞ , because of the geometrical derivation.

$$\begin{aligned} \alpha &= \frac{V_L - V_R}{L} t = \pi - 2 \operatorname{atan}\left(\frac{y}{x}\right) \quad \longrightarrow \quad \text{A: } V_L - V_R = \frac{L}{t} \left(\pi - 2 \operatorname{atan}\left(\frac{y}{x}\right) \right) \\ r &= \frac{L}{2} + \frac{V_R L}{V_L - V_R} = \frac{L(V_L + V_R)}{2(V_L - V_R)} = \frac{x^2 + y^2}{2x} \quad \longrightarrow \quad \text{B: } \frac{V_L + V_R}{V_L - V_R} = \frac{x^2 + y^2}{Lx} \\ \longrightarrow \quad \text{B' = A B: } \quad V_L + V_R &= \frac{x^2 + y^2}{tx} \left(\pi - 2 \operatorname{atan}\left(\frac{y}{x}\right) \right) \end{aligned}$$

$$A+B': \quad V_L(x,y) = \frac{1}{2t} \left(\pi - 2 \operatorname{atan} \left(\frac{y}{x} \right) \right) \left(\frac{x^2 + y^2}{x} + L \right) \quad (\text{Eq. 2-6})$$

$$A-B': \quad V_R(x,y) = \frac{1}{2t} \left(\pi - 2 \operatorname{atan} \left(\frac{y}{x} \right) \right) \left(\frac{x^2 + y^2}{x} - L \right)$$

We showed already in section 2.3.1 that the resulting velocity distribution (V_L, V_R) can be represented by a gaussian distribution around V_X (see fig. 2-9). This distribution will generate the “banana” shape. We assume a wheel speed V_x which is composed of the pretended wheel speed $*V_x$ and an error ΔV_x which is modeled according to a gaussian distribution.



Gaussian distribution:

$$p(V_x) = \frac{1}{\sqrt{2\pi}\sigma} e^{-\left(\frac{(V_x - \mu)^2}{2\sigma^2}\right)} \quad \begin{array}{l} \mu = *V_x \\ \sigma = \frac{1}{2}\Delta V_x \end{array}$$

Fig. 2-9: Gaussian distribution of the wheel speed

μ is the pretended wheel speed and σ represents the standard deviation. We choose an interval from $*V_x - \Delta V_x$ until $*V_x + \Delta V_x$ covering 95% of the entire gaussian-surface:

$$\boxed{\sigma = \frac{1}{2}\Delta V_x} \quad \text{because} \quad \int_{-2\sigma = -\Delta V_x}^{2\sigma = \Delta V_x} \left[\frac{1}{\sqrt{2\pi}\sigma^2} e^{-\left(\frac{x^2}{2\sigma^2}\right)} \right] dx \approx 95\% \quad (\text{Eq. 2-7})$$

Equation 2-7 shows that the standard deviation σ and therefore the size of the PPD model (banana) depends only on the assumed maximal speed error ΔV_x .

In order to calculate the PPD of the final robot location (which is a function of the gaussian distributed velocities), we must multiply the velocity probability function by the Jacobian determinant of the velocities with respect to the x and y locations:

$$f_{xy}(x,y) = f_{V_L}(V_L(x,y)) f_{V_R}(V_R(x,y)) \cdot \Sigma |J_{V_L V_R}(X(V_L, V_R), Y(V_L, V_R))|$$

$$\text{Jacobian Matrix: } J_{V_L V_R}(x,y) = \begin{bmatrix} \frac{\partial V_L}{\partial x} & \frac{\partial V_L}{\partial y} \\ \frac{\partial V_R}{\partial x} & \frac{\partial V_R}{\partial y} \end{bmatrix} \quad (\text{Eq. 2-8})$$

Inverting the product and the Jacobian:

$$f_{xy}(x,y) = \frac{f_{V_L V_R}(V_L(x,y), V_R(x,y))}{|J_{xy}(V_L(x,y), V_R(x,y))|}$$

$$\text{Inverted Jacobian Matrix: } J_{xy}(V_L, V_R) = \begin{bmatrix} \frac{\partial x}{\partial V_L} & \frac{\partial x}{\partial V_R} \\ \frac{\partial y}{\partial V_L} & \frac{\partial y}{\partial V_R} \end{bmatrix} \quad (\text{Eq. 2-9})$$

This final equation represents the probability of each position of the banana, located on the position (x, y) .

We use Mathematica^{®1} to calculate the equations and to visualize the PPD. The following image in fig. 2-10 shows the PPD produced by a Khepera[®] robot, moving straight ahead for 3 meters (see the scale beside the graphic in mm) with a wheel speed ($*V_x$) of 100mm/sec and a speed deviation of maximal $\pm 1\%$ ($\Delta V_x = 1\text{mm/sec}$, $\sigma=0.5$) for each wheel. The deviation of 1% is the result of some practical experiments with a Khepera robot on a clean table. Another robot, or the rolling conditions, will certainly change this value. However, the characteristics of the PPD shape will remain the same. The wheelbase L is 52mm and corresponds to that of a Khepera[®].

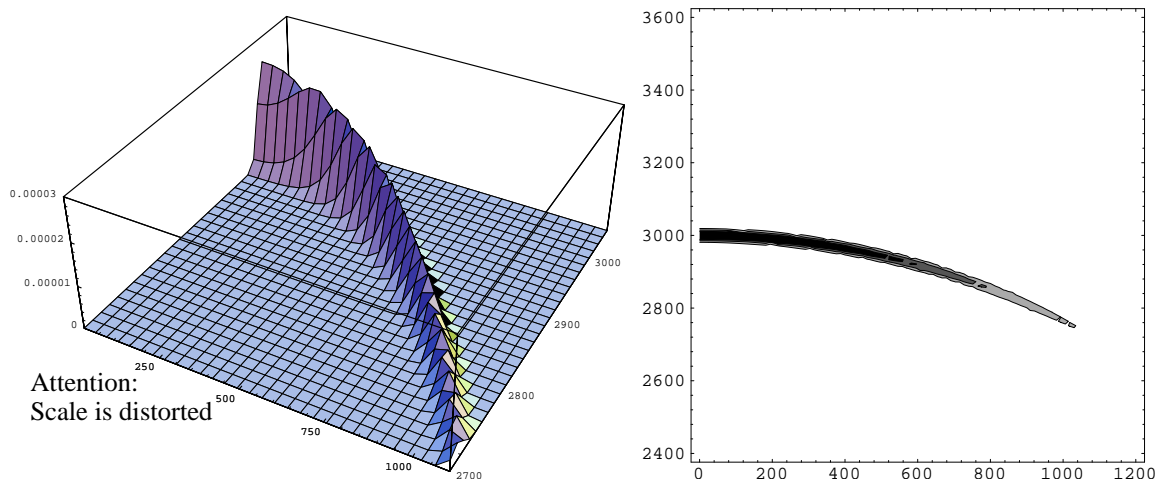


Fig. 2-10: PPD shape (right side). Path length 3m; straight ahead; wheel slip 1%

It is to be noted that the angular deviation is far bigger than the forward deviation. The robot has a lateral error of about 1 meter, as compared to a forward error of about 30 mm. The banana is like a part of a very thin arc, evenly decreasing to the end.

1. Mathematica[®] is a commercial software product by Wolfram Research Inc. [Wolfram, 1992].

2.3.4 Differences between the circular and the PPD model

Fig. 2-11 shows the impact of the just developed PPD–shape and demonstrates the difference to the traditional circular approach. A Khepera[®] robot starts at point P_t and turns slowly to the right. After a certain time Δt , a signal is supplied by the light sensor, indicating a grid line underneath the robot. The robot assumes its position at point $P_{t+\Delta t}$ which is not on a grid line and has therefore to be corrected according to one of the two probability distribution models:

- **PPD model:** The three dimensional PPD (see fig. 2-10) is cut by the grid lines, which are known by the robot. The resulting two dimensional cut is shown on the right side on fig. 2-11. The highest point (maximum) of the cut indicates the position with the highest probability (A).
- **Circular model:** The three dimensional model of the circular gauss distribution (see fig. 2-6) is cut by the grid lines. The resulting position B is always the closest grid line element to $P_{t+\Delta t}$.

It can be clearly seen that the two models produce two very different results. This intuitive example does not show the PPD distribution in real proportion to the circular distribution to make the graphic more clear. However, it shows the importance of choosing the correct position distribution model.

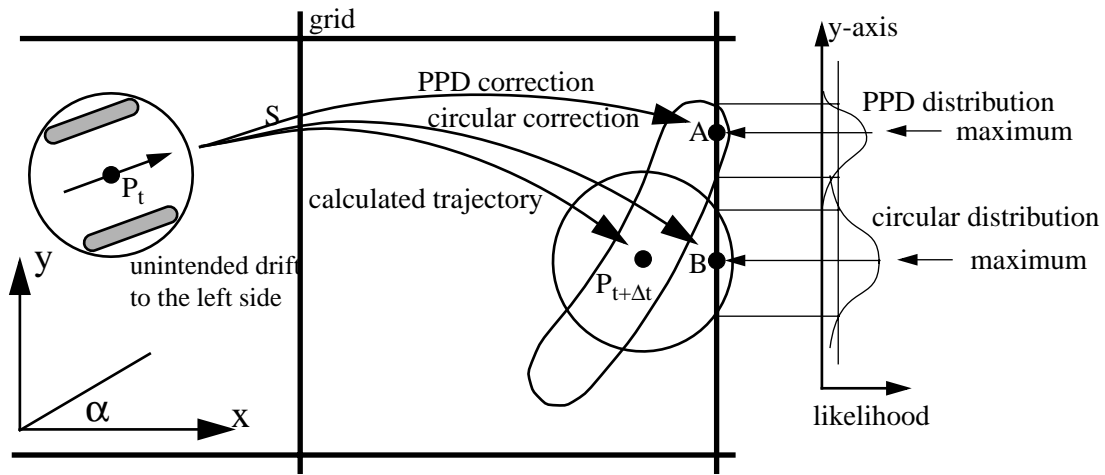


Fig. 2-11: Difference between the PPD and traditional circular approach

2.3.5 Deformation of the PPD by curves and corners

Robots move rarely straight ahead. Therefore it is important to analyze in which manner their PPD–shape changes when they take curves and corners. Unfortunately, the equations derived above for calculating the PPD–shape depend on constant wheel speeds (V_L and V_R). Therefore, corners and variable-radius curves cannot be calculated. The following iterative approach will be used to calculate this kind of PPD–shape (see section 2.3 "The PPD model"):

$$y = \left(\frac{L(V_L + V_R)}{2(V_L - V_R)} \right) \sin\left(\frac{V_L - V_R}{L} t \right) \quad (\text{Eq. 2-10})$$

$$x = \left(\frac{L(V_L + V_R)}{2(V_L - V_R)} \right) \left[1 - \cos\left(\frac{V_L - V_R}{L} t \right) \right]$$

We use Mathematica[®] to sweep V_x from $(*V_x - \Delta V_x)$ to $(*V_x + \Delta V_x)$ and for the following visualization of the PPD. There is no significant difference between the PPD shape calculated by the derived equation or by the iterative calculation because of the high numerical precision of Mathematica[®]. In order to visualize the path covered by the robot, we use only the iterative calculation for all further simulations.

2.3.6 Examples

The following examples illustrate the behavior of the PPD in curves and corners. The right graphic shows the same PPD as the left one, but from a top view. The dotted line on the right graphic indicates the robot's ideal trajectory without any wheel slip. All examples show the PPD 4 times (for each meter). A short source listing of the PPD simulator generating the following examples can be found in the "Appendix" on page 87.

Example 1: Curve

The following simulation in fig. 2-12 shows a robot constantly turning right for 4 meters. The left wheel is turning 4.08% faster than the right one (left: 102mm/sec, right: 98mm/sec) in order to perform a curve. The maximal wheel speed derivation is 1% for both wheels.

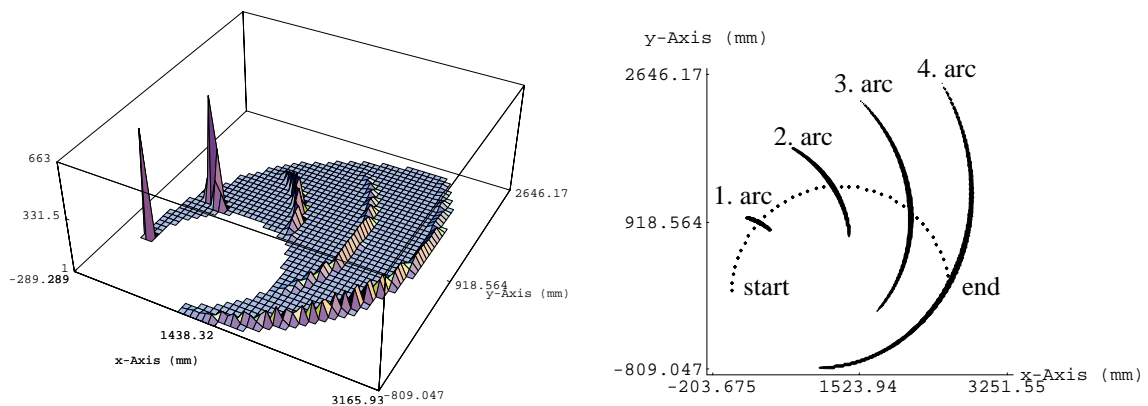


Fig. 2-12: Example 1: PPD behavior in a curve

Note: It is interesting to observe that the orientation of the PPD changes only slowly and it does not have at all the same direction as the robot.

Example 2: Turning back after one meter

This simulation in fig. 2-13 shows the PPD of a robot turning left for 90° after one and two meters. The maximal wheel speed derivation is 1% for both wheels.

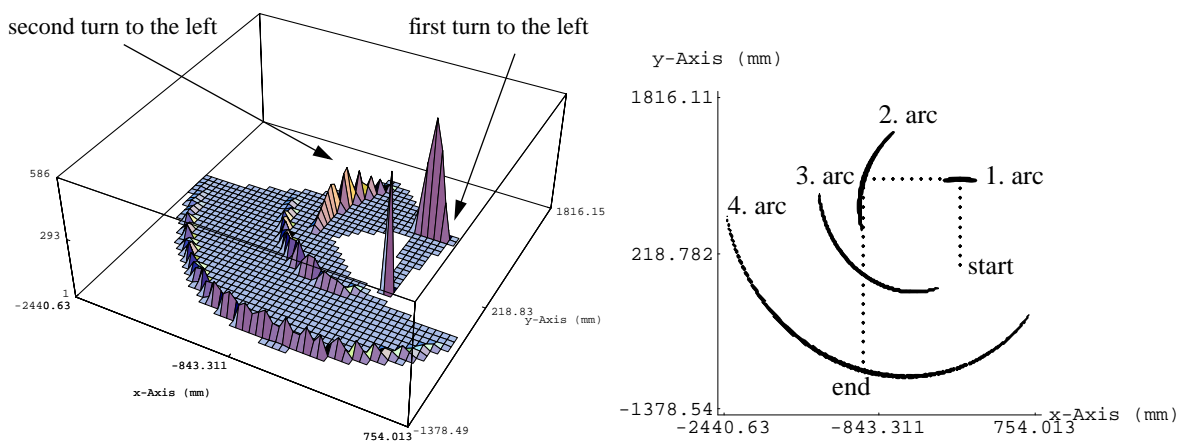


Fig. 2-13: Example 2: PPD behavior after a turn back

Note: The orientation of PPD doesn't correspond to the average orientation during the experiment, therefore the orientation of the second PPD is not 45° as expected ($\frac{0^\circ + 90^\circ}{2} = 45^\circ$).

Example 3: Turning back after 2 meters

The last example in fig. 2-14 shows the PPD of a robot moving straight ahead, but turning back (180° turn) after 2 meters. The maximal wheel speed derivation is 1% for both wheels.

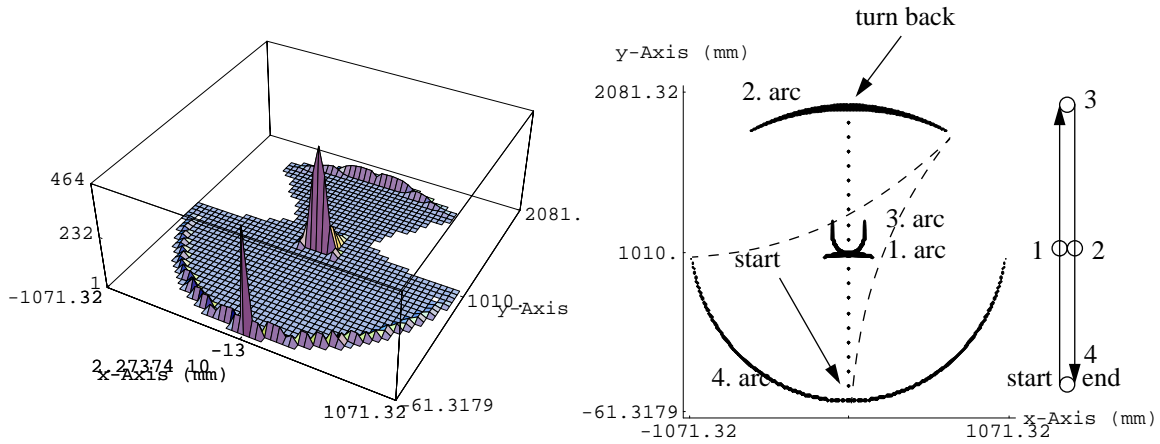


Fig. 2-14: Example 3: PPD decrease after a turn back

Note:

- The robot will again, after 2 meters, reach its starting point. Nevertheless, the corresponding second PPD is bent (downwards), like after a move in the direction of the negative y-axis. This illustrates that the PPD is more influenced by the covered trajectory than by its final position.
- The third PPD is much smaller than the previous PPD which shows that the size of a PPD can also decrease. Nevertheless, the PPD is strongly bent, indicating that the orientation of the robot is somewhat uncertain in contrast to its position.

The result becomes clearer by comparing the comportment of PPDs and water waves. Outspreading waves can be reconcentrated by obstacles and regain more power. For instant waves on the beach are created by a similar effect. Fig. 2-15 shows the re-concentration of a PPD step by step.

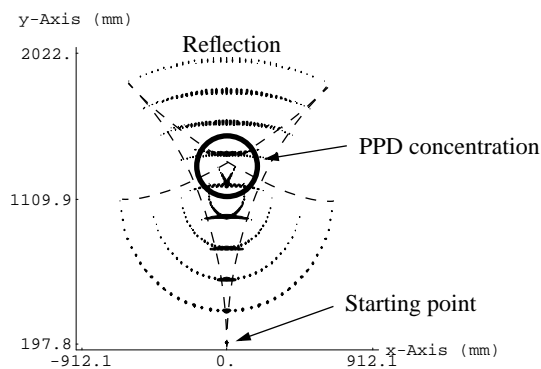


Fig. 2-15: Spreading out and concentrating of the PPD

2.3.7 Conclusion

- It is interesting to note that the shape of the PPD remains a segment of an arc. Even the thickness of the line is independent of the trajectory.
Condition: the wheel's drift has to be constant. The difference consists only in radii and orientations.
- The latest example shows also that a certain motion strategy can shrink the PPD. This result is surprising but nevertheless understandable. To better comprehend, follow the dotted line in fig. 2-14 which indicates the trajectory of the robot with a constant drift to the right side.

2.4 Statement of the problems

The suggested grid-solution is an easy way to correct odometry errors with limited additional expenses. Nevertheless, this simplicity hides some problems which will be described in the following section.

2.4.1 Angular drift, caused by inaccuracy robot direction

Because the robot knows the geometry of the painted grid on the floor, an independent correction of the x and y position while crossing a grid-line seems to be an easy way to continuously correct the corresponding dimension of the robot's position (see fig. 2-16). Unfortunately, this approach ignored the angle α of the robot, whose estimation accuracy will decline with time. This causes a strong angular drift of the robot, which cannot be recognized by this algorithm. The robot will get lost.

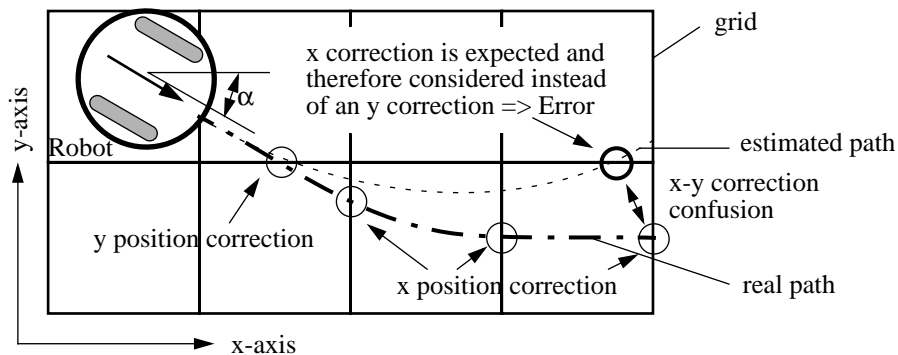


Fig. 2-16: Wrong correction because of confused grid lines

We notice that this angular drift is the main reason for the position error. The distance and speed error while moving straight ahead are comparatively small (see section 2.3 "The PPD model").

2.4.2 Information distribution of a grid

Another problem of this approach is the information distribution on the grid. Not every region of the grid field have the same positive effect. There are large regions containing no information or a helpful grid line which was by chance missed by some few cm. An active element is needed directing the robot to regions containing interesting information. This active element called "grid fitter" is presented later in section 2.6.

2.5 Strategy for PPD, robot position and direction correction

The strategy for odometry correction is divided into two sections.

- This section describes how to use the knowledge of the PPD to correct in a *passive manner* the estimated position of the robot.
- The section 2.6 on page 28 describes how to influence the trajectory of the robot in order to improve the result of the passive correction system.

2.5.1 Passive correction, using the PPD

The robot's position is chiefly obtained by odometry. The suggested algorithm corrects the odometry errors by knowing the evolution of the PPD motion and by taking advantage of the thin PPD-shape.

The following conditions have to be satisfied:

- The initial position and starting direction of the robot have to be known.
- The robot has to be equipped with a light sensor fixed underneath to detect the painted grid.
- The robot must know the characteristics of the painted grid (organization, grid distance).

2.5.2 Correction of the position

The robot starts from a known position and updates its PPD and therefore its position P_i continuously by odometry (see fig. 2-17). When the robot crosses a grid line (P_r) in the real world (see P_r in the left image), it will choose the intersection position between its PPD and the grid line as its new corrected position (see P_c on the right image). In case of several matching points, the cross-point with the highest PPD distribution wins (see also fig. 2-11 for other examples of intersections between PPDs and grid lines).

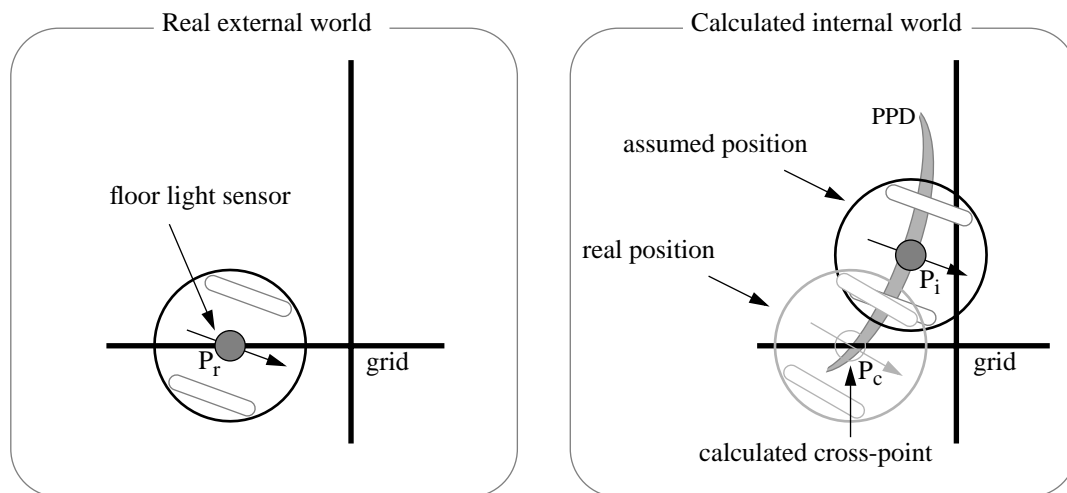


Fig. 2-17: Correction of the assumed position by matching the PPD with the grid

Note: Although the robot on position P_i is much closer to the vertical grid line, the final corrected position will be placed on the horizontal grid line (and moreover far away from the robot). Less carefully computed PPD or an intuitive solution would lead to a wrong position located on the vertical line.

2.5.3 Correction of the angular error

As already mentioned, the angular error is the most significant factor of odometry error. Therefore, correcting the direction is very important. The matching point P_c of the PPD (see fig. 2-17) is not only the new robot's position, but contains also information about the angular error. This means that every point on the PPD depends on a certain wheel speed difference $V_L - V_R$ (see section 2.3 "The PPD model") and therefore on a certain angular drift $d\alpha$.

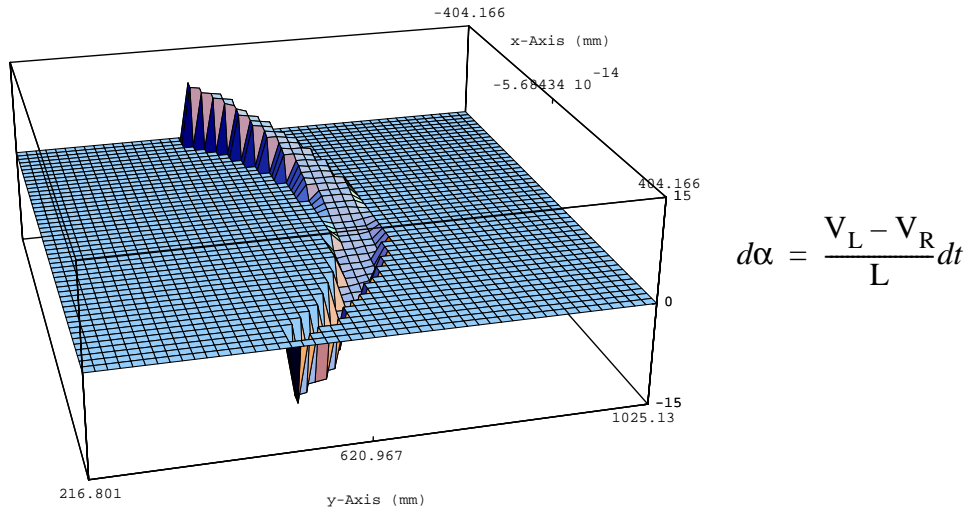


Fig. 2-18: Each point of the PPD indicates the angular drift

Fig. 2-18 shows a plot of the wheel speed difference $V_L - V_R$ in the PPD. An intersection point of the PPD with a grid line supplies the average wheel speed difference and therefore the average angular drift. Condition: Wheelbase L and the traveling time t are known; speed is constant.

2.6 Integration of a path planner “grid fitter”

The classification algorithm has to be backed up by a “path planner” moving the robot (and therefore the sensors) to an optimal position. This biological inspired approach is used by different researchers such as Rudolf Bauer [Bauer R., 1995] or Georg Hartmann [Hartmann G. et al, 1995]. Both papers show, that an adequate path planner can drastically improve the position quality of robots and ants, which are based on odometry.

In this experiment, a “grid fitter” influencing the robot’s planned trajectory is implemented in order to take advantage of the different type of information distributed on the grid lines. The type of information depends on the region, which can be divided into 4 zones:

- V zones supply information about the x position of the robot.
- H zones supply information about the y position of the robot.
- C zones are difficult to interpret and should be therefore avoided by the robot.
- N zones supply no direct information, but are still necessary to distinguish between V and H zones.

It is clear that the robot should pass alternating V and H zones, separated by N zones. At the same time, C zones should be avoided because they provide several cross points with the PPD and the grid lines. The C zone will not be further considered in this paper.

The “grid fitter” change slightly the planned trajectory of the robot. However, the resulting advantage is relevant and will be illustrated by the following example (see fig. 2-19).

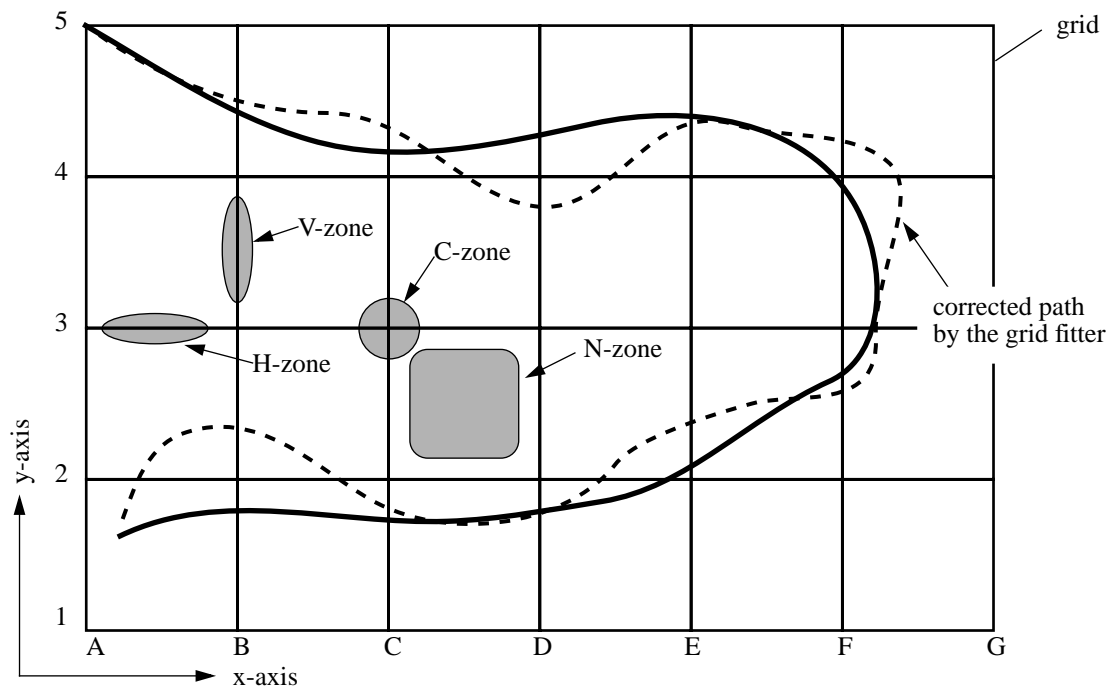


Fig. 2-19: The modified path (dotted line) leads to many more V-H transitions

The robot is asked to follow the solid line, starting at position A5. While driving to the right side (to F4), the robot goes past four V-zones, which allows adjustment of the x position four times. During this time, the y position and the direction are ignored in favor of the freshly calibrated x position. After that, the robot will turn right, but unfortunately going past a C zone containing no useful information. Close to the point F3, the robot crosses finally a H-V zone, supplying significant position information. The way back to the left side of the scene (E2-A1) is similar to the first trajectory segment (A5-F4). The lack of V-H transitions prevents a sufficient odometry correction.

Fig. 2-20 illustrates the events produced by the intended path (upper diagram) and the corrected path (lower diagram). It can be seen that the corrected path contains more useful V-H transitions (compare with (see fig. 2-19)).

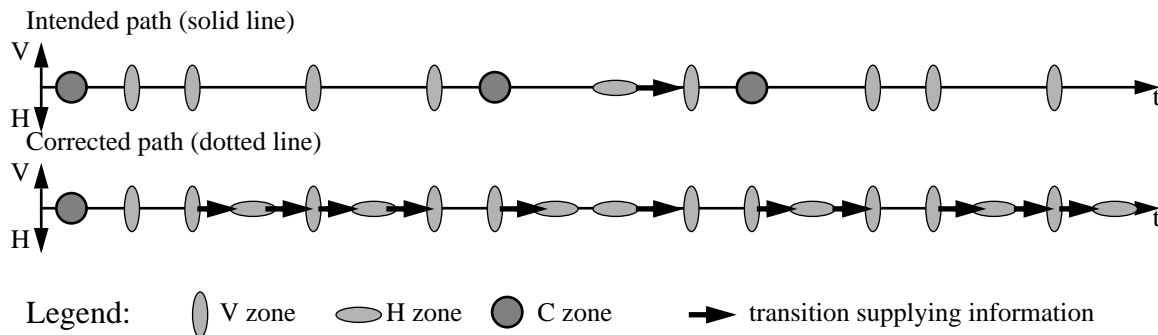


Fig. 2-20: Zone events, caused by the intended and corrected way

2.6.1 Including the grid fitter into a navigation system

The navigator steers the robot to a chosen destination, considering known objects in a map and some unexpected events. The navigator can normally be separated into a path planner and a pilot (see fig. 2-21):

- Path planner (global navigator):
The path planner calculates the path defined by the task describer (a), considering the obstacles indicated in the environment map (b). The path planner will previously calculate the result (c) and hand it over to the pilot step by step (for example left, right, slightly right, stop, ...).
- Pilot (local navigator):
The pilot has no knowledge of the environment, nor of the aim of the task. It reacts to different motion suggestions submitted by the path planner (c), grid fitter (e) and obstacle avoider (d). Each unit supplies a motion suggestion (like right, left, strong right), combined with a mathematical priority (urgent, when convenient, mandatory). The pilot has to select or to combine these suggestions by selection or fusion.

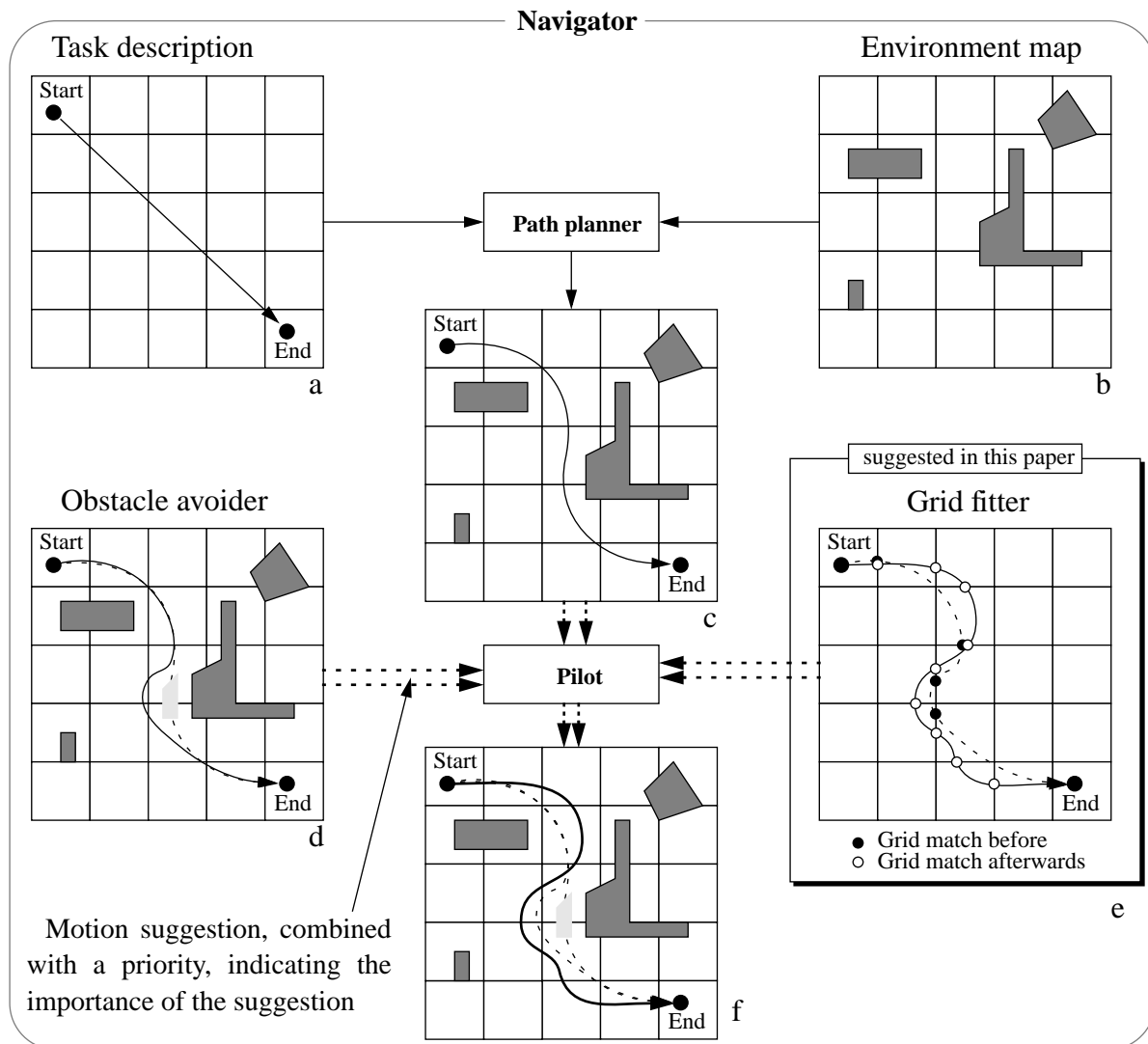


Fig. 2-21: The grid fitter can easily be inserted into a standard path planner

It is important to understand that the grid fitter is only included in the navigator in order to improve the odometry correction algorithm. The illustration (see fig. 2-21) shows that the “grid fitter” module has the same structure as the other modules (path planner and obstacle avoider) influencing the pilot. The integration of the grid fitter into the navigator produces no complications, if the interface of these modules is well defined. A call from the pilot to the grid fitter (and other modules) could be defined as follows:

```
void GridFitter(int actual_position, int actual_direction,
               int actual_speed, int *sug_direction, int *sug_priority);
```

The grid fitter module gets the actual position, direction and speed of the robot from the pilot and returns the suggested direction with a certain priority. Obviously the grid fitter has a local knowledge about the grid.

2.6.2 Grid fitter implemented by potential fields

The grid fitter optimizes the odometry correction algorithm by guiding the robot into interesting V and H zones (see section 2.6). This task can easily be modeled by potential fields and the following rules (see fig. 2-22):

- Single (isolated) grid lines (V and H zones) attract the robot (striped circles).
- Grid cross points (C zone) repel the robot (black circles).
- The robot’s speed and direction are also considered in this potential field. This means that the attractive zones behind the robot have a much lower influence than attractive zones in front of the robot. The robot can be seen as a moving mass with a certain speed and direction. Of course, the potential field does not change the absolute speed of the robot, which would be the case for a moving mass.
- V and H zones have to be crossed over alternately, thus when the robot has crossed some vertical lines, it has to be attracted in a stronger way by the horizontal ones.

The arrows in fig. 2-22 indicate the attractive and repulsive power of the zones.

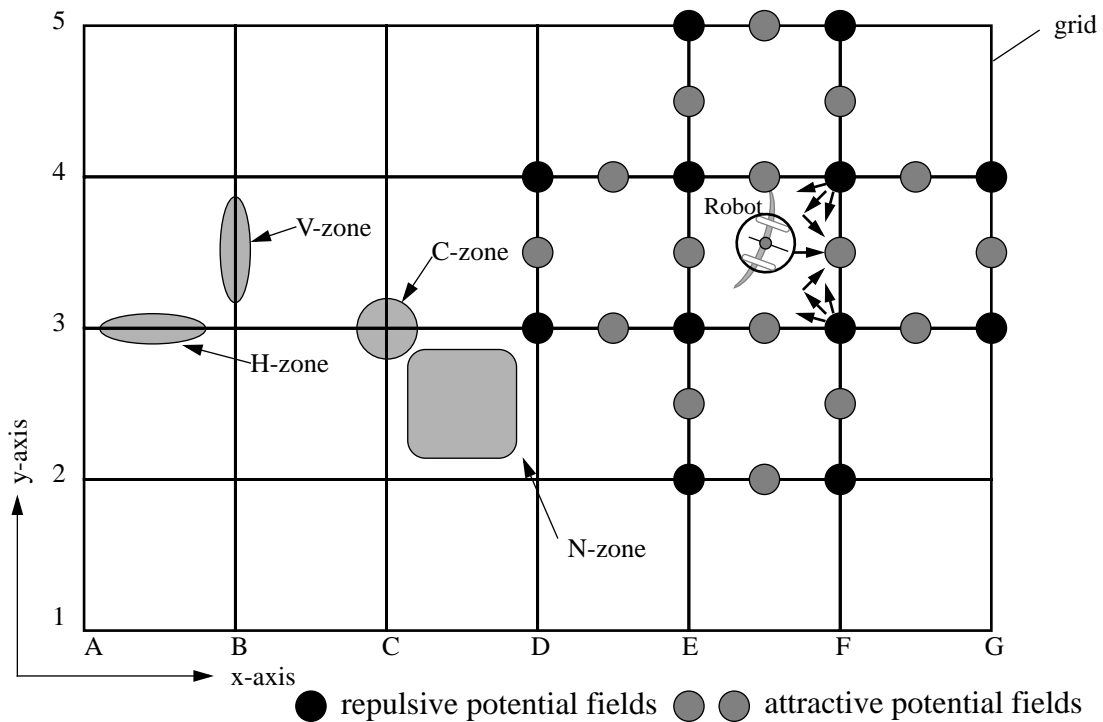


Fig. 2-22: Potential fields guide the robot over V and H zones

2.6.3 Other tasks of the grid fitter

The robot is not simply attracted and repelled by the potential fields. In order to improve the correction of the angular drift, the PPD should have a certain angle to the grid line (horizontal and vertical) while crossing it. This situation improves the recognition of angular errors. The following illustration (see fig. 2-23) shows the difference, if the PPD is crossing in a parallel way the grid line (left image) or with a certain angle (right image).

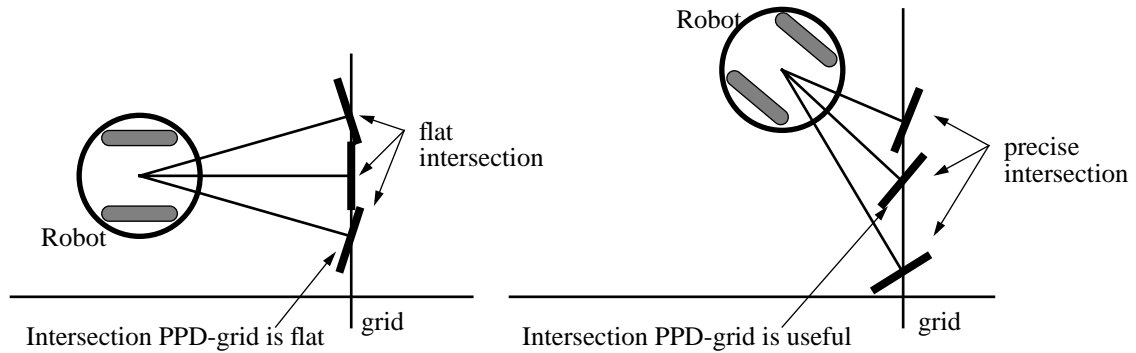


Fig. 2-23: Diagonal grid line crossing improves the recognition of angular errors

The left image shows the PPD going past the grid line at right angles. The intersection (grid line and PPD) is flat and supplies no exact cross point. In the right image, the intersection supplies an exact cross point, because the PPD has a certain angle to the grid line.

Please note that the orientation of the PPD is important and not the orientation of the robot. In certain circumstances, the PPD could have a good angle to the grid line caused by its “inertia”, even if the robot crosses it at right angles (see fig. 2-24).

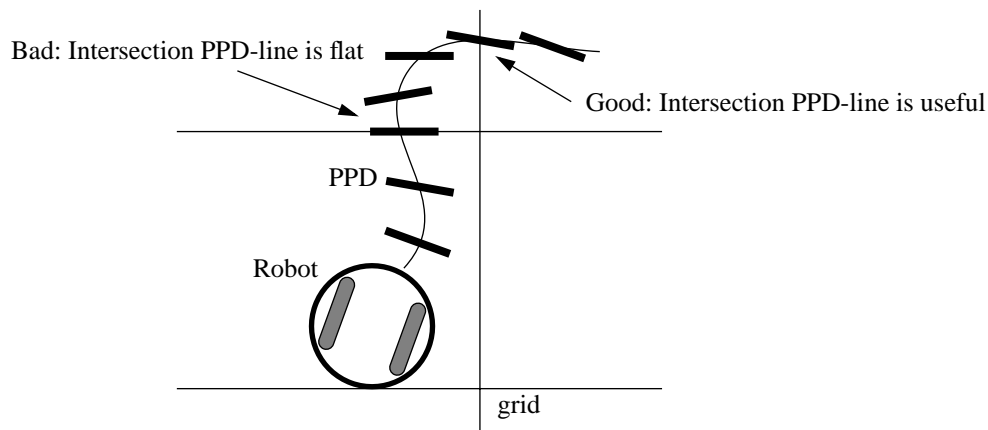


Fig. 2-24: Angle between PPD and grid line is important (not PPD and robot path)

Tasks of the grid fitter:

- Avoiding bad and going past good zones (see section 2.6.2 "Grid fitter implemented by potential fields").
- Crossing grid lines with a certain angle in order to improve the quality of the intersection.
- Minimizing the size of the PPD by adequate motion maneuvers.

2.6.4 Practical tests

The experiment was done by a real Khepera robot which calculated simultaneously its position by a corrected and uncorrected odometry algorithm. Fig. 2-25 shows the result of the uncorrected (left image) and the corrected (right image) calculations which was performed during the same experiment-run.

- Without any correction:** The left image shows the robot starting on a known position & direction and performing a trajectory indicated by small squares \square . The actual position of the robot is based on pure odometry. Because of the odometry error, the robot starts to drift to the left side, which is indicated by the solid line (calculated trajectory). Each detection of a grid line is marked by a cross \times on the calculated trajectory. It can be seen that the distance between the crosses and the corresponding squares (which indicate the real cross points) increases continuously. The final error is about half a square unit, which means about 5 cm (one square has a length of 10 cm).
- With correction:** The right image is the result of the same experiment as described before, but it shows the result of the corrected odometry calculation. Therefore the solid line is replaced by a trace of several small “bananas” which indicates the PPD. The size is constantly growing due to the accumulated odometry error. Each time a grid line is detected by the light sensor, the PPD is cut by the closest grid lines. The intersection is marked by a cross \times , which is obviously always on a grid line. The corresponding real position is shown by little squares \square (identical to the left image). The assumed robot position is corrected towards the intersection marked by a cross \times . At the same time, the PPD size can be reduced, because of the known odometry error. The average size of the PPD stays constant and is an indicator for the reliability of the robot position.

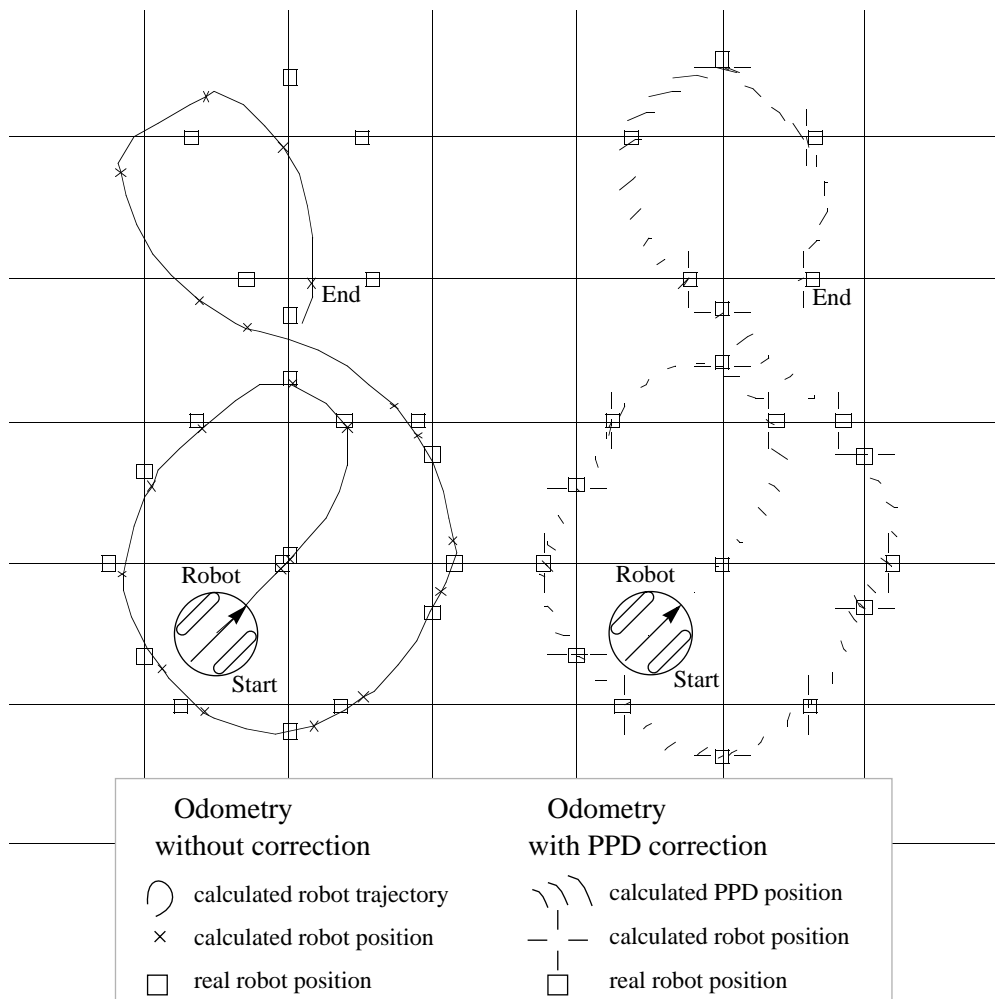


Fig. 2-25: Experiment without and with odometry correction

2.7 Conclusion

The presented algorithm allows a two wheeled robots, equipped with a single light sensor underneath, to efficiently correct their odometry by detecting grid lines on the floor. Some simplifications¹ make the algorithm light enough to run local on a Khepera robot equipped with a Motorola 68331 Microprocessor with 128K Ram and 256K Rom. The grid could also be represented by grooves in the ground. Robots can thus navigate with minimum external help depending on the intended trajectory.

This algorithm demonstrates also an optimal exploitation of available information by using a new model of position probability distribution (PPD) and an interacting pilot (Grid Fitter). The examples show that the direction of the PPD is not always at right angles to the robot trajectory. The direction follows slowly the robot's direction and can have a completely unexpected shape after a narrow curve. This effect could be interesting in the future in order to shrink the size of the PPD as a function of the trajectory actually effected (see section 2.3.6 "Examples"). This effect is similar to the re-concentration of water waves after reflection by a concave object. However, the position distribution probability can shrink but not the direction distribution, which makes a reasonable use of this special point questionable.

The experiment showed that the system is stable enough to miss three or more grid lines in case of sensor problems or other unfavorable conditions. However, the system will rapidly lose its position after passing the maximal odometry error due to misinterpretation of the following grid signals and the resulting position corrections in the wrong direction. Once in this state, there is no way back to synchronize the robot with its real position.

1. Because of the low CPU power of the Khepera robot (68311 @ 16MHz), neither the shape of the PPD nor the 3D cut with the grid lines could be calculated in real time. Therefore, the PPD was approached by several sample points representing the PPD. These sample points can be updated 10 times per second which allows to perform real time applications. The expenditure of intersection calculation can be ignored.

3 Unsupervised Passive Positioning System (UPPS)

In the former chapter, the robot used the grid dimension and its initial position. The algorithm presented now allows a robot to recognize its position and to navigate without prior information regarding its environment. What is going to be used as a landmark is initially unknown and practically no specific preprocessing of the sensor signals is done. The robot nevertheless extracts meaningful information from its environment and uses it to build a map. Both landmark definitions and the map are continuously adapted. It can be shown that navigation without predefined categorization takes better advantage of the sensor abilities and extracts more information from the environment.



Fig. 3-1: Robot exploring an unknown environment¹

The result shows that a robot can reliably recognize and rediscover self-defined landmarks suitable for its sensor capability as well as create a map of its environment.

1. Cartoon by Isabelle Follath, Zürich, source [Pfeifer R., 1996] permitted by Prof. R. Pfeifer

3.1 Unsupervised classification approaches

This section shows some general methods of unsupervised classification. Unsupervised classification is a huge field and can be grouped into three approaches. The main differences between these approaches are explained in the following paragraphs.

3.1.1 Statistical approach

Statistical approaches are generally based on a probability model. These models are usually invented or constructed by statisticians who are aware of the overall problem and their special attributes. Statistical approaches provide therefore a *probability* of being in certain predetermined classes and is unable to create or change these classes by itself.

3.1.2 Machine learning approach

Machine learning is difficult to define. It generally encompasses automatic computing procedures that learn a task from a series of examples by classifying and selecting significant parts of the example.

The *decision-tree approach* which can be created by *Q-learning* is a typical example of machine learning techniques. The classification is guided by proceeding a given sequence of logical steps, encompassing the complexity of the problem. This technique allows to represent and to classify complex problems.

*Genetic algorithm*¹ is one of the most examined techniques in machine learning. It allows to deal with more general types of data. Their combination can be very different. However the evaluation of all these combinations demand a lot of CPU power.

Similar to the statistical approach, machine learning techniques demand generally a certain background of human knowledge to construct the algorithm. However, the operation is assumed without any human intervention.

3.1.3 Neural network approach

Classification can also be done other than by technical and mathematical frameworks. Since classification is an essential feature of natural intelligent systems, it is interesting to study how nature solves this problem and tries to copy the solution. The brain owes its capability to about 10^{11} neurons. Each of them is connected to about 10^4 other neurons, constituting an enormous “computer power”.

1. Genetic Algorithm is an evolutionary algorithm which generates several individuals from some encoded forms known as a “chromosomes” or “genomes”. Chromosomes are combined or mutated to breed new individuals. An offspring's chromosome is created by joining segments chosen alternately from each of two parents' chromosomes which are of fixed length.
GAs are useful for multidimensional optimization problems in which the chromosome can encode the values for the different variables being optimized.

The following rather simple mathematical description is widely used in the domain of computer science to model the function of a neuron. It consists of a number of input nodes x_i and one output node s (see fig. 3-2). Each input x_i is multiplied by w_i and added up with the other inputs. The resulting sum ρ is normalized by a transfer function δ . The exact nature of δ will depend on the neural network model under study. A few examples of the most frequently used transfer functions are shown below:

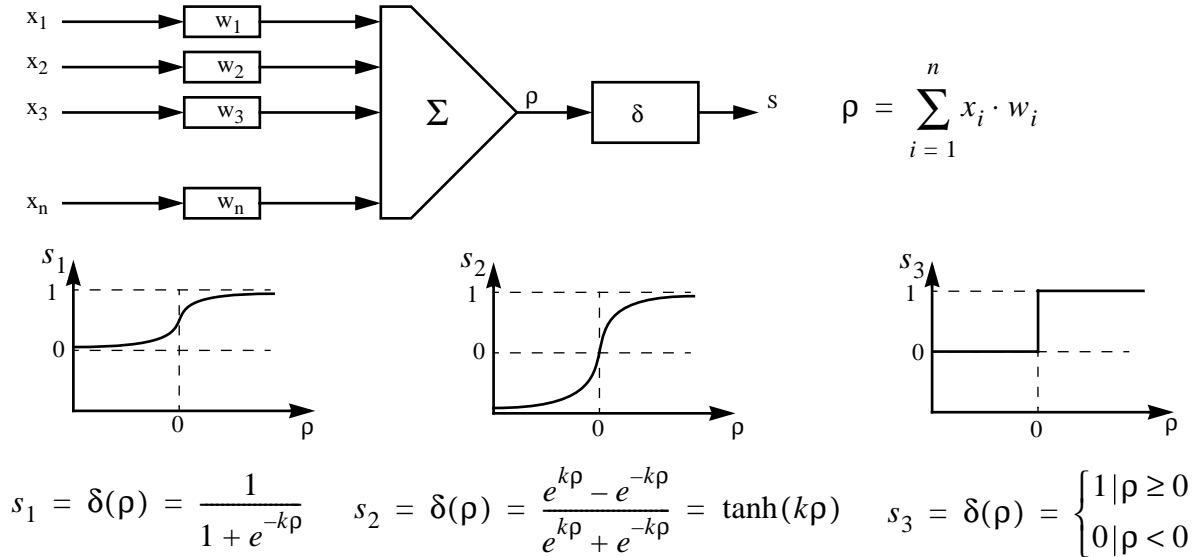


Fig. 3-2: Three different representations of an artificial neuron

An artificial neural network consists of several layers of interconnected neurons. Each neuron produces a nonlinear function of its input, which may come from another neuron or from the main input. This construction incorporates any degree of non-linearity, allowing very general functions to be modeled.

The neural network approach combines the complexity of some statistical techniques with the machine learning strategy of imitating human intelligence. However, this is done on a more “unconscious” level. The neural network is able to create and adapt autonomous classification rules, but makes a learned concept non-transparent to the user.

3.2 Experimental Set-up

This chapter gives an overview of the aim and algorithm used to carry out this experiment.

3.2.1 Aim

A robot moves randomly in an unknown static environment (see fig. 3-3) and is repelled by obstacles. It records information from its sensors in order to recognize significant signals as landmarks. The robot has no prior knowledge about the environment nor of the type or character of its own sensors. The robot should learn to recognize any significant signals as landmarks, store them into a memory as well as compare them with new landmarks in order to rediscover familiar situations and positions.

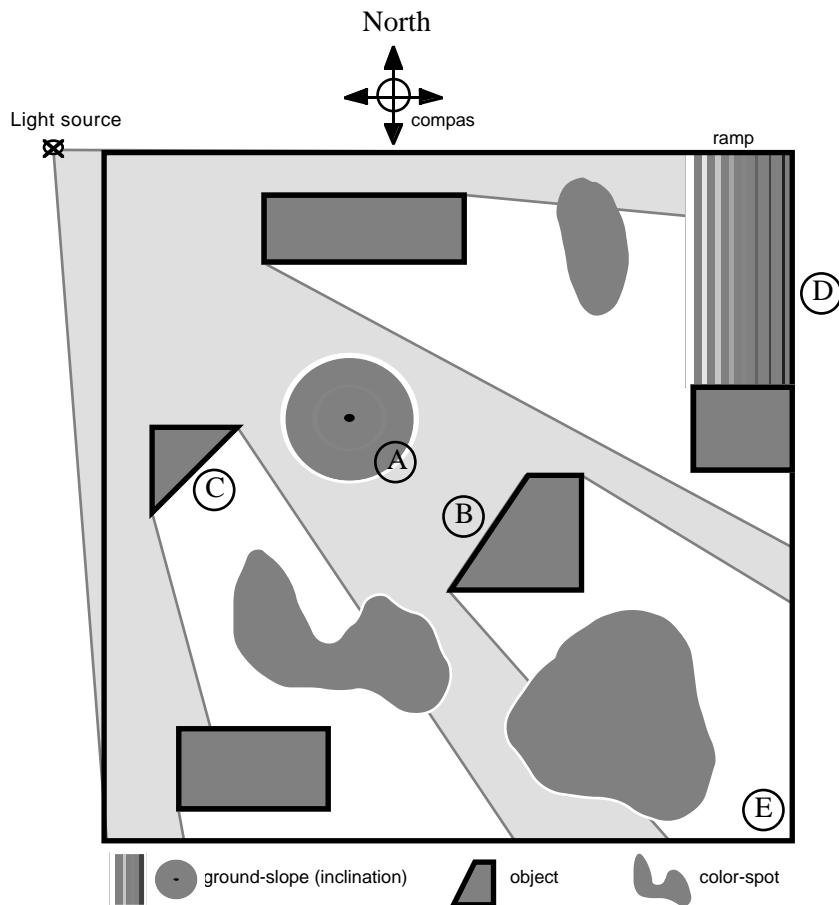


Fig. 3-3: Standard environment offers several stimuli

Fig. 3-3 illustrates a robot environment without prior defined landmarks, containing a lot of obstacles (even similar ones), several color spots, ramps etc. Recognizing these stimuli with a pre-programmed algorithm would not be very efficient. However, the environment offers a lot of unique characteristics, which are not obvious to recognize:

- Point A shows the only slope on which the robot climbs in directing himself toward the light source.
- Point B is the only diagonal wall illuminated by the light source.
- Point C shows the only diagonal (compass sensor) wall which is not illuminated by the light source.
- Point D shows the only wall which the robot has to climb up.
- Point E is the only corner which is not illuminated by the light source.

There are a lot more (less obvious) landmarks which can be extracted from this robot's standard environment. The example also shows that it would be difficult to preprogram these stimuli as landmarks because:

- They are not always foreseeable.
- They are too varied (special processing for each landmark).
- They could change with time (in a slow manner, no dynamic environment).

The aim of this experiment is to show that unforeseen stimuli can be linked with the robot's position and can therefore be used as a landmark in order to stabilize the odometry error. This allows restriction of the odometry error to a certain limit. The robot can also use its own self-discovered landmarks to create a cognitive¹ map, allowing it to navigate in an unknown environment.

3.2.2 Experience set-up

I used a Khepera simulator programmed by Olivier Michel [Michel O., 1996] to test the algorithm. The simulated robot is equipped with eight short distance sensors (about 5 cm range), eight ambient light sensors, an odometry module calculating its estimated position, a compass and a linear camera which reads a horizontal line of 64 pixels with a view angle of 36 degree in front of the robot. This can be used to recognize obstacles and wall contours in front of the robot. The environment consists of a field of about 15 x 15 x diameter of the robot.

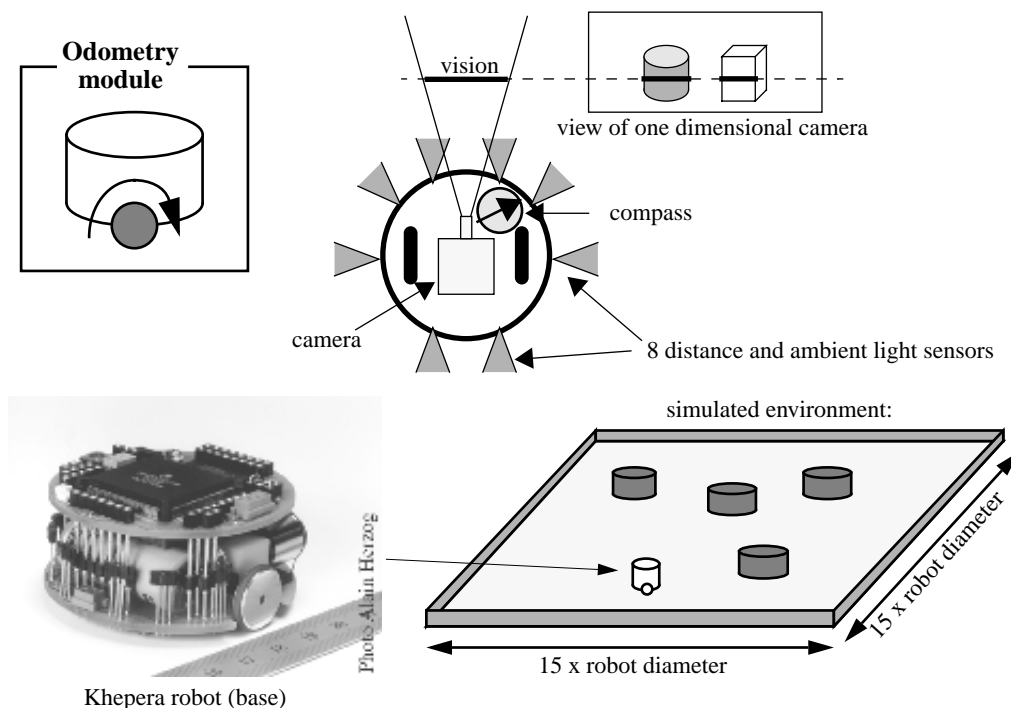


Fig. 3-4: Equipment of the extended Khepera robot

The simulator was written in C and runs on a SUN Ultra 1 (143 MHz) allowing the robot to move twice as fast as in reality.

1. A cognitive map can be defined as the internal representation of the geometric relations among noticeable points in the animal's environment. In operational terms, an animal using such a map must be able to determine its position relative to home, or any other charted point, even when it has been displaced inadvertently to an arbitrary location within its environment. Quoted from [Wehner R., 1992].

3.2.3 Algorithm overview

The signals coming from different sensors are partially mixed together and directed into several unsupervised classifiers creating clusters (see chapter 3.4 "Classification", page 45). Each classifier is based on a different type of neural network in order to take advantage of each specific characteristic. The classes thus created are analyzed by the statistics module, which selects "usable" classes and stores them into a memory including its estimated robot position. These entries are compared with new incoming classes. In case of a sufficient match, the robot position will be corrected to the position previously stored with the class. Fig. 3-5 shows the described algorithm.

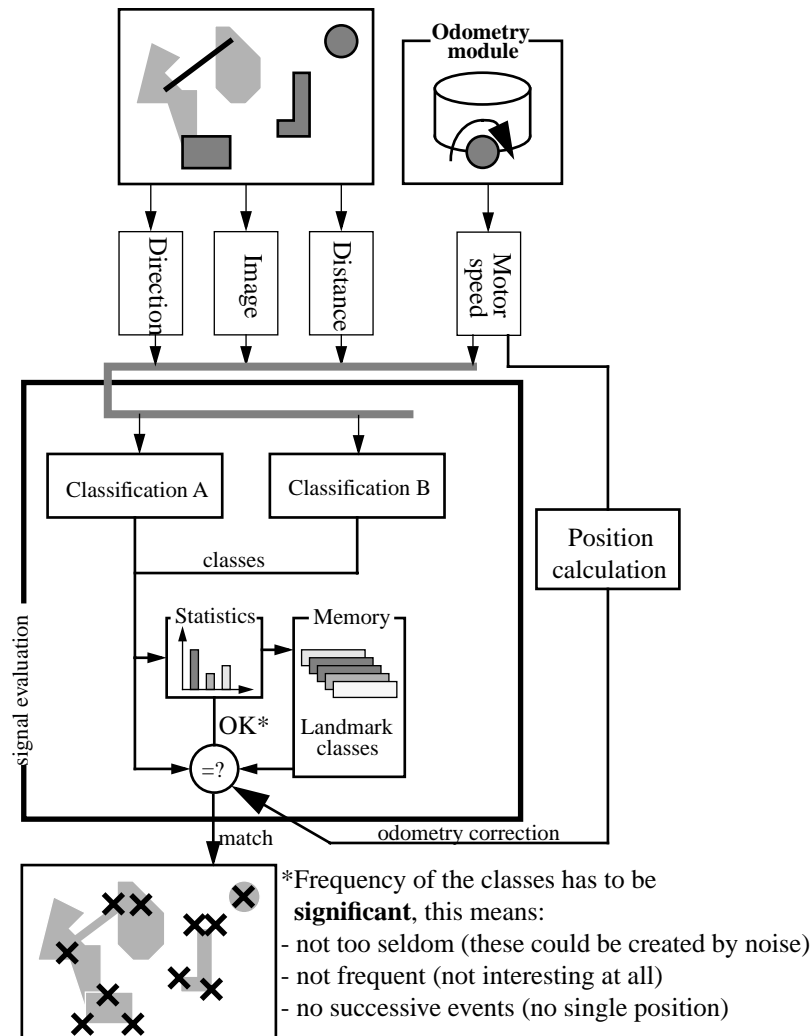


Fig. 3-5: Sensor signal data processing and recognition of significant features

The statistical module analyzes the incoming classes based on its frequency and search for significant classes which can be used as a reference point. Therefore, the activation of such **significant classes** has to be:

- not too seldom (because it might represent noise)
- not frequent (it might represent different positions which is not interesting at all)
- not activated successively (because it doesn't represent a single point)

This is inspired by Nake [Nake F., 1974] and others [Schmidhuber J., 1997] who suggest that most interesting data exhibits an ideal ratio between expected and unexpected information. The numerical limits for these three criteria were found empirically.

3.3 Normalization

The aim of this project is to use raw sensor signals and to avoid task specific signal preprocessing as much as possible. Nevertheless, minimal noise filtering and signal normalization is difficult to avoid and is presented in this section. The reason for the normalization is not only to simplify the experiment but also to adapt the signals to the characteristics of the used classifiers. Even the application of unsupervised neural network contains some limitations, i.e. the classification of several sinusoidal sensor signals of different frequencies cannot be done without a Fourier preprocessing.

3.3.1 Distance and ambient light sensors

Because of the poor quality of the distance and ambient light sensors, a simple two state value is used to distinguish distance from obstacles or bright from dark regions. A threshold value of 300 (corresponding to an object distance of 2.5 cm) was chosen.

3.3.2 Camera preprocessing

The image of the camera was simulated and calculated by a pseudo reflectance image model [Song H., 1996], assuming a diffuse light source and a Lambertian object's surface (matte surface, no texture).

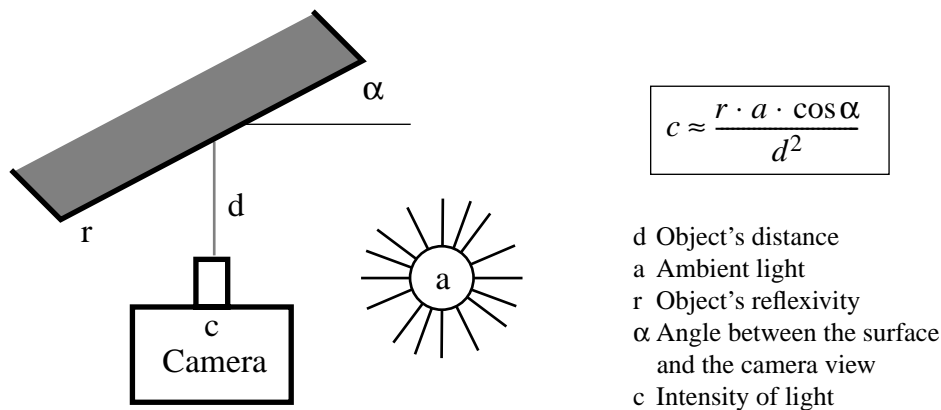


Fig. 3-6: Calculation of light intensity, first approximation

The measured light intensity c can be approximated by the equation in fig. 3-6, using as variables: The distance of d , the ambient light a , the reflexivity of object r and the angle between the surface and the camera view α . Each object transition causes a strong change in d and α , which is very suitable for edge detection.

The edge extracting process is a little unusual and therefore further explained in fig. 3-7. The environment (point A) is taken by a camera (point B) which produces according to the Lambertian theory the image shown in point C. Edge detection is done by the 2nd derivative of the normalized, discrete camera image (showed in point D). After that, the image is stretched until the far left edge reaches the left border and the far right edge reaches the right border (point E). This conversion makes the image stable against rotation and any kind of movement, as long as the same obstacles (contours) are visible to the camera.

The resulting image is preprocessed in two different ways in order to feed them into two separate neural networks (NN):

- In the *Gaussian neuron distribution*, neurons are stimulated according to the position of the edges in image E. The neighboring neurons are stimulated as well but less, according to a gauss distribution. This makes the network less sensitive to minor position changes of an edge in the image E.
- In the *group neuron distribution*, neurons are arranged into groups according to how many edges are seen in image E. In our example, five edges can be seen in image E, though the far left and the far right ones are not interesting because of their fix positions. So we get 3 edges which activate group 3. The individual activation of these 3 neurons correspond to the distance between the edges and the left border (point F). All other neurons of the other groups are not activated. This kind of preprocessing makes the second NN very sensitive to a change in the number of edges, which is often ignored by the first NN, especially if the edges are close to each other.

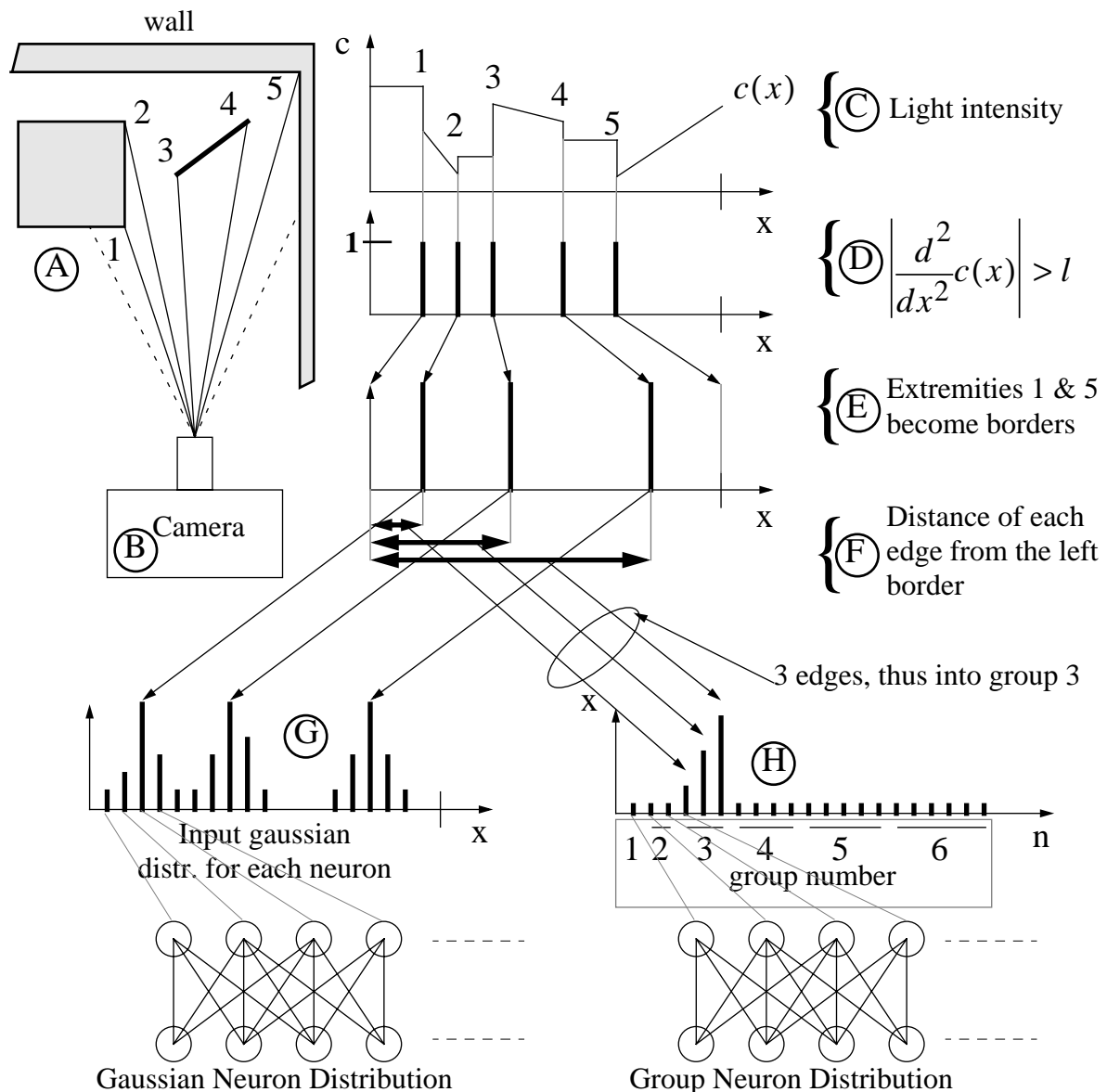


Fig. 3-7: Two different images preprocessing to improve classification

Fig. 3-8 shows a screen shot of the same “image processing” performed by the simulator. The on-board camera K213¹ simulated on the robot “sees” four obstacles which produce the image and the double derivation shown in the diagram on fig. 3-8. The image is stretched until detected corners (double derivation) reach the borders and “smoothed” by a Gaussian distribution in order to reduce the reaction of the network by small lateral object shift. This operation makes the relative position of the inner two objects to the outer objects decisive, instead of the whole image. The distance of the inner two objects from the left border is used to activate group number two of the second neural network. Each group is activated depending on the number of detected inner objects. This preprocessing makes the network sensitive to emerging and disappearing objects.

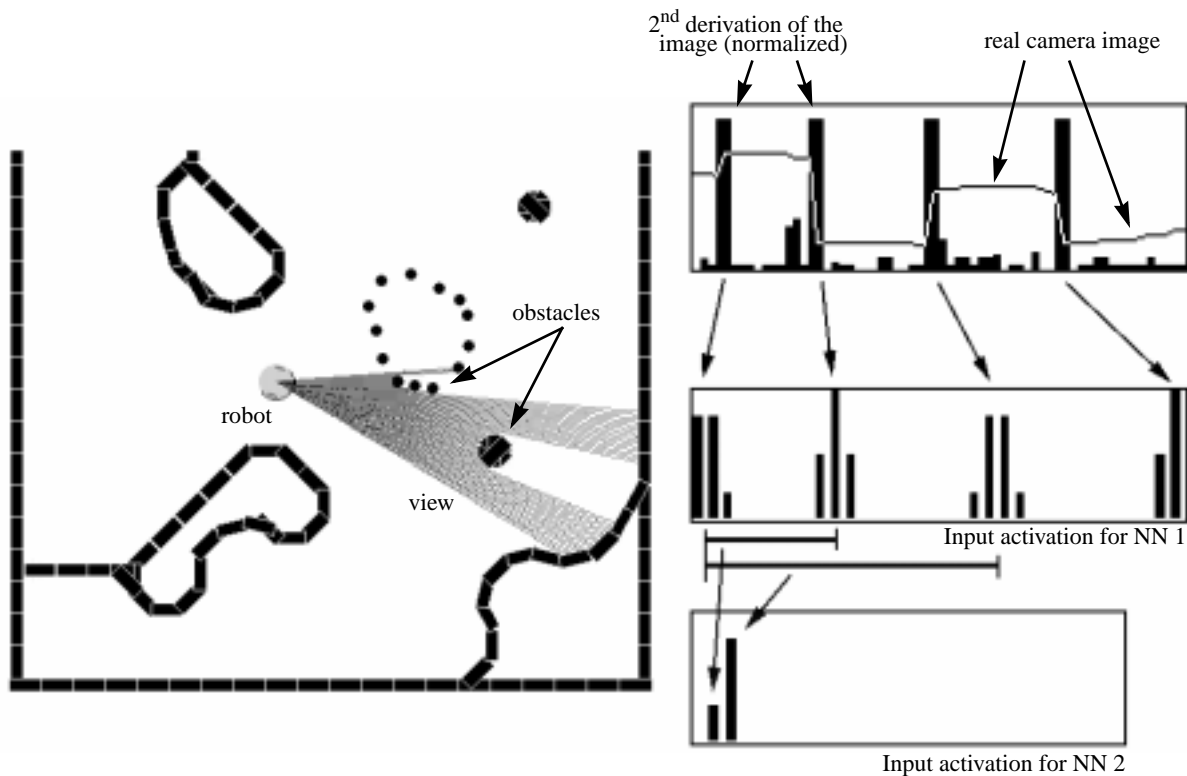


Fig. 3-8: Screen-shot of the camera image normalization

Fig. 3-9 illustrates the characteristics of these two kinds of preprocessing and why they are used together.

- The first situation in fig. 3-9 shows a Khepera robot approaching 3 obstacles. Two obstacles are one behind the other and are therefore recognized as one obstacle by the *group neuron distribution* preprocessing. Therefore group 2 is activated. While approaching the obstacles, the robot changes its angle of vision and suddenly distinguishes all three obstacles, which causes the activation of group 3 in the *group neuron distribution* preprocessing. The same striking event is hardly recognized by the *gaussian neuron distribution's* preprocessing.
- The second situation in fig. 3-9 shows the opposite case. The *group neuron distribution* preprocessing always recognizes three obstacles. On approaching the obstacles, the angle of vision changes and therefore the activation of the neurons in *group neuron distribution* preprocessing also changes. However, this change is very small compared to the change in the *gaussian neuron distribution*.

1. K213 is a vision turret produced by K-Team (<http://diwww.epfl.ch/lami/robots/K-family/K213.html>)

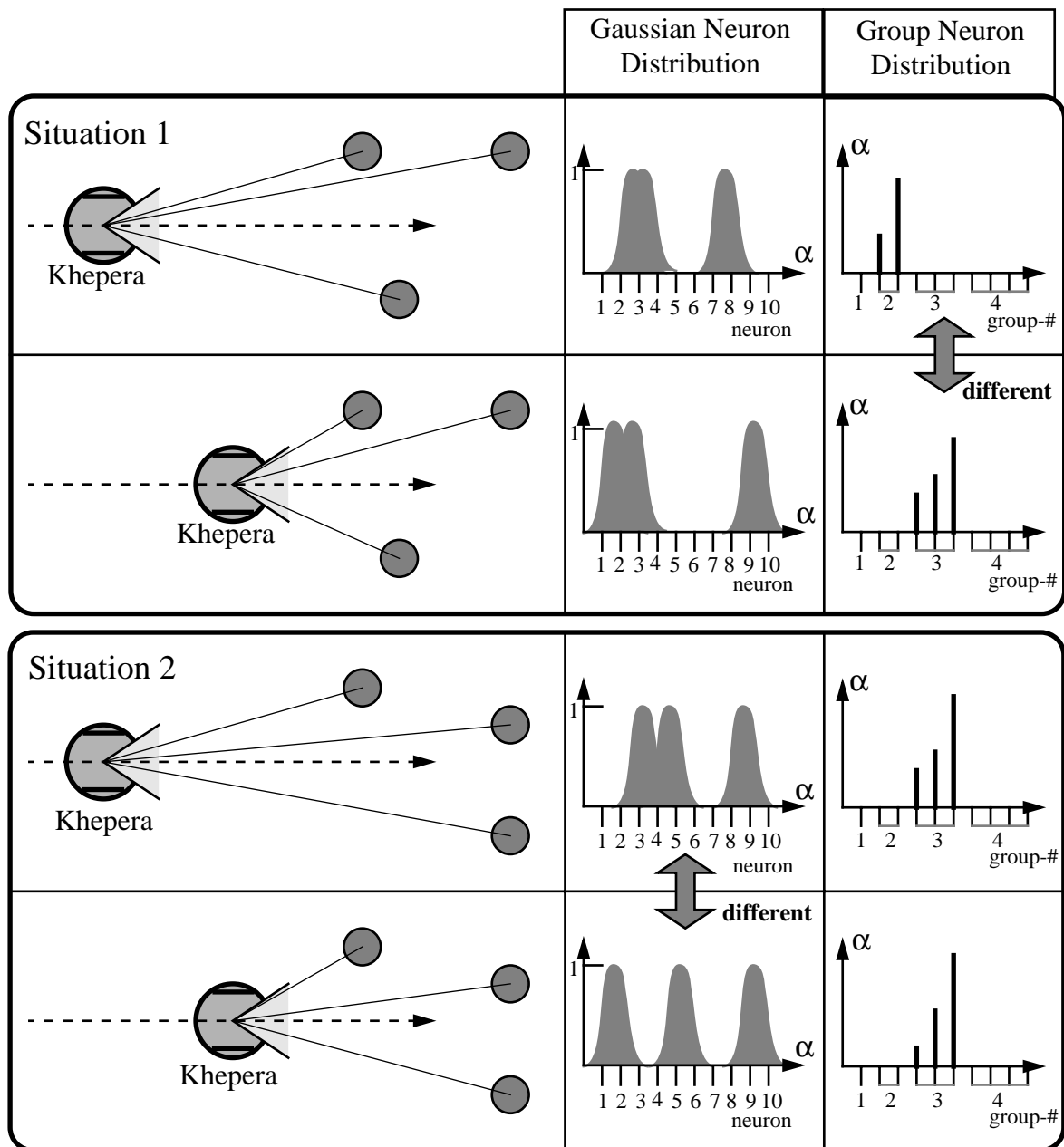


Fig. 3-9: Effect of camera processing

3.4 Classification

This section explains the need to classify the robot sensor information and how this can be done. After that, the methods applied are described and how they are modified to solve the problem.

The principal idea is to merge all raw sensor signals together and to treat them in the same way independently of the source. A self-organizing process clusters the signals in classes which correspond to the environmental situations. However, two exceptions have to be considered:

- Analog and digital signals should be treated differently in order to take advantage of the classifier characteristics. Digital signals always use the full input range from zero to one which is not the case for analog signals. Strong range differences decrease the efficiency of neural networks.
- The raw camera image is filtered and processed by an edge detection algorithm before further use (see chapter 3.3 "Normalization", page 41).

Robot sensors supply continuously various informations like object distance, ambient light, compass direction as well as some image information. All this information can be represented by a single sample point $S(t)$ in an n -dimensional input space. The dimension of the input space depends on the sensor type. A distance sensor usually provides only a one-dimensional number, which is the distance. On the other hand, a color sensor supplies normally a three dimensional value (red, green, blue).

Fig 3-10 shows an example of a two dimensional input space created by a distance sensor and a compass (both supply one dimensional information).

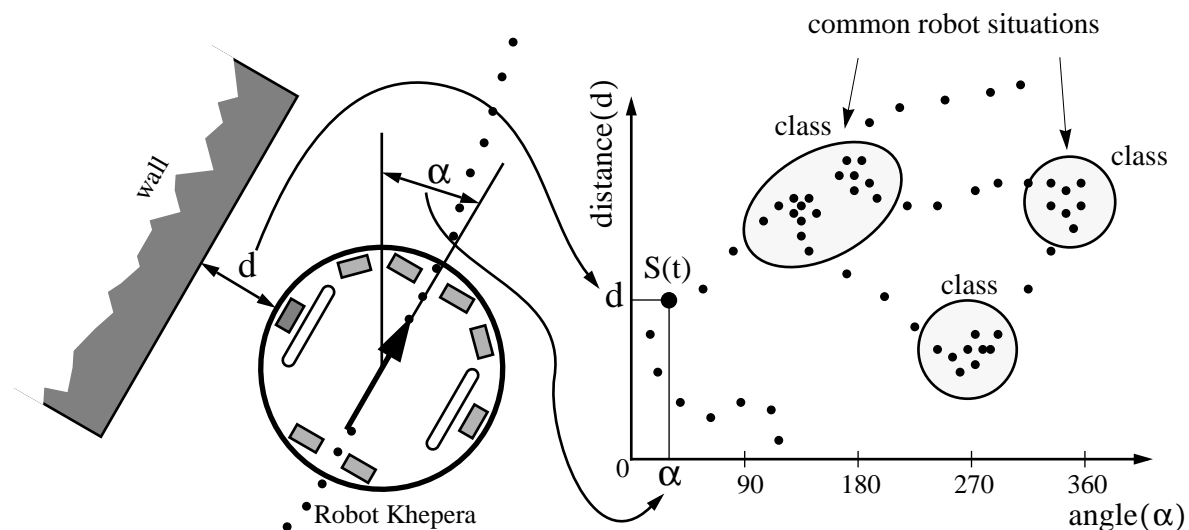


Fig. 3-10: 2-D robot sensor space defined by a distance sensor and a compass

The n -dimensional sensor input space is filled up with sample points $S(t)$ during a run. Storing all these sample points would be useless and would overflow the memory. It is much more efficient to organize accumulations of sample points into *classes* which will then be used for further treatment.

The process of putting points into a common group is called “clustering” or “unsupervised classification”. The accumulation of sample points corresponds to a robot situation which is encountered frequently i.e. following a straight wall with a constant distance. Such a cluster of sample points can be “bound together” and used as a reference for further use without even knowing what it “means”.

3.4.1 The Growing K-means Algorithm

The K-means algorithm can be implemented by a competitive network. The output neurons, of which the weight vector is closest to the correct pattern vector, becomes the winner neuron. This is reminiscent of the K-means algorithm, in which the cluster center with the shortest Euclidean norm distance to the input pattern vector, acquires the pattern and earns the right to respond to that pattern.

As previously mentioned, a pattern vector of n dimensions may be considered as representing a point within an n -dimensional Euclidean space. The K-means algorithm identifies vectors which are geometrically close together based on the assumption that geometrically close points belong together.

Before presenting details of the standard K-means algorithm, a more precise notion of the metrical distance is needed. The Euclidean norm of a vector $x=[x_1, x_2, x_3, \dots, x_n]$ is defined as shown in equation 3-1:

$$\|x\| = \sqrt{\sum_{i=1}^n x_i^2} \quad (\text{Eq. 3-1})$$

Equation 3-1 represents the length of the vector x . As we wish to discover the distance or length between two vectors within the pattern space, we need only to apply the equation 3-1 to the vector difference x and z of the same order n as shown in equation 3-2:

$$\|x - z\| = \sqrt{\sum_{i=1}^n (x_i - z_i)^2} \quad (\text{Eq. 3-2})$$

After defining the distance between pattern vectors, we need a procedure that will establish a set of clusters (with associated cluster centers); for example the distance between an input vector and the closest cluster center serves to classify the vector.

The K-means makes the assumption that the number of cluster centers is known *a priori*. The version implemented in the experiment ignores this assumption which allows to trim the neural network regarding to the optimal distance between the cluster centers (vigilance) and not to the maximal number of cluster centers. On the other hand, this makes the system sensitive to the temporal order in which input data are presented.

Description of the algorithm:

Before describing the K-means algorithm, some nomenclature must be established. We follow the notation in [Pandya A.S. et al, 1996]. Let $x^{(p)}$ represent the p^{th} input vector. The complete set of input vectors will then be $\{x^{(1)}, x^{(2)}, \dots, x^{(p)}\}$. The cluster center for each of the K clusters will be represented by z_1, z_2, \dots, z_k . The notation $S_j = \{x | x \text{ is closest to cluster } j\}$ will be used to represent the set of samples that belong to the j^{th} cluster center. N_j indicates how many samples belong to the cluster j . The K-means algorithm is implemented in the following steps:

K-means (batch version)

1. Initialize:

Choose the number of clusters. For each cluster i pitch an initial vector W_i . This initial vector can be arbitrary or simply the first K input vectors.

$$\{W_1, W_2, \dots, W_K\} = \text{random or first } K \text{ input vectors} \quad (\text{Eq. 3-3})$$

2. Classification by closest prototype:

Search for S_j to which $x^{(p)}$ belongs. Each sample vector $x^{(p)}$ is attached to one of the K clusters according to the following criteria:

$$x^{(p)} \in S_j \text{ if } \|x^{(p)} - W_j\| < \|x^{(p)} - W_i\| \text{ for all } i = 1, 2, \dots, K, i \neq j \quad (\text{Eq. 3-4})$$

3. Move prototype center to cluster:

After allocating all member vectors to their clusters, the centers have to be adapted such that the sum of the distances from each member vector to the new cluster center is minimized. The equation 3-5 calculates the new cluster center of the cluster j containing N_j members.

$$z_j = \frac{1}{N_j} \sum_{x^{(p)} \in S_j} x^{(p)} \quad (\text{Eq. 3-5})$$

4. Checking for convergence:

The algorithm jumps back to step 2 if the prototype is still moving after one cycle. The neural network has learned the example if the prototype does not move anymore.

The K-means algorithm can be implemented in a two layer neural network. Each output neuron defines a class j which is described by the vector W_j . Fig. 3-11 shows the simple network structure:

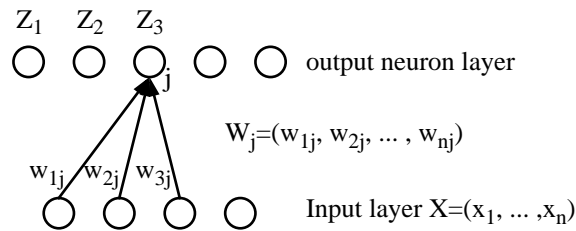


Fig. 3-11: Network structure of the K-means algorithm

Growing K-means

The *Growing K-means* classifier that I have used does not exist in literature. We extended the standard K-means classifier with the ability to acquire new output neurons if the existing ones are not sufficient close to the new input vector. The name was inspired by the *Growing Neural Gas* theory [Fritzke B., 1997] which has the same ability as well.

3.4.2 Adaptive Resonance Theory (ART)

The ART network has been introduced by Carpenter & Grossberg in 1987 [Carpenter et al, 1987a]. It is, broadly speaking, an unsupervised clustering network where the cluster size can be controlled and the cluster formation can continue during the network's operation. The input pattern can be binary (ART-1) or analog (ART-2). The input neurons have no vicinity relationship. This means that the activation of adjacent inputs has the same importance as the activation of inputs far away from each other. More information can be found in [Carpenter et al, 1987b], [Grossberg S., 1988], [Pandya A.S. et al, 1996], [Hertz J. et al, 1991], [Fausett L., 1994] and [Freriks L.W. et al, 1992].

ART networks attempt to address the *stability-plasticity dilemma*. Grossberg describes the stability-plasticity dilemma as follows:

How can a learning system be designed to remain plastic, or adaptive, in response to significant events and yet remain stable in response to irrelevant events? How does the system know how to switch between its stable and its plastic modes to achieve stability without chaos? In particular how can it preserve its previously learned knowledge while continuing to learn new things? What prevents the new learning from washing away the memories of prior learning?

So ART networks contain a mechanism which allows new patterns to be learned without forgetting or degrading old knowledge. This ability is very useful for robot application because the environment of robots (representing the input of the ART) cannot be registered and learned in a regular and repeated manner. The sensor information depends on the robot's position, which is random or even locally repetitive if the robot stays in the same spot. Therefore old knowledge has to be preserved from newer ones, even if the older knowledge has not been confirmed for a long time.

The degree of pattern similarity in a cluster and therefore also the number of clusters can be controlled by a *vigilance*¹ test. A new cluster (neuron) will be created if a new input pattern fails the *vigilance* test with the existing clusters. This helps to overcome the *stability-plasticity dilemma* (which is the ability to constantly learn new experiences without ignoring old ones.)

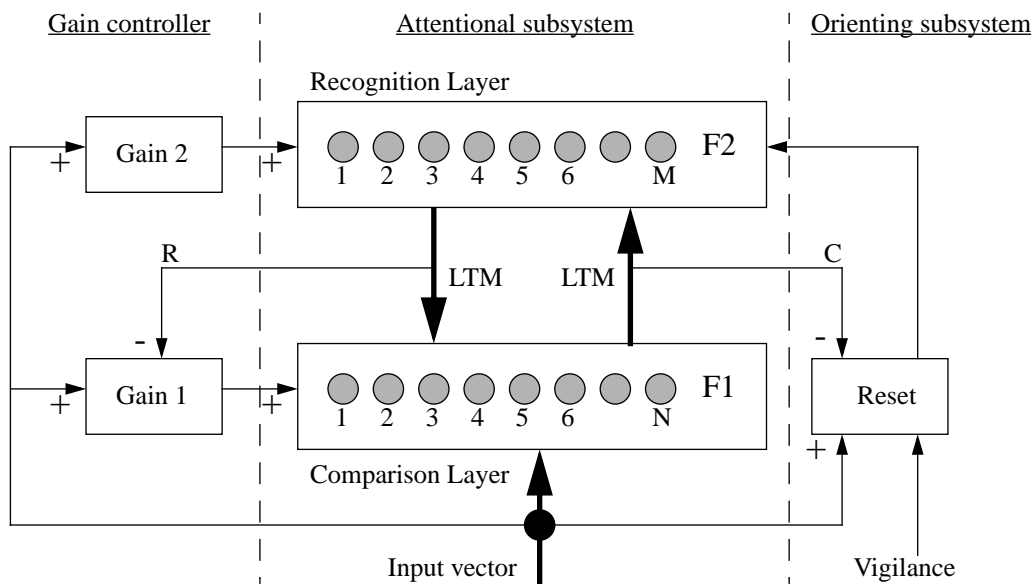


Fig. 3-12: General ART architecture

1. The vigilance is a common parameter in ART networks which describes the minimal distance between cluster centers. It can be changed during an experience (on-line).

The ART architecture got its name because of the information oscillating between the two layers until a stable situation is achieved. Fig. 3-12 illustrates the ART-1 architecture which consists of 3 parts: a *gain controller*, an *attentional subsystem* and an *orienting subsystem*.

The *attentional subsystem* is the central part of the ART-1 network and consists of two layers, a bottom layer F1 (also called comparison layer) and a top layer F2 (also called recognition layer). Each new input vector containing N elements will be copied into the F1 layer before starting the learning algorithm. The input vector can only contain digital values. The final classification decision is indicated by a single firing neuron in the F2 layer (winner take all). The synaptic connections (weights) between these two layers are modifiable in both directions. These connections are called Long Term Memory (LTM) and contain the “knowledge” of the network.

The input vector stored in the F1 layer is transferred through the weights (LTM) towards the F2 layer and back again to the F1 layer. This response from the F2 layer is compared with the original input vector by a *vigilance* check performed by the *orientation subsystem*. The *vigilance* is the maximum distance between the input vector and the cluster center corresponding to the firing recognition layer neuron.

If the *vigilance* check fails, a previously unallocated neuron will be allocated to a new cluster category corresponding to the input vector. If the test passes, the winner neuron (inhibiting the other neurons in the recognition layer) is trained such that its associated cluster center in feature space is moved toward the input vector. This mechanism of inhibiting other neurons is common in artificial neural architectures, inspired by the visual neurophysiology of the biological systems.

The outstanding features of ART are:

- Only one parameter has to be adjusted: *vigilance*:
The *vigilance* represents a measure of the distance between the input vector and the cluster center. A previously unallocated neuron will be allocated to the new cluster category if the *vigilance*-test fails.
- ART attempts to address the *stability-plasticity dilemma*:
meaning the ability to constantly learn new experiences without ignoring old ones. This capability is mainly based on the fact that ART can constantly allocate new cluster categories.
- The learning process is performed on-line i.e. the net can learn in the same cycle while responding to an input stimulus.

3.4.3 Simplified Fuzzy ART

As the name implies, *Simplified Fuzzy ART* (SFA) is a simplification of the *Fuzzy ARTMAP* which was developed at Boston University by Steve Grossberg and Gail Carpenter [Carpenter et al, 1991c] (see also [Kasuba, 1993] and [Dubrawski et al, 1994] for a good introduction).

The *fuzzy* in the term SFA refers to the fact that the network incorporates fuzzy logic. The *ART* portion refers to Grossberg and Carpenter's Adaptive Resonance Theory, which was already described in the previous section. The *MAP* portion in the Fuzzy ARTMAP refers to the ability to map input and output events. This leads the ARTMAP to be a supervised learning system, thus the MAP part was removed in order to get an unsupervised learning system.

Fig. 3-13 shows an overview of the *Simplified Fuzzy ART* algorithm.

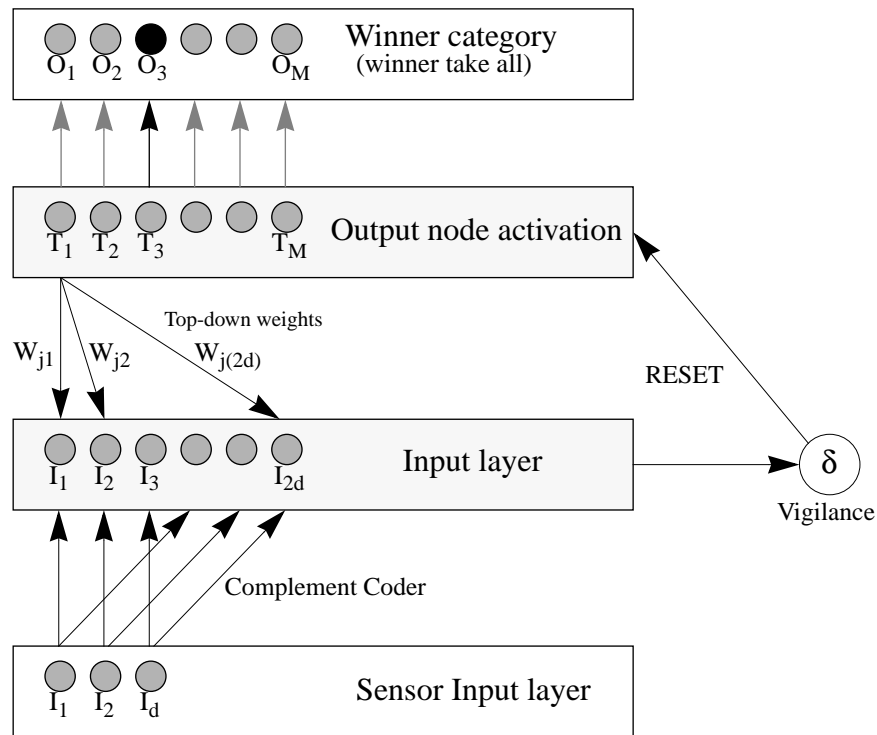


Fig. 3-13: Sketch of a Fuzzy ART network

The Simplified Fuzzy ART is a two-layer neural network able to classify input vectors of d dimensionality by one winner category. The number of categories can be increased at any time. The following two characteristics make the network unusual compared to other neural networks:

- The comparison of an input vector and neuron weights is based on fuzzy logic. The fuzzy AND operation used is defined in eq. 3-6. Section 3.5.3 describes the resulting effect.

$$\text{Fuzzy AND: } (A \wedge B) = \min(A, B) \quad (\text{Eq. 3-6})$$

- The *Simplified Fuzzy ART* algorithm described later can be simplified by normalization of the input vector. This is done by *complement coding* of the input vector which therefore represents the presence and the absence of a particular feature. The input vector with the elements (a_1, a_2, \dots, a_d) is doubled by its complement vector $(\bar{a}_1, \bar{a}_2, \dots, \bar{a}_d)$ with $\bar{a}_i = 1 - a_i$. The full input vector I has a dimension of $2d$. The norm of $|I|$ is always d , since:

$$|I| = |(a, \bar{a})| = \sum_{i=1}^d a_i + \left(d - \sum_{i=1}^d a_i \right) = d \quad \text{where: } |p| = \sum_{i=1}^d p_i \quad (\text{Eq. 3-7})$$

The Fuzzy ART algorithm:

The network requires a mechanism to form an activation of the output layer in response to the input of the network. When a “complement coded” input vector is presented to the network, the output nodes become activated as showed in eq. 3-8.

$$T_j(I) = \frac{|I \wedge W_j|}{\alpha + |W_j|} \quad (\text{Eq. 3-8})$$

α is kept at a small value close to zero, usually about 10^{-7} . This value has been found by experience by [Kasuba, 1993] and is independent of the application. It only avoids a division by zero in the start condition. The winning output node is the node with the highest activation (winner take all):

Winner= $\max(T_j)$.

The match function (see eq. 3-9) calculates the degree to which the complement coded input vector I is a fuzzy subset of W_j . If the match function value is greater than the vigilance parameter δ , the network is said to be in resonance which means that the output node j is good enough to encode the input vector I .

$$\frac{|I \wedge W_j|}{|I|} = \frac{|I \wedge W_j|}{d} \geq \delta \quad (\text{Eq. 3-9})$$

The top-down weight vector W_j from the output node is updated according to eq. 3-10 if the match function in eq. 3-9 is greater than δ . β is the learning rate which can be slowly decreased to reduce change of the weight with time. Greater values of β result in faster learning.

$$W_j^{new} = \beta(I \wedge W_j^{old}) + (1 - \beta)W_j^{old} \quad 0 < \beta \leq 1 \quad (\text{Eq. 3-10})$$

If no winning output node can be found, a new output category has to be created, assigning to it a new vector of adaptive weights initialized with $W_j^{new} = 1$. Thus new patterns may fit in and modify the shape of the existing category, if they match closely, or require establishment of new categories when necessary. Since the weights values may only decrease during the adaptation process, the learning method is convergent.

As opposed to other neural network types, no source code example could be found, neither on the WEB nor in the literature. Therefore it is worth investing two pages for a C-source in the Appendix on page 83.

3.5 Discussion of unsupervised classifiers

The three unsupervised classifiers mentioned above are derived from three different ideas. However, they are based on the same mathematical principle. All three neural networks use *euclidean distance* between the new input vectors and the classes. This is not astonishing because there is no more information available for a better decision.

Therefore, the euclidean distance plays a central role and is very decisive for the similarity of classes. The unsupervised classifiers used are designed (or modified) in such a manner as to have a stable maximum euclidean distance between classes. Therefore the network keeps the ability to learn new input vectors at any time. Assumption: Unallocated classes can be recruited at any time.

In order to analyze the differences between the unsupervised classifiers, they were implemented into the Khepera simulator to classify all sensor information. During about 10 minutes of random movement, 15'000 sensor examples were simultaneously presented to all classifiers. This experiment was repeated several times but with different vigilance parameters. The result is shown in the following sections:

3.5.1 Significant versus entire number of classes

The only relevant parameter to trim the behavior of the unsupervised classifiers is the maximal euclidean distance called *vigilance*. A small vigilance increases the number of classes and vice versa.

Fig. 3-14 shows the number of significant classes¹ produced by all three unsupervised classifiers as a function of the entire number of classes used. It can be seen that there is a more or less a linear relationship between the number of significant and used classes. This is comprehensible because both depend on the decreasing vigilance.

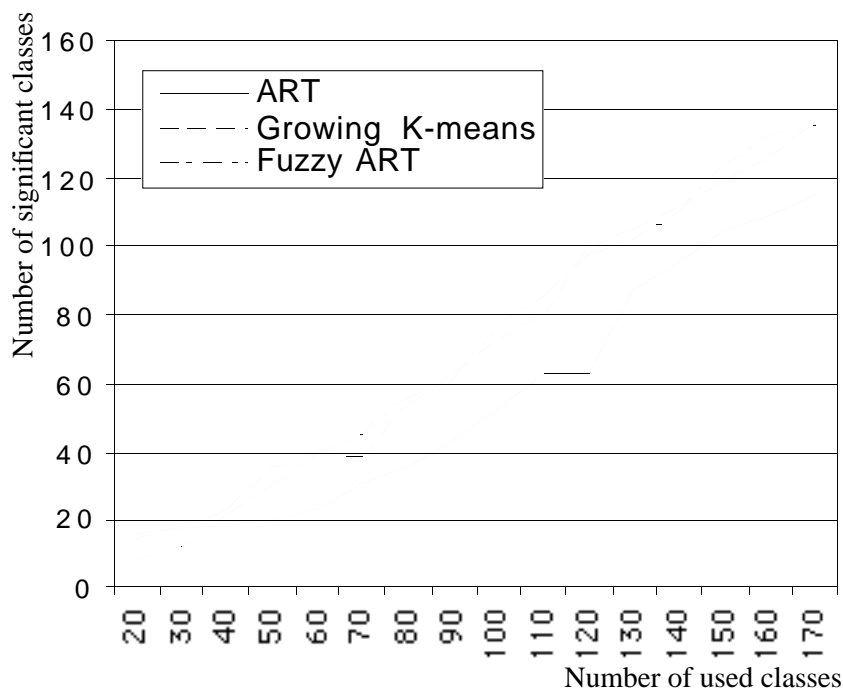


Fig. 3-14: Almost linear relationship between significant and used classes

1. see chapter 3.2.3 on page 40 for definition of a significant class.

3.5.2 Quality versus quantity of significant classes

The euclidean distance between class prototypes must be short in order to separate sensory experiences corresponding to significantly different environmental situations. On the other hand, too small classes (in the spatial sense) might be reactivated very rarely and might probably represent a hardly reproducible event. A compromise for the number of classes has to be found.

The diagram in fig. 3-15 shows how often a significant class was *reactivated* as a function of the total number of classes. The data are collected during the same experiment as described above. The descending lines in the diagram show that classifiers containing too many classes are less efficient. The reason is that these classes are too specific for a special input vector and therefore not reliable enough to recognize a noisy landmark.

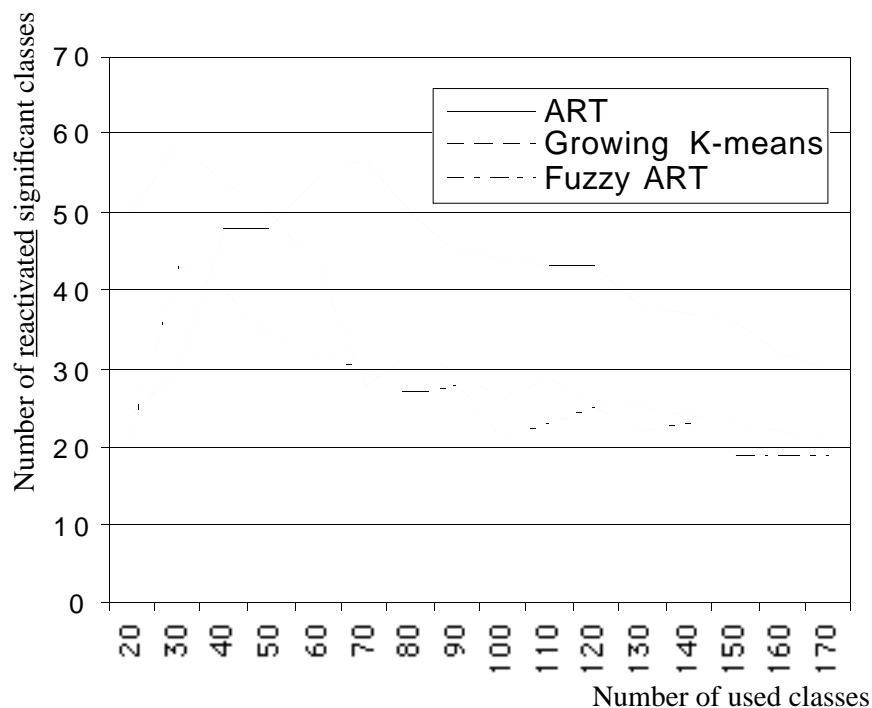


Fig. 3-15: The reactivation of significant classes is better for small neural networks

3.5.3 Class distribution in the input space

Increasing the number and types of classifying systems gives a more reliable characterization of the environment. Each classifier has an other way to separate clusters from each other. The different types of cluster shapes showed in fig. 3-16 increase the probability that all significant features of the environment are reliably enclosed. The cluster distribution showed in fig. 3-16 was generated by the classifiers applied in the simulator, but processing artificially generated two dimensional random values. This was necessary in order to present the distribution characteristics on a two dimensional figure.

It is interesting to mention, that the Fuzzy ART region marked by A and B in fig. 3-16 belongs to the same class, in spite of their very different places. This shows that a significant class could be put into a much larger region of another class without splitting it up.

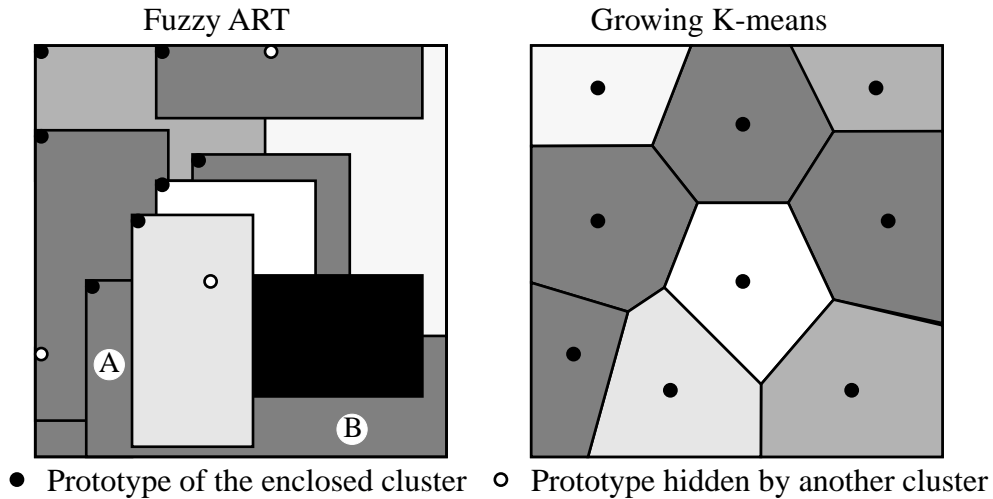


Fig. 3-16: Different cluster shapes for a 2D input space

3.5.4 Representing the environment by classes

Fig. 3-17 shows how the produced classes (black traces) represent the environment consisting of walls (squares \square) and light sources (crosses \times). The simulated environment produced about 19% good classes (a, b), 33% risky classes (c), 12% too large classes (d, e) and 36% unsuitable classes (f). Only classes of type a, b and c were used to create landmarks in order to recalibrate the robot's odometry error. We can make the more precise following comments about these classes:

- a) A just perfect class: Small and unique, perfect for position recalibration.
- b) A usable but large class, produced probably by the walls on both sides of the robot.
- c) Small but scattered class representing a risk of confusion during assignment of a position.
- d) Too large class, produced by the light source, unusable for odometry calibration.
- e) Too large class produced by the camera seeing a special obstacle constellation in front of the robot.
- f) Too large and too distributed class.

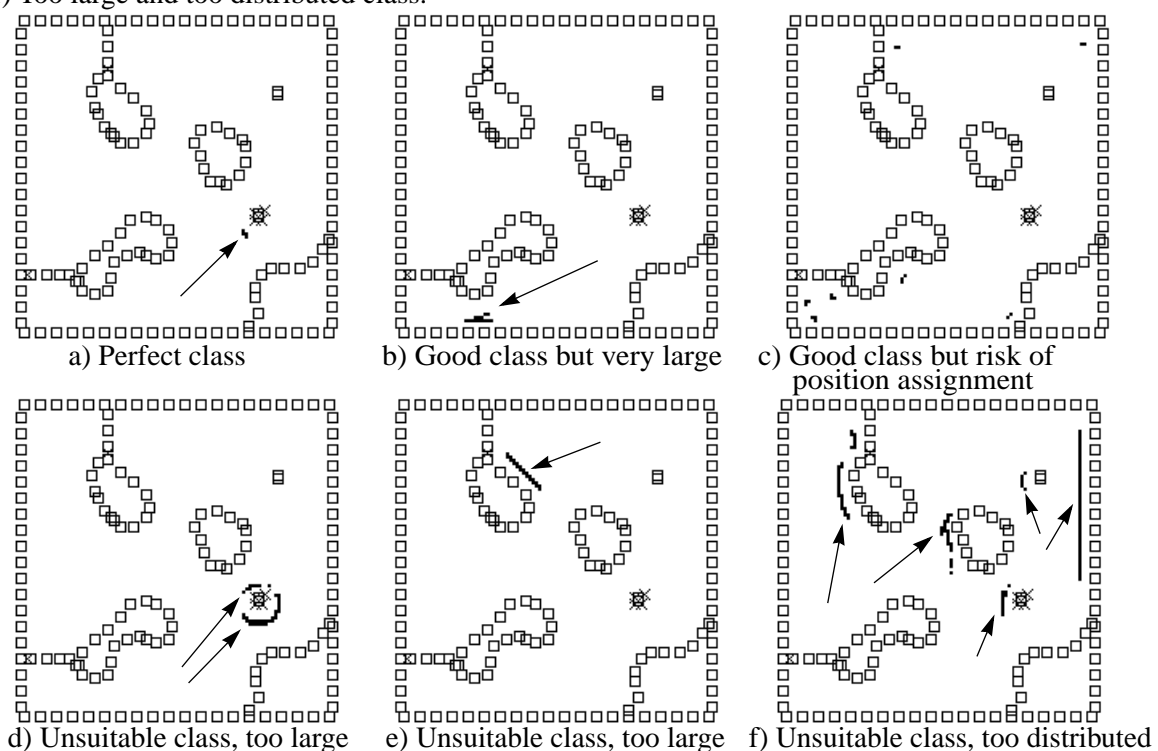


Fig. 3-17: Different environment representation by classes

3.6 Classes become landmarks

During the robot's mobility, each classifier produces a stream of classes. It is almost impossible to link classes directly to significant robot situations because the classes are too sensitive to signal noise or minor changes of the environment. Therefore we combine several classes from all class streams together into a new unity called *landmark*. Fig. 3-18 illustrates how to make this class combination.

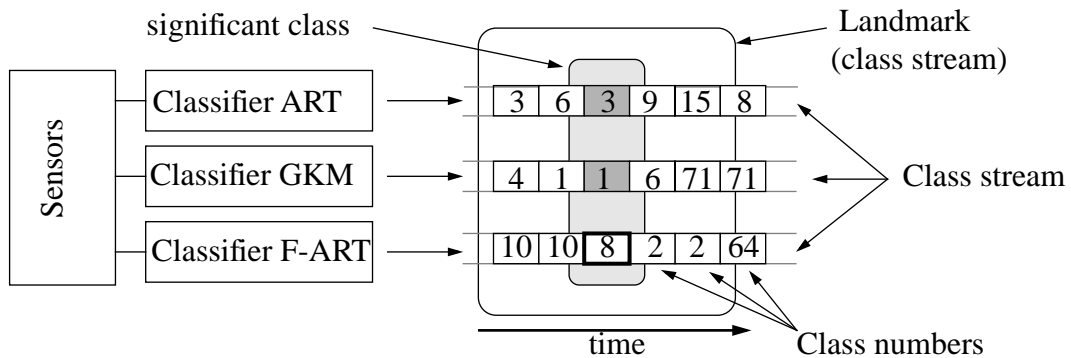


Fig. 3-18: Combining three class streams into a landmark

The statistical module already mentioned (see section 3.2.3 "Algorithm overview") selects a significant class from all class streams. This class becomes the "kernel" of the future landmark (illustrated as class number eight in the example above). The kernel will be surrounded by fifteen former and fifteen future classes of the same class stream. In addition, the chronological identical class stream of the other classes are merged into the same landmark as well.

A *landmark* is a significant class combined with its adjacent classes thus increasing the uniqueness of a landmark and decreasing the risk of confusion. Additionally a landmark contains some further information such as the estimated robot position, time, direction, etc. Our simulated robot environment contains about 300 landmarks. However, only a few will be used regularly. Fig 3-19 illustrates "in reality" how a landmark is created by one or several significant classes and its surrounding "normal" classes.

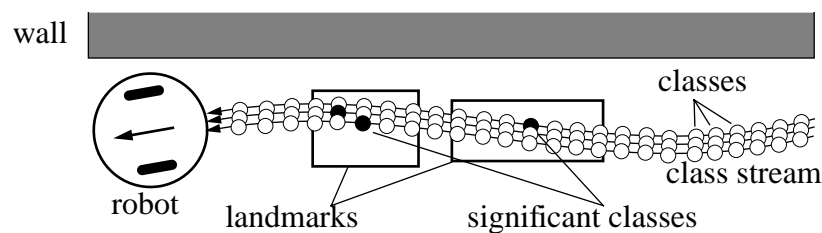


Fig. 3-19: A landmark is a significant class combined with the adjacent classes

3.7 Landmark comparison by Levenshtein Distance

Landmarks (a set of classes) created by a robot at the same position and in equivalent circumstances are rarely identical; they always differ by some entries: some former classes may be missing, new ones may have been added.

A powerful algorithm for string comparison is needed which can handle such missing and additional entries in a string. The problem is similar to a database engine searching entries by an approximate keyword. For instant, the keyword “color” should also find entries containing the word “colour”, even if the string itself is almost completely different. The *Weighted Levenshtein Distance* (WLD) developed in the seventies by V.I. Levenshtein [Levenshtein V.I., 1975] is a widely used algorithm to compare two strings of discrete elements in a tolerant way.

3.7.1 Introduction

The *Levenshtein Distance* (LD) is a distance measurement of two discrete string elements for example the string A=“ec” and B=“fc”. The LD of A and B is defined as the minimum number of editing operation needed to convert string A into string B by using the following three operations:

- *Deleting* of an element.
- *Insertion* of an element.
- *Substitution* of an element.

In our example, the LD of string A=“ec” and string B=“fc” is 1 (one substitution of the first element). The LD can be evaluated by a dynamic programming algorithm, see [Reuhkala E., 1983].

The *Weighted Levenshtein Distance* (WLD) is a generalization of the unweighted Levenshtein Distance (LD) which additionally assigns a cost to each operation. The WLD is defined as the minimum total cost required to convert string A into string B and can be evaluated using the same dynamic programming algorithm as for the LD.

3.7.2 Recursive mathematical definition

The WLD algorithm (see eq. 3-1) was mainly used for speech processing and molecular biology. Generally, the WLD algorithm can be used to compare strings of discrete elements in a tolerant way, for example to look up words in a database by slightly derivating key words. The following discussion explains the general algorithm. Let’s call the two strings a and b with the length of i and j , so the WLD can be defined in the following recursive manner:

$$L(a_i, b_j) = \min \begin{cases} L(a_{i-1}, b_j) + \text{cost}_{del} \\ L(a_i, b_{j-1}) + \text{cost}_{ins} \\ L(a_{i-1}, b_{j-1}) + \begin{cases} 0 & \text{if } a[i] = b[j] \\ \text{cost}_{sub} & \text{if } a[i] \neq b[j] \end{cases} \end{cases} \quad (\text{Eq. 3-1})$$

x_n : The first n elements of the chain x
 $x[n]$: The n^{th} element of the chain x
 cost_{del} : Deletion cost of an element
 cost_{ins} : Insertion cost of an element
 cost_{sub} : Substitution cost of an element

The WLD is defined as the minimum total cost required to convert string a into string b by using *dele-*

tion, insertion and substitution. The recursion is completed by adding the value specified in eq. 3-2, if at least one string becomes empty.

$$\begin{aligned}
 L(0,0) &= 0 \\
 L(a_n,0) &= n \cdot \text{cost}_{del} \\
 L(0,b_n) &= n \cdot \text{cost}_{ins}
 \end{aligned}
 \tag{Eq. 3-2}$$

This recursive formulation is a very long calculation because every operation will cause three sub-operations. Fig. 3-20 shows the combination tree needed to compare two strings containing each two characters. The tree shows that there are 13 different combinations of deletion, insertion and substitution to proceed until one string gets empty.

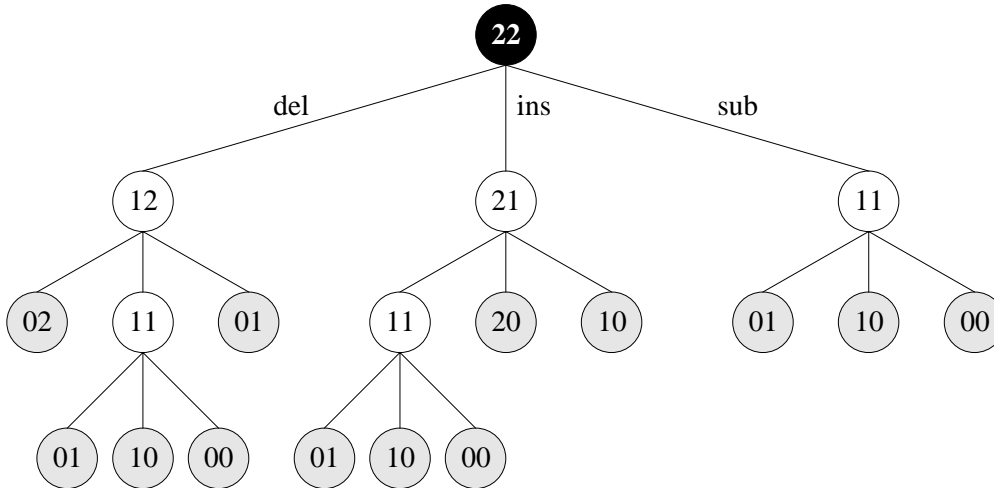


Fig. 3-20: Combination tree to compare two strings with each two characters

Table 3-1 shows in the general manner the number of comparisons required to compare two strings of equal length n (the case $n=2$ correspond to fig. 3-20):

string length n	number of comparison c
1	3
2	13
3	63
4	321
5	1'683
6	8'989
7	48'639
8	265'729
9	1'462'563
10	8'097'453

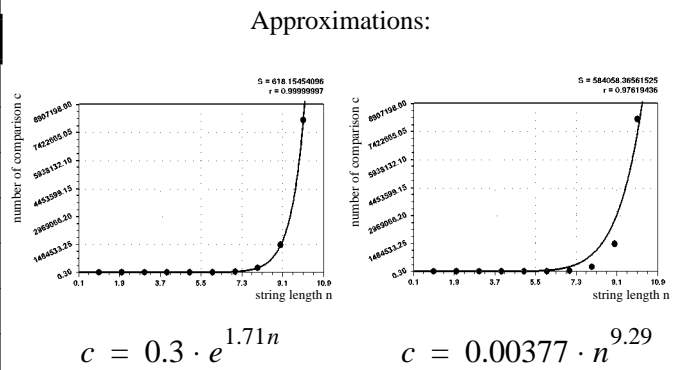


Table 3-1: Number of comparison required for the recursive definition and it's approximation

It can be seen that the recursive implementation is unsuitable for long strings. The reason is that the calculation expenditure grows by the power of three for each new element in the string. For instance, the Levenshtein Distance $L(a_{i-1}, b_{j-1})$ is calculated three times. The first time directly on the first recursion level and a second time on each following level after a deletion followed by an insertion or after an insertion followed by a deletion.

3.7.3 Flat implementation of the LD

The recursive definition of the LD is intuitive, but too costly for an implementation. In fact, there exists only $(i+1)$ times $(j+1)$ different combinations for the string a_i and b_j . Therefore, the problem can be represented by a matrix of $i+1$ columns and $j+1$ rows (see table 3-2). Each cell contains the Levenshtein Distance of the string part corresponding to the column and row index.

The value of all cells in the first row ($i=0$) and the first column ($j=0$) can be calculated by the termination condition (see eq. 3-2):

	0	1	2	...	j
0	0	1	2	...	j
1	1				
2	2				
...	...				
i	i				result

Table 3-2: First setting up of the LD table containing up to now only the termination value (see eq. 3-2)

From now on, every upper left free cell can be calculated by eq. 3-1 with the following replacements:

$L(a_{i-1}, b_j)$ is the value over the cell
 $L(a_i, b_{j-1})$ is the value left of the cell
 $L(a_{i-1}, b_{j-1})$ is the value in the diagonal upper left cell

The final Levenshtein distance can be found after calculating every cell in the right under cell (grey result cell).

Note: To make global string shifts invariant, the LD algorithm can be changed so that deletion and insertion cost zero at the beginning and at the end of the strings.

3.7.4 Code example for WLD

The following algorithm may be used for the computation of the WLD. String A will be transformed into string B using the matrix M (flat implementation). The weights are shown as functions, thus $d(A(3))$ denotes the “cost” of deleting the third character 3 of string A, $i(\dots)$ denotes the insertion cost and $r(A(i), B(j))$ denotes the cost to replace the character i in string A with the character j in string B. The unweighted Levenshtein distance (LD) is a special case with $r(\dots)=d(\dots)=i(\dots)=1$.

```

/* Algorithm Weighted Levenshtein Distance */
begin
M(0,0):=0;
for x:=1 step 1 until length(A) do M(x,0):=M(x-1,0)+d(A(x));
for y:=1 step 1 until length(A) do M(0,y):=M(0,y-1)+i(B(y));

for x:=1 step 1 until length(A) do
  for y:=1 step 1 until length(B) do
    begin
      m1:=M(x-1,y-1)+r(A(x),B(y));
      m2:=M(x-1,y)+d(A(x));
      m3:=M(x,y-1)+i(B(y));
      M(x,y):=min(m1,m2,m3);
    end
  end
WLD:=M(length(A),length(B));
end

```

3.7.5 Graphical example

In order to demonstrate the efficiency of the Levenshtein Distance algorithm, a short example¹ will be shown. We want to calculate the distance between the word “STEPHAN” and “STEFANIE”.

First, we can fill the first row (string length =0) and the first column (string length =0) with the values corresponding to the condition specified in eq. 3-2. After that the other cells can be calculated corresponding to eq. 3-1, starting with the upper left free cell. It is useful to mark each cell which case of eq. 3-1 was used (ins, del, rep, eq=equal). This can be used later to reconstruct the shortest operation chain. Table 3-3 shows the complete resulting matrix. The lower right cell (highlighted) shows the result of 4 operations.

(length)	(0)	S (1)	T (2)	E (3)	F (4)	A (5)	N (6)	I (7)	E (8)
(0)	0 [eq]	1 [ins]	2 [ins]	3 [ins]	4 [ins]	5 [ins]	6 [ins]	7 [ins]	8 [ins]
S (1)	1 [del]	0 [eq]	1 [ins]	2 [ins]	3 [ins]	4 [ins]	5 [ins]	6 [ins]	7 [ins]
T (2)	2 [del]	1 [del]	0 [eq]	1 [ins]	2 [ins]	3 [ins]	4 [ins]	5 [ins]	6 [ins]
E (3)	3 [del]	2 [del]	1 [del]	0 [eq]	1 [ins]	2 [ins]	3 [ins]	4 [ins]	5 [ins]
P (4)	4 [del]	3 [del]	2 [del]	1 [del]	1 [rep]	2 [rep]	3 [rep]	4 [rep]	5 [rep]
H (5)	5 [del]	4 [del]	3 [del]	2 [del]	2 [rep]	2 [rep]	3 [rep]	4 [rep]	5 [rep]
A (6)	6 [del]	5 [del]	4 [del]	3 [del]	3 [rep]	2 [eq]	3 [rep]	4 [rep]	5 [rep]
N (7)	7 [del]	6 [del]	5 [del]	4 [del]	4 [rep]	3 [del]	2 [eq]	3 [ins]	4 [ins]

Table 3-3: Levenshtein distance table for the word “STEPHAN” and “STEFANIE”

To identify the minimal operation steps to go from “STEPHAN” to “STEFANIE”, we start on the right lower cell and search the cheapest way through the matrix to the left upper cell. “Cheap” means to follow the preceding cell along with the smallest distance value. The resulting path is highlighted in table 3-3.

The result can be read by following the highlighted path from the left upper cell to the right under cell. Cells marked by [eq] has to be ignored, so you get the following operation chain:

- Step 1, table position [4,3]: Delete the ‘P’ in “STEPHAN”
- Step 2, table position [5,4]: Replace the ‘H’ of “STEPHAN” by a ‘F’
- Step 3, table position [7,7]: Insert an ‘I’
- Step 4, table position [7,8]: Insert an ‘E’

Some very nice Levenshtein simulators on the WEB are cited in the Appendix on page 86.

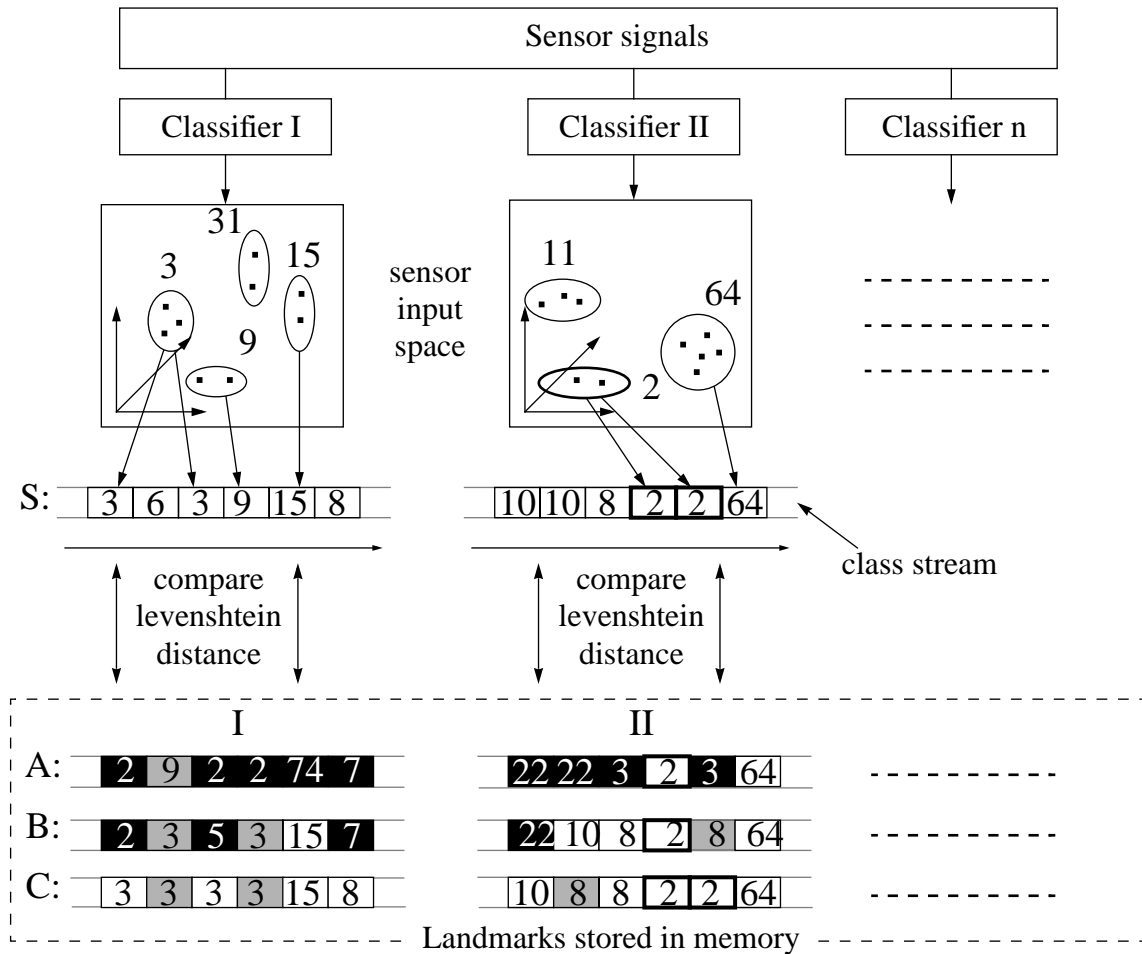
3.7.6 Calculation of substitution cost

The explained WLD algorithm was slightly enlarged to better fit the demanded task. The substitution cost (cost_{sub}) is taken as a constant value in the original WLD algorithm. Because of the high sensitivity of the unsupervised classifier, even close events will be declared by different class numbers, which suppress any relationship between these two classes from the WLD algorithm. Therefore the substitution cost (cost_{sub}) is calculated by measuring the euclidean distance of the two concerned classes in the sensor input space. The class numbers are not used in this calculation, since they are chosen arbitrarily and contain no distance information.

1. The example was inspired by http://www.media-eng.bielefeld.com/~mlab/hholzgra/fh_diplom

3.7.7 Recognizing identical landmarks by WLD

A freshly received landmark will be compared with landmarks which only contain the same significant class. Fig 3-21 shows the sensor signals produced by two different classifiers (I and II). The classifier II discovers a trigger class #2 (framed with a bold rectangle), which even appears twice by chance. All class streams (I and II) of the same time sequence are compared with the corresponding class streams of other landmarks already stored which contain the same significant class (#2). The example in fig. 3-21 shows three suitable landmarks (A, B and C) which are already stored in memory. The similarity of class streams is measured by the WLD.



- 3 Perfect matching class in a class stream
 - 9 Existing class, but shifted in a class stream
 - 8 Not existing class in this class stream
 - 2 Significant class which is generating and selecting a landmark
- A-C: Landmarks already stored in the memory

Fig. 3-21: Landmark comparison by analyzing all class streams

Result of the three landmarks comparison:

- Landmark A was generated due to the same trigger class (#2). However, almost every classes in the class stream are different (white number on black background) indicating a completely dissimilar landmark.
- The landmark B is different as well (mainly because of class stream I). This can happen, if the sensors-group, analyzed by classifier II, is not sensitive for a specific dissimilarity. That's why several class streams are analyzed independently.
- Landmark C is determined as identical to the received landmark, because all class streams are sufficiently similar.

3.7.8 Estimated robot position supports landmark recognition

The presented Levenshtein distance recognizes landmarks very reliable, even if some classes are missing or added. However, similar environments produce similar classes and therefore similar landmarks which could confuse later processing. Thus the assumed robot position becomes relevant if the Levenshtein algorithm identifies several matching landmarks. The conditions to appoint a matching landmark depends therefore on the Levenshtein distance d_{lev} and the estimated position difference d_{dist} . Each distance is a dimension of a two dimensional space and the final difference is the scalar distance between the investigated landmark and zero (see fig. 3-22). A new landmark will be generated if no landmark inside a certain threshold can be found.

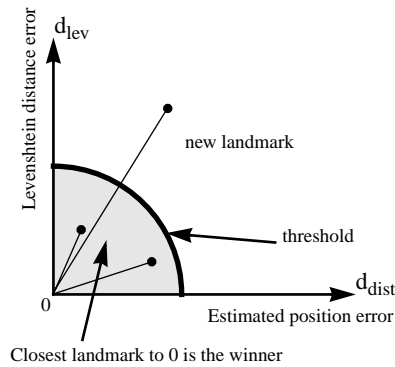


Fig. 3-22: Similarities of landmarks depend on Levenshtein distance and position error

3.8 Constraining robot movements

An autonomous robot moving without any strategy in a two dimensional space will not find enough landmarks to calibrate its odometry. Some hints could help to take advantage of available landmarks. Steering the robot actively to landmarks requires knowledge about the characteristics of the sensor systems, knowledge about the landmarks and the direction/distance of landmarks/obstacles. The robot doesn't possess this information, so an active steering system cannot be implemented in this way.

A possible solution for a steering system is to let the robot be attracted by any "interesting" signal event. "Interesting" means stimulating; the short range distance sensor of the robot supplies only interesting information, if the robot is close to an obstacle. This signal can be used in a fitness function to train a NN for a "wall following behavior". In the same way, the camera supplies only interesting information if the robot heads to an object presenting sufficient contrast. This signal can be used in a NN fitness function to visit edges and objects.

The fitness function can be described as following:

- A The robot has to move (cover a certain trajectory over time). This excludes stops, turning on the spot and bumping into obstacles.
- B Any kind of active sensor event produces a positive feedback. This has basically two effects:
 - B1:** The robot is attracted by walls and obstacles, through the short range distance sensors.
 - B2:** The robot is attracted by obstacles (in general contours) located in the view of the camera.

The rules B1 and B2 are contradictory because the robot has to give up the wall (or obstacle) attraction to aim an object (or contour) in the free space. Therefore the rules for B1 and B2 cannot be combined in the same fitness function and have to be used separately to train for instance two different NNs. The NNs are applied randomly, influenced by the presence of walls and contours. However, at this time the NN was not implemented but the expected result was programmed to save time for more important parts of this experiment.

Combining these behaviors allows the robot to explore the environment by following interesting stimuli as shown in fig. 3-23.

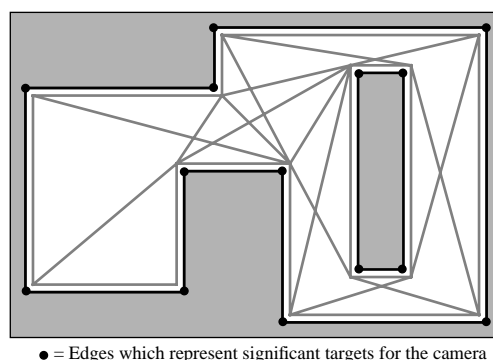


Fig. 3-23: Robot path attracted by walls and contours

The expected result by combining these behaviors is shown by gray lines in fig. 3-23. The robot explores the environment by following interesting stimuli. Thanks to the camera, it can even overcome empty areas to reach other objects.

3.9 Introduction to concepts

For us humans it is normally not very difficult to become familiar with a new environment. We scan it for objects and remember their spatial relationships. When explaining the route to a tourist, we link successive objects (bridges, monuments, etc.) with spatial information (left, facing, 50 feet, etc.). Of course, what is a striking object depends on our perception. Living in a different sensory world, a dog would rather select odor traces than monuments. Because of its different sensor ability, a dog “sees” its environment differently and therefore it uses different *concepts*. A real outdoor environment offers a huge quantity of information like colors, compass-direction, light, distance, etc. Human beings and superior animals (agents) are able to process this flood of information and abstract *concepts* like door, road and object, which are used to handle the environment on a higher level. Thus, the concepts depend strongly on the task and the sensor abilities and are therefore different for humans, animals and robots.

The Webster¹ defines a *concept* as follows:

A concept is the collection of objects and other entities that are assumed to exist in some area of interest and the relationships that hold among them. A conceptualization is an abstract, simplified view of the world that we wish to represent. For example, we may conceptualize a family as the set of names, sexes and the relationships of the family members. Choosing a conceptualization is the first stage of knowledge representation.

A short definition for robot application could be:

Organization and interpretation of the robot’s sensory input in a way suitable for its capabilities and requirements.

The application of *concepts* is not new [Pfeifer R., 1996], [Scheier Ch., 1996] and many recognition algorithms (specially in Artificial Intelligence) leave the creation of concepts to the machine in order to increase the flexibility and to reduce pre-wired and therefore unsuitable influences by the designer. But automatic concept creation is normally applied on quite a high abstraction level, for example to recognize objects or situations, based on sensor and status information which are preprocessed in a more conventional manner. Such preprocessing always leads to a loss of information which could be important and therefore reduces the power of automatic concept creation.

The concepts in this paper are based on landmarks, which are created without any model or specific preprocessing, which make the concept very adaptive to the environment. Therefore the automatic concept creation depends on the raw sensor signals [Kuipers B., 1997]. Ideally, such a system should be completely sensor-independent and free of any preprocessing. Of course, this is a quite theoretical consideration and some minimal preprogrammed functions like constant speed and constrained robot movement have to be implemented. However, we want to investigate the limitations of such a method for automatic sensor concept creation and see how close an experiment can approach this theoretical, ideal case.

The following section shows the dependencies of automatic concept creation:

1. FOLDOC (Free On-Line Dictionary Of Computing): <http://wombat.doc.ic.ac.uk>

3.9.1 Concept quality depending on frequency

The object's frequency is important to define concepts. Fig. 3-24 shows a scout looking for a mail box in a town. He will use significant objects like trees and churches as a concept to find his way again. On the other hand, the same scout looking for his tent in a forest will not use trees but houses as a concept. It is interesting to note that an empty space like a clearing can also be a significant concept.

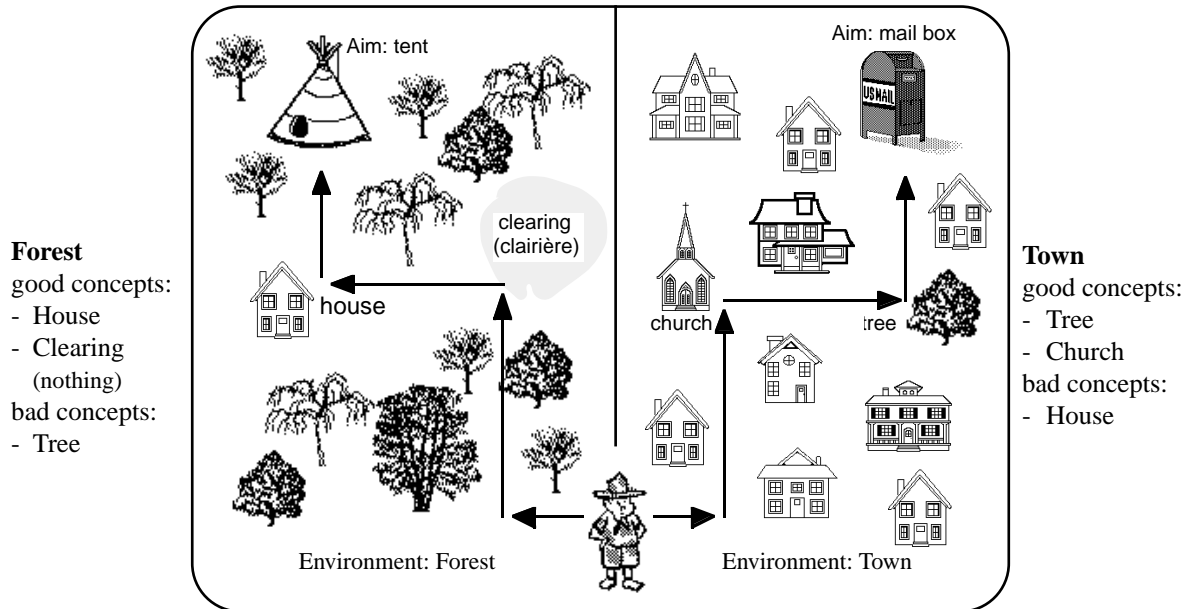


Fig. 3-24: Object's frequency influences the concept's quality

3.9.2 Concept quality depending on the sensor ability

Different sensor abilities influence the concept as well. A blind person uses mainly his tactile capabilities and the surrounding noise to orientate in an environment. They are normally much better developed than the tactile capability of "normal" persons, of which orientation is based mainly on vision. Therefore, the concepts of the environment can be very different depending on the sensor's ability.

Fig. 3-25 shows a Roman and a dog (agents) observing the same monument. They link different concepts to the monument due to different sensor abilities.

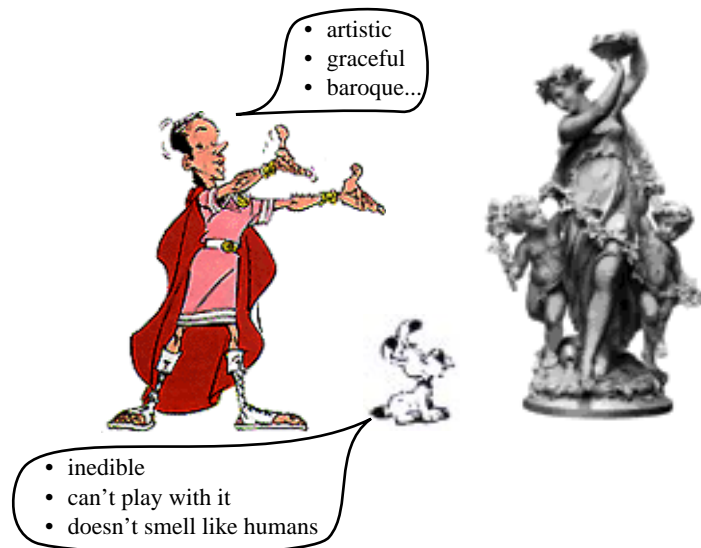


Fig. 3-25: Different concepts for the same object depending on the sensor ability

3.10 Landmarks become places

Landmark recognition can be done without any knowledge about sensor and environment and is very useful for sporadic odometry correction every time a landmark appears. However, landmark recognition is not a reliable reproducible process. Small change in the environment or a robot trajectory drift can disable or regenerate a landmark. That's not bad because the new landmark will be rediscovered next time and is therefore useful for odometry correction. On the other hand, reliable navigation and path planing is not possible due to the unstable detection of landmarks.

A further abstraction level called *places* has to be introduced. A place has to be recognized if the robot is close to it, even if some containing landmarks are not detected. Landmark and place recognition have the following different characteristics:

- The spatial position of classes inside the landmarks is decisive in order to use landmarks for odometry correction.
- The temporarily succession of landmarks inside the places is decisive in order to rediscover the same place, even if some landmarks are missing or shifted.

3.10.1 Recognizing places from the landmark stream

A *place* is defined by a spatial accumulation of landmarks and represents therefore a significant robot position (e.g. corner, narrow corridor or rare optical event). The landmark position is defined relative to the position inside the *place*. Fig. 3-26 shows a robot's path producing 12 landmarks (small rectangle). The corner groups 5 landmarks into a place A. In the same way, the rising light (hidden behind a bend) groups 4 landmarks to a place B.

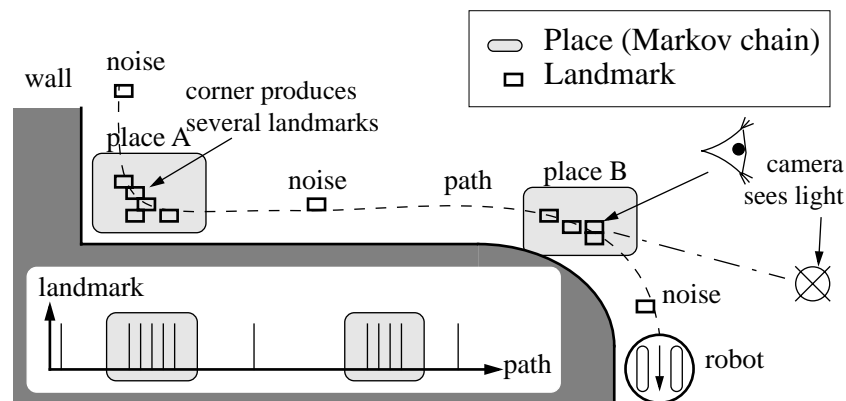


Fig. 3-26: Accumulations of landmarks become places

3.11 Place identification by Markov Chain

It is evident that a *place* (that is a sequence of landmarks) offers a better guarantee of proper identification than just one landmark. However, landmark sequences of the same *place* (S_{place}) always differ due to the drift of a robot's trajectory. The experiment shows that only some *subparts* of the sequence stay identical, but their associated position may also have drifted. The Weighted Levenshtein Distance cannot distinguish between missing and shifted landmarks and is therefore unsuitable for place recognition.

Markov chains are a better tool to recognize sequences because they are sensitive to landmark transitions and not to their position in the sequence. So it is probable that a place corresponding to a sequence of landmarks can be deduced. Fig. 3-27 shows a short overview of the Markov chain theory (see also [Collins L, 1975] and [Ross S. M., 1997]).

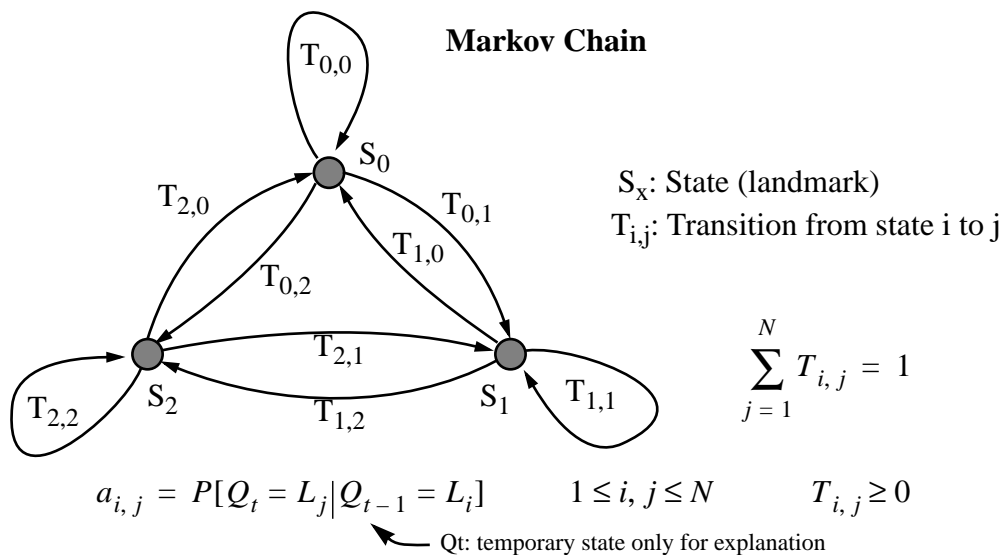


Fig. 3-27: Definition of First Order Markov Chain

Markov chain is an efficient and elegant tool for *describing* and *analyzing* any kind of probabilistic changes of state. Markov models which describe the changes can also be used to *forecast* future states. The basic concept of Markov chain is best explained by a short example.

3.11.1 Concept of probability

In our real world, we often attach a *probability* to the occurrence of every event. There is a probability that an airplane will crash or that I will win the main prize in the lottery this year. For most course of action we must assume or estimate the probability of such events. Thus in the case of airplanes, insurance companies need to know the probability of such events in order to charge the right rate to their customers. In case of wrong probability estimation, the company will either go bankrupt or it will charge rates of such high value that its clients will seek insurance coverage elsewhere.

Moreover, insurance companies also need to predict the probability of such events in the future to adopt the rates as early as possible. This change of probability can be predicted by a Markov model.

3.11.2 A simple Markov model

Information relating to the observed probabilities of past trends can be organized into a matrix which is the basic framework of a Markov model. I would like to use an example found in “*Models of the evolution of spatial patterns in human geography*” [Harvey D., 1967] which uses the probabilities of inhabitant movement between London city, its suburbs and the surrounding country between 1950 and 1960. The inhabitant movement (transition) during these ten years can be described in the following table (see fig. 3-28).

	London	Suburbs	Country
London	0.6	0.3	0.1
Suburbs	0.2	0.5	0.3
Country	0.4	0.1	0.5

Fig. 3-28: Transition probability matrix $[T]$ of English inhabitants

The three locations in this matrix forms the *states* of the model and each element the *transition* value (probability of moving from one state to any other state). Therefore we assume that of all the people living in London in 1950, 60% were still in London in 1960, 30% had moved to the suburbs and 10% had moved to the country. Thus, each row of the matrix, unlike the columns, sums to 100%. So the transition matrix describes the probability of movement from one state to any other state during a specified or *discrete* time interval (10 years). Let us further assume that 50% of the total population of the three states lives in London, 30% in the suburbs and 20% in the country. This leads to the following *initial state*:

$$S^0 = (S_1^0, S_2^0, S_3^0) = (0.5, 0.3, 0.2)$$

The initial state vector s^0 refers to the state of the system in 1950. s^1 would refer to the state of the system in 1960, s^2 to 1970 and so on. Using theorems of matrix algebra we can obtain s^1 by multiplying the initial state vector s^0 by T so that

$$S^1 = S^0 \cdot T = (0.5, 0.3, 0.2) \times \begin{bmatrix} 0.6 & 0.3 & 0.1 \\ 0.2 & 0.5 & 0.3 \\ 0.4 & 0.1 & 0.5 \end{bmatrix} = (0.44, 0.32, 0.24)$$

Thus, in 1960, 44% of the habitants of the three states will live in London, 32% in the suburbs and 24% in the country. We assume that the transition probabilities T and the total population in the three states remains stable during the following 10 years. So we can calculate the population in 1970 s^2 .

$$S^2 = S^1 \cdot T = (0.44, 0.32, 0.24) \times \begin{bmatrix} 0.6 & 0.3 & 0.1 \\ 0.2 & 0.5 & 0.3 \\ 0.4 & 0.1 & 0.5 \end{bmatrix} = (0.424, 0.316, 0.260)$$

Note that between 1950 and 1960 the suburban proportion increased from 30% to 32% but in the next decade declined to 31.6%.

3.11.3 Implementation of Place recognition by Markov Chain

Each *place* is represented by a ergodic¹ transition matrix:

$$M_{\text{place}} = \begin{bmatrix} a_{1,1} & a_{1,j} & a_{1,N} \\ a_{i,1} & a_{i,j} & a_{i,N} \\ a_{N,1} & a_{N,j} & a_{N,N} \end{bmatrix}$$

$a_{i,j}$: Transition from landmark i to j
 N : Maximal number of landmarks
 M : Ergodic Markov chain for a place

The size N of the matrix is fixed and defined by the number of different landmarks. Each cell contains the probability of transition from landmark i to landmark j . Remember that each row of the matrix sums to 1.0. Each new landmark sequence S_{new} has to be compared with all transition matrixes M_k (representing *places*). The following equation calculates the index w of the transition matrix M_w representing the most probable *place* for S_{new} :

$$w = \underset{k}{\operatorname{argmax}} P(M_k | S_{\text{new}})$$

index of winner place

A posteriori probability of being in place k given the new sequence of landmarks.

Bayes transformation allows to calculate the posteriori probability $P(M_k | S_{\text{new}})$ from the likelihood $P(S_{\text{new}} | M_k)$:

$$\text{Bayes identity: } P(M_k | S_{\text{new}}) = \frac{P(S_{\text{new}} | M_k) \cdot P(M_k)}{P(S_{\text{new}})}$$

A posteriori prob. Likelihood A priori prob.

The probability that M_k contains the sequence S_{new} is identical for every k , because new sequences have are by default independent to all existing place matrixes. Therefore $P(M_k)$ is constant for all k . $P(S_{\text{new}})$ is independent of k and can therefore be ignored as well. So w can be calculated as following:

$$w = \underset{k}{\operatorname{argmax}} P(S_{\text{new}} | M_k)$$

The winning transition matrix M_w represents the place with the highest probability of belonging to place S_{new} . Real correspondence has to be verified by a *normalized score* $L(S_{\text{new}}, w)$ which has to be over the threshold T :

$$\text{Normalized score: } L(S_{\text{new}} | w) = \frac{P(S_{\text{new}} | M_w)}{P(S_{\text{new}} | M_{\text{new}})} > T$$

The new sequence S_{new} corresponds to the winner place M_w if the normalized score exceeds the threshold T , otherwise a new place M_{new} will be added to the database. The threshold T of about 0.008 was found empirically.

1. Ergodic relates to a process in which a sequence or sizable sample is equally representative of the whole (as in regard to a statistical parameter), involving or relating to the probability that any state will recur, especially having zero probability that any state will never recur.

Ref: <http://pespmc1.vub.ac.be/ASC/ERGODIC.html>

In contrast to the posteriori probability $P(M_k|S_{new})$, the likelihood $P(S_{new}|M_k)$ can easily be calculated as following:

$$P(S_{new}|M_w) = a_{S_{new}^0, S_{new}^1} \cdot a_{S_{new}^1, S_{new}^2} \cdot a_{S_{new}^2, S_{new}^3} \cdot \dots \cdot a_{S_{new}^{n-1}, S_{new}^n}$$

$a_{S_{new}^x, S_{new}^y}$ is an element of the Markov matrix indicated by the x^{th} and y^{th} element of the landmark sequence string S_{new} . It is obvious, that the multiplication results in a very low number which could become very close to the computer's capacity. So it is advantageous to use the following equation to implement the *normalized score* $L(S_{new}, w)$:

$$\log a_{S_{new}^0, S_{new}^1}^W \cdot \log a_{S_{new}^1, S_{new}^2}^W \cdot \dots - \log a_{S_{new}^0, S_{new}^1}^{new} \cdot \log a_{S_{new}^1, S_{new}^2}^{new} \cdot \dots > \log T \mid a \in \mathfrak{R}^+, a > 0$$

After finding the winning transition matrix index w , we have to integrate the landmark sequence string S_{new} into the Markov matrix M_w . This can be done by incrementing the corresponding matrix cell by a value (i.e. $\frac{10}{N}$) followed by a normalization to satisfy the row sum of 1. The comparison of *places* can also directly be done using the corresponding Markov matrix. See also *Duda & Hart* on page 349 for further explanations [Duda & Hart, 1973]. The following equation delivers the index w of the nearest Markov Matrix to M_{new} out of all the examined Markov Matrix M_k .

$$w = \min_k \sqrt{\sum_{x=1}^N \sum_{y=1}^N (a_{x,y}^{new} - a_{x,y}^k)^2}$$

3.12 What's the difference between Levenshtein and Markov?

Markov chains and Levenshtein distance are used in this thesis to measure the distances between two strings. However, they have very different characteristics because of their individual mathematical background which is the reason to apply both algorithms independently.

The table 3-4 shows five different cases of string matching performed by Levenshtein and Markov algorithms. Each case is preceded by Levenshtein and Markov and is described in two boxes. The upper box explains the operation done by the algorithm independent of the task. The lower box (framed) explains the consequences depending on the task and is shaded if the consequences have a negative effect.

It can be seen, that both algorithms complement one another very well. There are only a few cases in which the algorithm does not suit the demands.

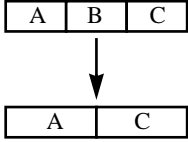
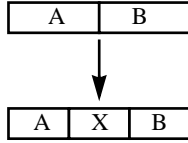
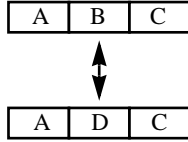
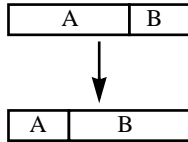
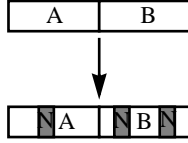
String manipulation	Levenshtein Distance	Markov Chain
Missing element 	Each former B element has to be replaced by an A or C element. This replacement will generate a high Levenshtein distance. => HIGH difference	The new transition A->C was unknown and produce therefore a negative effect. However, the extended A and C elements have no important effect. => LOW difference
	DESIRED! Missing and deformed classes makes the landmark "fuzzy" and unsuitable for position calibration.	DESIRED! Missing landmarks can happen and should not harm the place recognition.
Added element 	A part of the A and B element have to be replaced by the elements X. This replacement will generate a high Levenshtein distance. => HIGH difference	Adding a new elements X leads to a new A->X and X->B transition and several new and therefore punishing X->X transitions => HIGH difference
	DESIRED! New and deformed classes makes the landmark "fuzzy" and unsuitable for position calibration..	NOT DESIRED! Adding one new landmark to a place could happen and should not harm the place recognition.
Element oscillation 	An unstable element oscillating between two states generates always a high distance and cannot be learned by the Levenshtein algorithm. => HIGH difference	A Markov chain is able to store several transition chains which connects element A and C. So unstable elements have no punishing effect. => LOW difference
	NOT DESIRED! Class oscillation due to unsuitable classification can happen.	DESIRED! Single landmark oscillation should not harm place recognition.
Shifting element 	The second part of the A element has to be replaced by elements B which will produce a high Levenshtein distance. => HIGH difference	The shrank A element and the extended B elements have no important effect and produces no new transitions. => LOW difference
	DESIRED! Shifted classes make landmark unusable for position calibration.	DESIRED! Shifted landmark positions should not harm place recognition.
Noise influence 	Single different elements (noise) doesn't influence the Levenshtein distance in an important manner. => LOW difference	Noise (even of short time) produce a lot of new transitions (A->N, N->B and several N->N) increasing the distance. => HIGH difference
	DESIRED! Noisy classes doesn't harm the position of a landmark	DESIRED! Several new landmarks (noise) indicates different places

Table 3-4: Different characteristics for Levenshtein and Markov algorithm

3.13 Places become a map

The robot's position has to be matched with a map in order to execute a purposeful task. There are many different kinds of maps [Devy M., 1995] [Kampmann P., 1991] [Knieriemen T., 1991]. One approach is to project the environment on an already existing map in order to recognize the position by different matching algorithms [Piaget J., 1970] [Kurz A., 1994]. Another way is to project the environment on a topological map (cognitive map [Mataric M., 1990]), which does not contain the cartesian position, but a perception of the objects and how to find the way from one object to another.

This is also our approach. Maps are linked together in a topological map. Each link means an experience's trajectory to travel from one place to another. The robot recognizes its position by comparing its actual place with the places stored in the map. The estimated robot's position (by odometry) is only used to distinguish several identical places in the map.

The environment showed in fig. 3-33 generates about 40 places. Fig 3-29 shows some of the most important places which are sometimes overlapped. Place recognition is very reliable. They never get mixed up. However, places are sometimes not recognized due to too few landmarks.

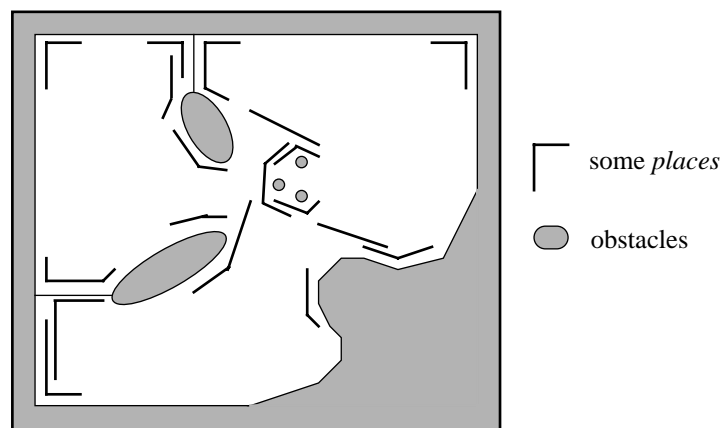


Fig. 3-29: Typical distribution of the most important places

As already mentioned, each place is linked with all reachable neighboring places. These links contain distance and direction information which allows to go from one place to another until the destination is reached. This network of links may be slightly distorted from reality and does not represent the environment true to scale. However, traveling from place to place is possible and should be largely sufficient for navigation. Fig. 3-30 shows how a real place distribution (left image) is projected in the robot's memory (right image).

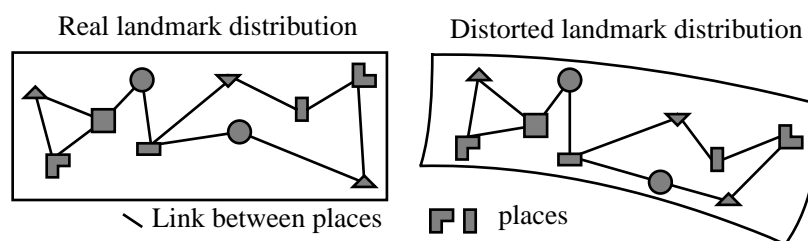


Fig. 3-30: Distorted representation of the environment

3.14 Algorithm controlling

Fig. 3-31 gives an overview about the interconnections of the different levels and the processing order divided into phases.

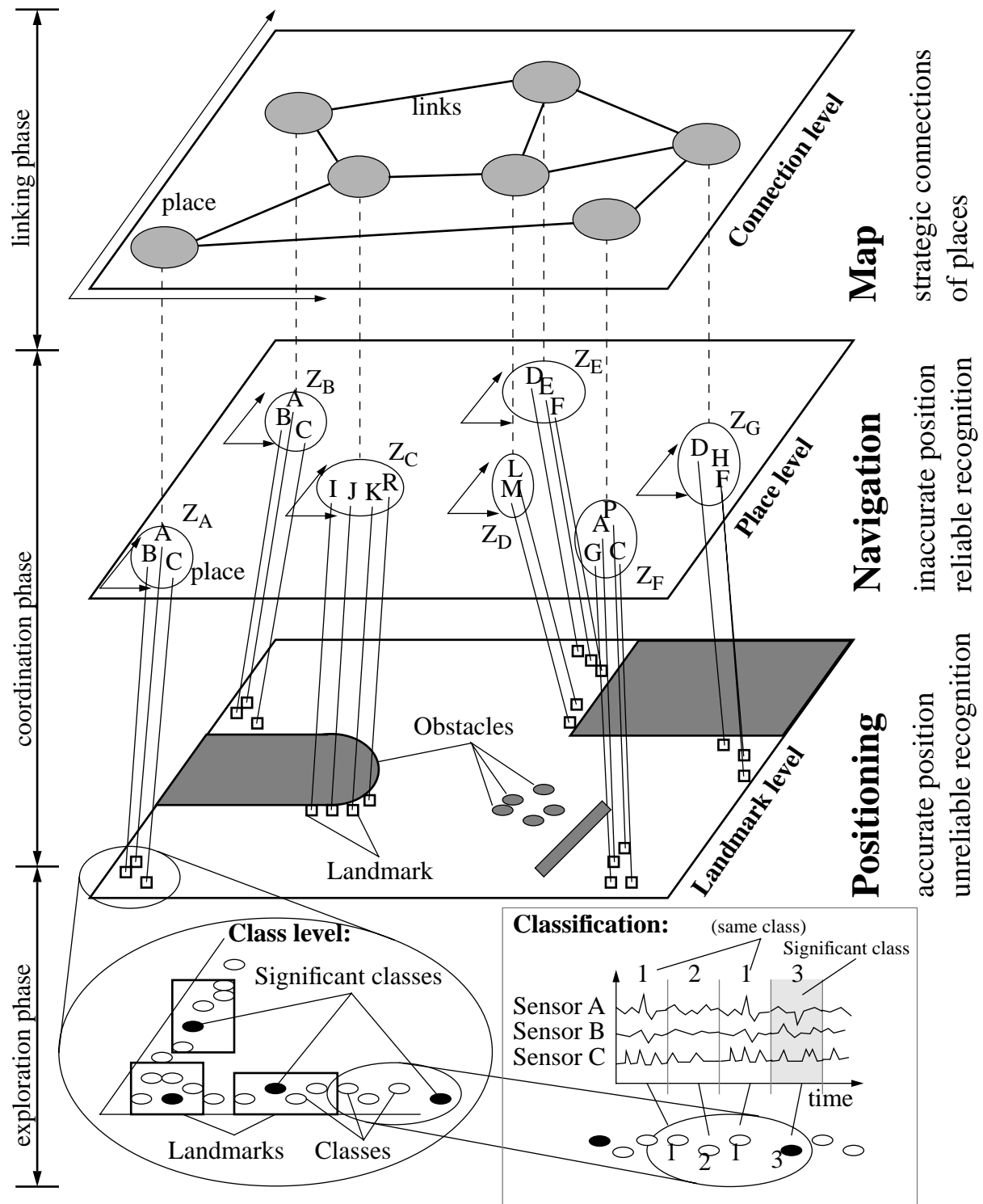


Fig. 3-31: Complete algorithm divided into levels and temporal phases

3.14.1 Interconnections between the different levels

The raw sensor signals are classified and examined for significant classes which are used to initiate landmarks. They are stored on the landmark level including the robot's estimated position. In case of reactivation of the same landmark, this mentioned position is used to calibrate the robot's odometry.

An accumulation of several landmarks (see A, B, C) define a place (Z_A). Place recognition is done by Markov chain (see chapter 3.11 "Place identification by Markov Chain", page 66) and therefore more efficient because recognition does not depend on reliable recognition of a single landmark. The place position corresponds to the average landmark position and is therefore not stable. It is only used to indicate the approximate spatial relationship between places. These relationships (links) are stored in the connection level and represent experienced trajectory between places.

This step by step learning, proceeding through interactions between an agent and its environment, and leading to increased performances, is akin to the development of intelligence in infants, such as described by the psychologist Jean Piaget [Piaget J., 1970]. Corresponding to this, the levels used in this experiment can be compared with the classification in biology. The following comparison gives an idea of the system similarity without claiming a real dependence:

- In the "*exploration-phase*", the preprocessing part of nerve signals adapts to the capacity of the sensor. This means, for example, that the preprocessing part of the eyes can recognize a straight line, even if the picture on the retina is far away from a straight line. This adaptation of the visual preprocessing part is very flexible, so turning over the visual view by wearing special glasses will be compensated.
- In the "*coordination-phase*" sequences or dependences on events will be recognized in a fortuitous manner. For example, the ability to touch and feel obstacles demands coordination of action and perception which will be learned from experiences.
- In the "*strategy-phase*" the learned sequences will be linked together in order to reach an aim. For example, the learned ability to control the movement of legs will be combined in order to walk, or in order to grasp an object, the hand, arm and body have to move together.

Of course, this division of the learning strategy is very abstract and cannot be observed in nature in such a divided way.

3.14.2 Learning phases

Acquiring “classes”, “landmarks” and “places” while exploring randomly the environment cannot be done simultaneously; it implies a three-phase strategy of which timing is shown in fig. 3-32. Their starting are shifted and they never completely stop.

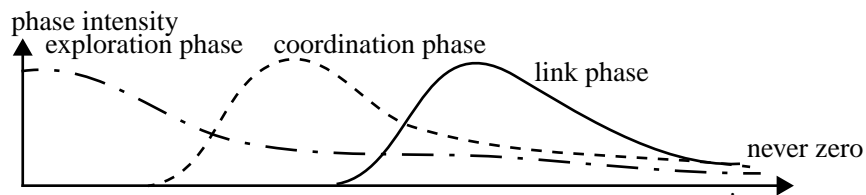


Fig. 3-32: Transition of the learning phases

The meaning of the phases is explained below. Please refer to fig. 3-31 for better understanding:

- 1) In the *exploration phase*, the robot moves aimlessly in its environment, gaining experience about its sensors. The signal events (see Sensor A-C in fig. 3-31) are classified and most landmarks are created. The classifiers (consisting of several neural networks) become more and more rigid, which stabilizes the classes in the sensor input space. After a certain time, the classes are rigid enough to allow the *coordination phase* to be started. However, the robot will never completely quit the *exploration phase* in order to stay adaptive to new nonclassified sensor events.
- 2) In the *coordination phase*, accumulated landmarks are grouped into places (in fig. 3-31 landmarks I, J, K and R become place Z_c). During this phase, the mutual positions of landmarks inside the same place become stabilized, independently of the global *place* position itself. A typical place contains about 20 landmarks, but only about 5 are discovered for each pass. As before, the *coordination phase* will never completely quit so that the robot can stay adaptive to future landmark changes.
- 3) In the *linking phase*, the places are linked together, which creates a topological map. The links results from the allowed movements of the robot (see chapter 3.8 "Constraining robot movements", page 62). Each link contains a weight, which is reinforced after each use. These reinforced links allows to predict a future place and is an indicator for the reliability of the robot's odometry.

3.15 Experimental results

This section describes the experiment results of this chapter obtained.

3.15.1 Landmark distribution

The described algorithm was tested on a simulated robot environment containing static obstacles (see fig. 3-33). The freedom is reduced to some paths showed by gray lines in the sketch on the right side. During this constrained movement about 150 landmarks are extracted from the camera, compass and distance sensor signals. The reason why some spots are chosen as landmarks is not always intuitive for a human. Some of them were generated as the robot was located in corners which activated more distance sensors than usual. Others were generated by the appearance of objects in the camera view.

Sometimes the opposite question comes up. Why is this position not recognized as a landmark? The reason therefore is often an unsuitable sensor which is not able to recognize the for human so obvious stimulus. This unsimilar world representation confirms the striven aim to take fully advantage of the robot's sensor ability unconsidering the human world representation. On the other hand, communication with the robot (i.e. task order) can not be performed if there is no minimal common language (world representation). It seems, that the robot needs a certain minimal supervisor for landmark definition in order to establish a common definition for communication between these two worlds.

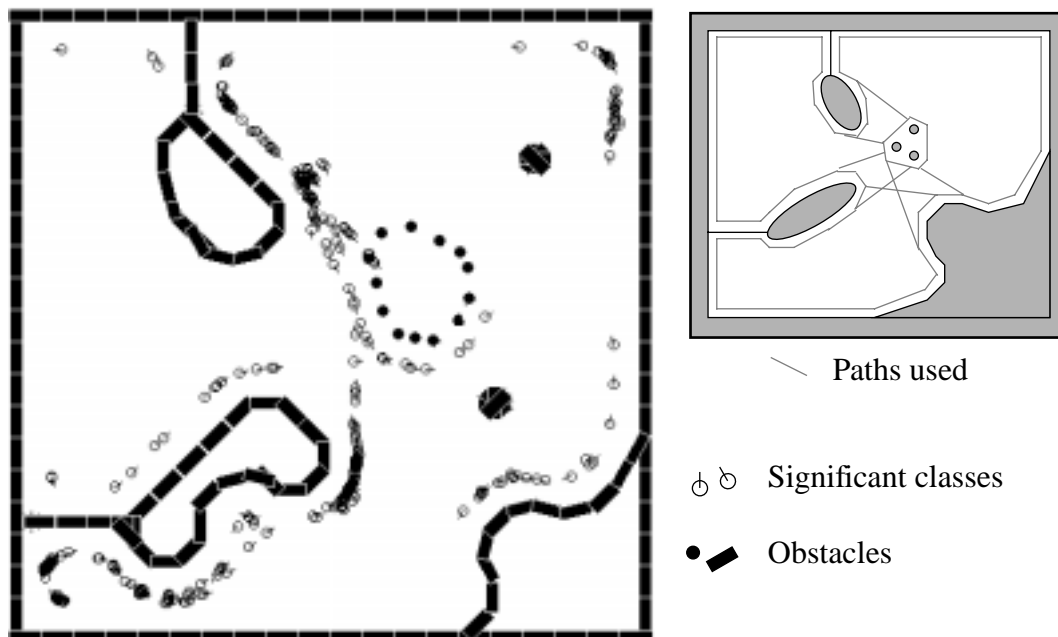


Fig. 3-33: Robot environment with all created landmarks from all classes

Fig. 3-33 shows all significant classes used by all landmarks (which can contain several landmarks) and gives therefore an overview which kind of situation the robot judge as interesting. Fig. 3-19 on page 55 shows significant classes as filled circles, which correspond to the \odot in figure fig. 3-33 above. The reason for the occasionally high concentration is that several classifiers produce a significant class at the same time (or just one behind the other), which will lead to one landmark, but however produces the overcrowd representation. Each landmark is stored with an associated estimated position which will be used to calibrate the robot position while it is running. Every time the robot rediscovers a landmark on a plausible position, it corrects its odometry, setting it to the position which was associated to the landmark. The result is an odometry error which it will not cross over a certain boundary.

3.15.2 Odometry error correction

This subsection shows the efficiency of the odometry error correction. The lower, filled graph in fig. 3-34 shows the absolute odometry error (maximum peak is about a third of the robot's diameter). There is always a small absolute position error which never exceeds a certain limit if regularly some landmarks can be found. This limit describes the minimal distance between two identical landmarks and is an estimated parameter of about half the robot's diameter. The upper graph shows the correction made by the rediscovered landmarks (same scale). Every positive peak of the curve means a right identification and therefore decreases the odometry error. Notice that there are also wrongly identified landmarks indicated by negative peaks. The reason for such faulty updates are several identical landmarks inside the described minimal distance. Such mistakes increase the odometry error, but they are quickly compensated.

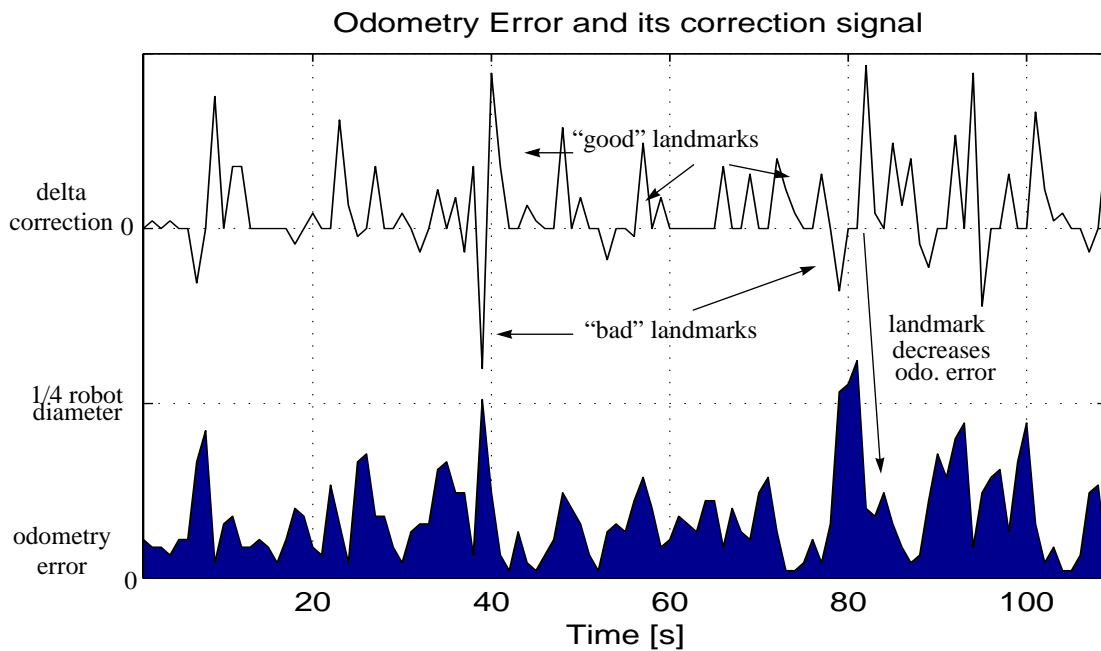


Fig. 3-34: Stabilized error due to landmark synchronization

3.15.3 Place production

Fig. 3-35 shows the number of places as a function of calculation iterations (1000 iterations per unit corresponds to about one minute). It can be seen that the number of places first grows fast and then becomes more and more flat. Place production will never completely stop. Due to the absence of a supervising model, noise and sensor errors can always be interpreted as a new landmark which can lead to a new place.

The growing number of landmarks can be compensated by two different methods:

- The first method consists in freezing the classifier which stabilized the learned landmarks. However, there is a risk of freezing the classifier too early which makes any future learning impossible. Another disadvantage is that this approach excludes dynamical environments as well.
- Another method is to use “garbage collection”, which means erasing old landmark entries. This approach avoids the problems mentioned above, but demands an extensive data management (see chapter 4 “Discussion and conclusion”).

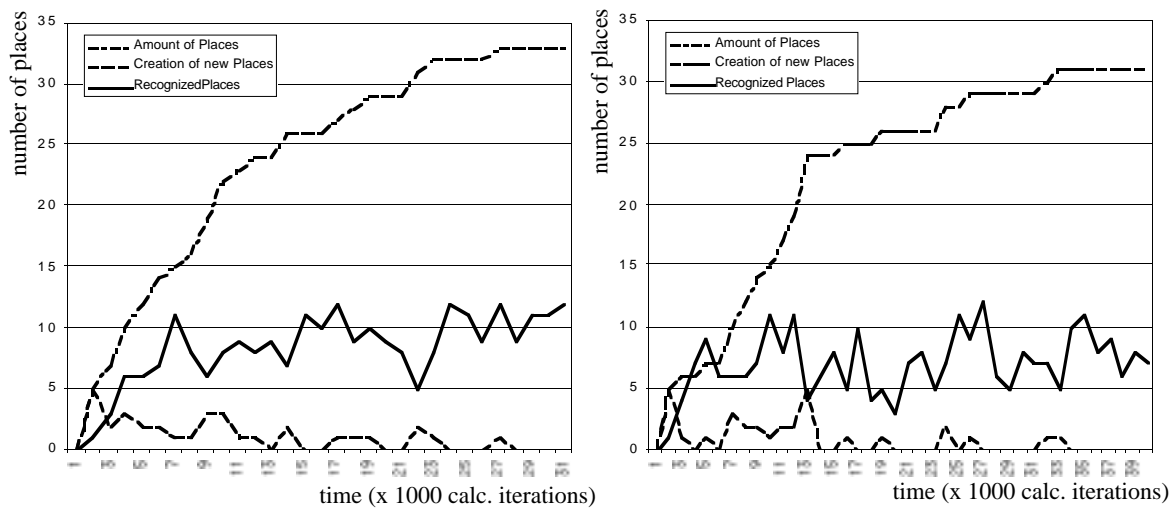


Fig. 3-35: Place evolution in time

3.15.4 Structured environment

The algorithm works quite well in disordered environments, due to the (often unique) diversity of the sensor input. On the other hand, the information variance of *structured environments* is often too little to guarantee a reliable operation of the algorithm. Fig. 3-36 shows the class production of a *structured environment* which contains almost as many classes as shown in fig. 3-33. However, they are of worse quality. The reason therefore is that the structured environment contains many rightangle corners, which make their corresponding class no longer significant and will hardly ever be used as landmark. Therefore, the classes located close to corners (see fig. 3-36) represent slight deviations of the “typical” corner class, this means they are slightly changed by noise and other influences.

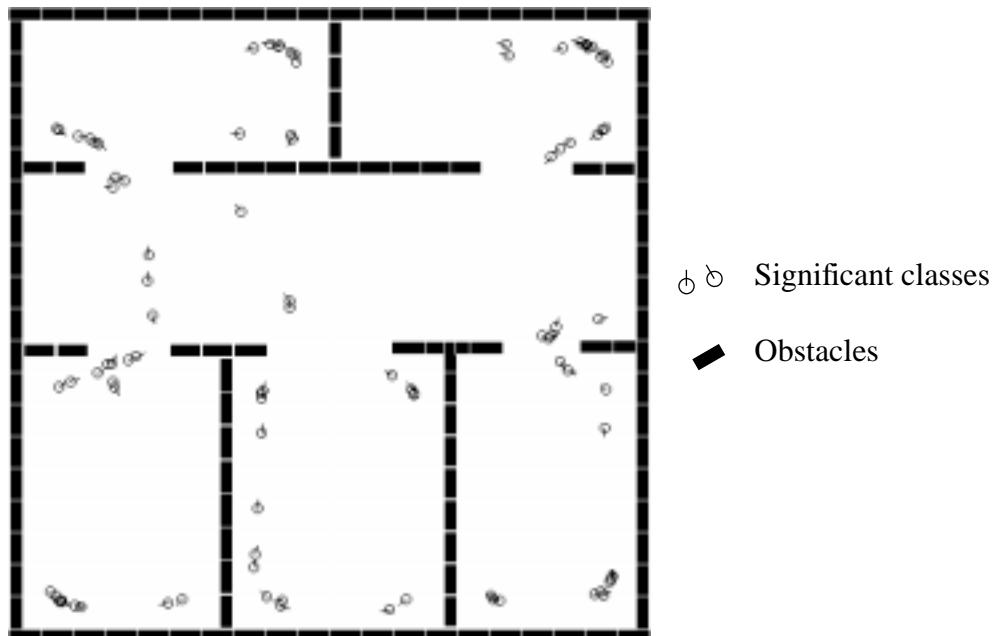


Fig. 3-36: Structured environment leads to little and less reliable class production

The recognition of such classes is therefore less reliable, affecting the landmark and the place recognition algorithm. Fig. 3-37 shows the efficiency of the *odometry error correction algorithm* for a *structured environment* (compare to fig. 3-34). The lower graph indicates the absolute position error and the upper graph the correction made by landmark recognition.

A positive peak of one horizontal line unit means a recognition of a “good” landmark decreasing the odometry error; a negative deflection indicates the recognition of a “bad” landmark which increases the odometry error. It can be observed, that there are noticeable less correction events than compared to fig. 3-34.

After a certain time, the odometry error exceeds the limit of one robot diameter due to missing landmarks. No further corrections can be proceeded because rediscovered “good” landmarks, with an assumed distance of more than one robot diameter, are pronounced as unlikely and therefore ignored. The system lost its position and needs to be externally calibrated.

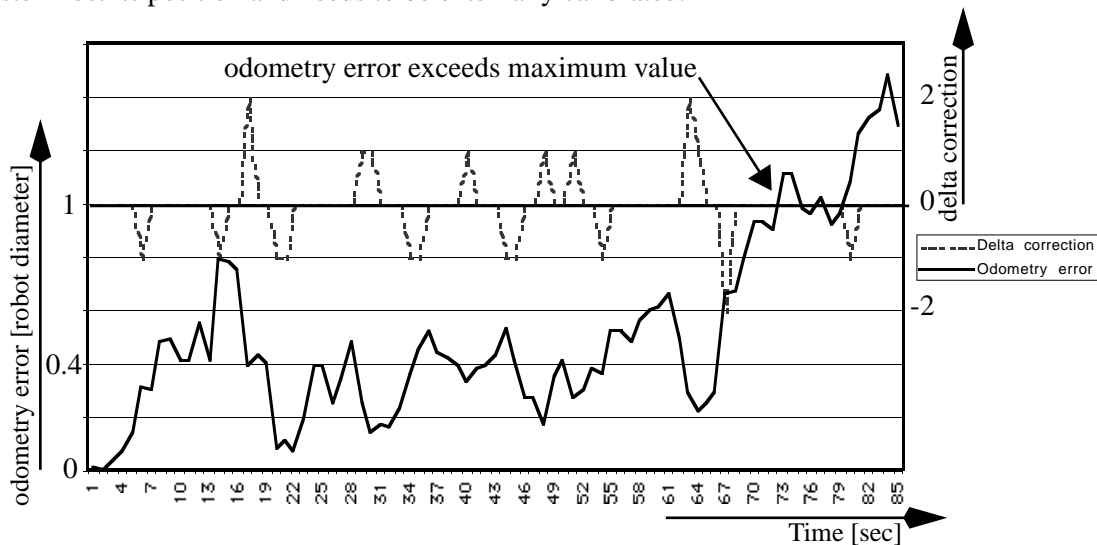


Fig. 3-37: Landmarks synchronization is less stable and fails after a certain time

It is very difficult to predict which environment type allows a continuous and stable *odometry correction* (as shown in fig. 3-33) or which environment will fail (fig. 3-36). It depends not only on the diversity of the environment but also of the sensor and odometry quality of the robot. The experiment demonstrated that different environments influence more the *quality of landmarks* (number of rediscoveries) than the *number of landmarks*. However, the number of *rediscovered landmarks* is not suitable as an indicator for the environment quality because “good” and “bad” landmarks cannot be distinguished during the process.

4 Discussion and conclusion

This thesis showed the successful implementation of two original positioning systems, correcting the robot's odometry by supervised and unsupervised algorithms. Both algorithms meet the same objective, that is to require as few as possible environmental changes (adding artificial landmarks) or supplementary information (about natural or artificial landmark features). Their sensor configuration is minimal:

- The supervised positioning system in chapter 2 uses only one light sensor underneath the robot and a mathematical model of the positioning probability distribution. More sensors or sensors of better quality would make the model much simpler. A new position probability distribution for a two wheeled robot was developed and allowed the examination of the odometry error behavior (drift) of such robots.
- The unsupervised positioning system in chapter 3 uses no external help and no knowledge about the environment or sensor configuration. The experience shows that a robot can extract and recognize suitable landmarks from pure sensor signals without any knowledge about the environment or its sensor configuration. This allows stabilization of the odometry error within a certain boundary. Association of landmarks generate concepts comparable to very basic human concepts such as "chairs" or "doors". The autonomous and unsupervised creation of concepts takes better advantage of the sensor capabilities and is adaptive to small changes in the environment. However, the proposed architecture is designed for static environment.

Since the detailed discussion about the first experiment is given in section 2.7 on page 34, we will focus here on the second experiment which brings a contribution to adaptive robotic research. Leaving the choice of suitable landmarks to the robot is a promising way to optimize the use of their sensors and take advantage of neural network plasticity. However, the following four critical points should be kept in mind:

- **Common language**

The map developed by a robot according to the approach presented will never equal the one humans would draw. A common language remains to be established if the user wants to describe a path, point a target, describe a task, etc. to the robot.

A possible solution would be to guide the robot to the target point in order to make it learn this position. However, there is a risk that the robot will not judge the destination point as interesting, preventing the robot from storing it. Therefore it seems that the robot has to be supervised for landmark definition in order to establish a common definition for the same landmark (see section 3.15.1 on page 75).

- **Choice of landmarks**

The learned landmarks should present a full set of all previously encountered and future situations. This aim is difficult to achieve because slight changes of the environment such as illumination can completely change the robot's world perception. The same environment can be suddenly appear unrecognizable after such a slightly change. On-line learning systems (such as neural network) are always challenged by this kind of danger. This is in contrast to batch learning systems which can, in principle, encounter the full set of all possible input vectors.

- **Convergence**

Due to the absence of a model defined by a supervisor, the robot will never stop recovering new significant events in the environment. The number of discoveries will decrease but never stop, because of noise and sensor errors.

One solution is to freeze the learned landmarks after a certain time. Since the movement is random, there is always a risk of freezing the network too early, which would ignore new unexplored zones. However, this solution would limit the system to static environments.

Another solution is to erase landmarks which have not been used any more for some time (garbage collection). This technique avoids the disadvantage mentioned above. The main problem of this approach is the fact that landmarks are defined by classes which are stored in a neural network and therefore have to be deleted as well. Erasing clusters in a feed forward network like K-means can be easily performed because the activation of a cluster is defined by the directly connected weights. On the other hand, the activation of a cluster in an ART (Adaptive Resonance Theory) network is influenced by all weights due to its oscillation of information. Therefore deleting a cluster influences the activation of all other clusters. The resulting consequences are difficult to foresee.

- **Multi-modal versus single-modal classification (fusion of different sensor types)**

Combining all types of sensor inputs into one data stream for unsupervised classification gives much flexibility, but makes it much harder to take advantage of specific characteristics of a single sensor type. Especially the combination of linear and nonlinear sensors makes a sensor specific classification almost impossible. The paper presented by Virginia R. de Sa and Dana H. Ballard [de Sa, Ballard, 1998] tries to avoid this problem by detecting "interesting events" in the combination of all sensors types, but classifying each single sensor type separately. This allows a better adaptation of the neural network to specific sensor characteristics. On the other hand, features which are coded by the relationship between different sensor types cannot be discovered. Fig. 4-1 shows a basic example of correlating the right and the left distance sensors (same sensor type) of a robot in order to detect significant events (corridor). Only a correlated classifier can detect the corridor as a significant situation.

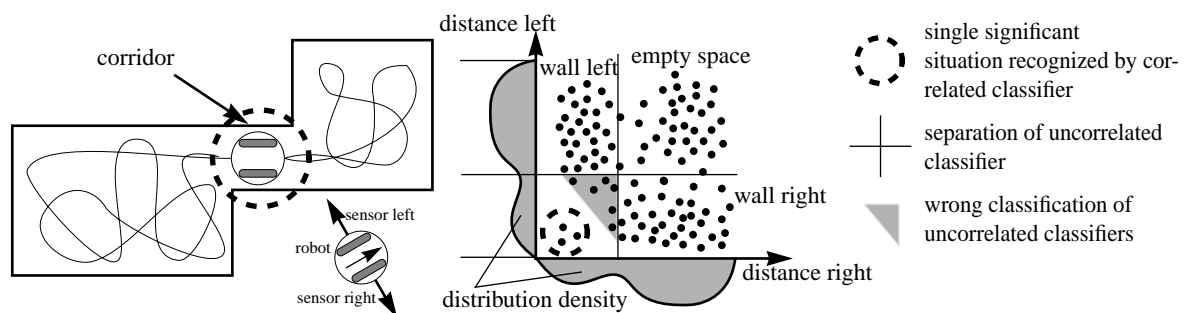


Fig. 4-1: Some significant situations can only be recognized by sensor correlation

This thesis shows that a robot is able to extract significant reference points from an unknown environment without any knowledge of its sensor configuration. Due to the unsupervised classification, the extracted reference points are well adapted to the sensor abilities and to the stimulus supplied by the environment. The reference points were used to correct the odometry error and enable the robot to follow its trajectory on a topological map.

4.1 Future work

A robot creating its own world representation is the dream for many researcher. The large variety of different worlds makes this subject interesting for further investigations in several topics, for example: How far can such an unsupervised system learn and usable interpret its environment without any supervisor? Is the advantage of an agent depending world representation bigger than the effort which has to be done for the creation of an autonomous world presentation? Is a further development of such an autonomous systems still controllable? Is there a method to build some bridges between these two words for minimal communication and task control?

Two concrete topics would be a logic continuation of this work:

- The topological map was not used for path planning but this would be an interesting and demanding future task, because of the different and unforeseeable reliability of the reference points. However, the critique mentioned in the conclusion above has to be considered. Further investigations would be necessary in order to find some common ground on which to base the world perception of the robot and the user. This common ground must not affect the liberty and independency of the robot to build it's own world perception.
- The principal idea could also be used to improve existing navigation systems by recognizing and identifying suitable (natural) landmarks for the robot sensor configuration already used. This information can later be used to develop to perfection the positioning system by improving the already existing landmarks instead of installing new artificial ones. Such kinds of environment analysis could for example point up that moving some tables drastically improves the uniqueness of a place, which makes it possible to perform robot positioning without using special sensors and landmark systems.
- A further suggestion for future work is to support the algorithm by an active pilot such as described in [Recce & Harris, 1998]. The assumption is that an agent, initially placed in an unknown environment, starts to explore its close surrounding and periodically returns to the point of departure in order to create an egocentric map. The advantage of this exploration strategy is that the odometry error can be kept small and consistent for the explored zones which are close to the point of departure. After sufficient knowledge of the closer surrounding, the exploration zone can be gently expanded until the whole environment is covered.

5 Appendix

This appendix contains some practical implementation tips and other hints in form of program source codes written in C and Mathematica. Although the general attitude to omit source code in theses, I added these lines to support other people working on the same field. The fact that I couldn't find any implementation tips in literature or on the WEB reaffirmed this decision.

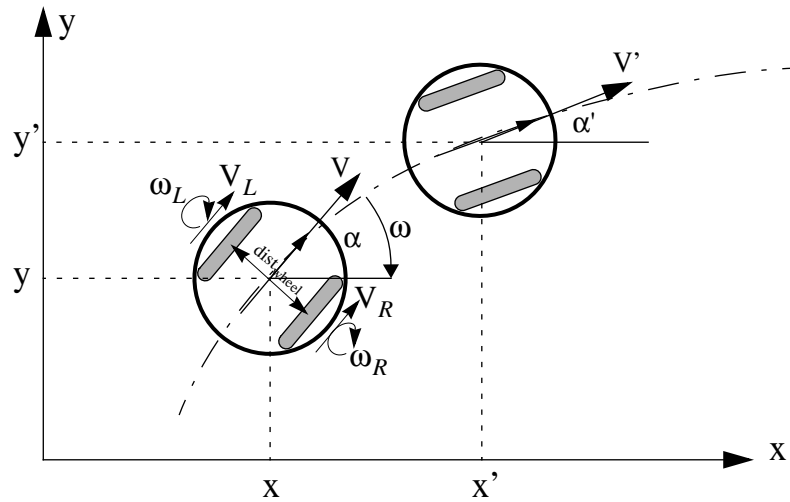
5.1 Different ways to calculate the odometry	84
5.2 Levenshtein simulators on the WEB	86
5.3 PPD simulation with Mathematica	87
Download: ftp://lamiftp.epfl.ch/pub/maechler/these/ppdsim.m	
5.4 Simplified Fuzzy ARTMAP	89
Download: ftp://lamiftp.epfl.ch/pub/maechler/these/fuzzyartmap.c	

5.1 Different ways to calculate the odometry

This section describes, besides the traditional method, a vector approach to calculate the odometry of an autonomous robot which doesn't use an angle α to describe the robot direction. The vector algorithm is more straightforward and avoids the problem of jumping from 360° to 0° and vice versa. This non-linearity can spoil the precision of calculation and demands special case treatment for route calculation. For example, the calculation of the *correction angle* $\Delta\alpha$ (between -180° and $+180^\circ$) to reach the *aim direction* α_{aim} for a robot with the *present direction* $\alpha_{present}$ demands 4 different case treatments and can't be calculated in one formula¹. More details about odometry calculations can be find in [Campion G., 1996] and [Canudas C., 1992].

5.1.1 Odometry calculation describing the direction of the robot by an angle

Fig. 5-1 gives a very rough overview about the geometrical relationship of a traditional odometry calculation approach.



The rotation speed is proportional to the speed difference of the two wheels:

$$\omega = \frac{V_L - V_R}{dist_{wheel}} = \frac{r_{wheel} \cdot (\omega_L - \omega_R)}{dist_{wheel}}$$

Forward speed is proportional to the average wheel speed:

$$V = \frac{v_L + v_R}{2} = \frac{r_{wheel}}{2} \cdot (\omega_L + \omega_R)$$

$$\left. \begin{array}{l} x' = x + V \cos \alpha \cdot \Delta t \\ y' = y + V \sin \alpha \cdot \Delta t \\ \alpha' = \alpha + \omega \Delta t \end{array} \right\} \left. \begin{array}{l} \Delta x = V \Delta t \cos \alpha \\ \Delta y = V \Delta t \sin \alpha \\ \Delta \alpha = \omega \Delta t \end{array} \right\} \begin{bmatrix} \dot{x} \\ \dot{y} \\ \dot{\alpha} \end{bmatrix} = \begin{bmatrix} V \cos \alpha \\ V \sin \alpha \\ \omega \end{bmatrix}$$

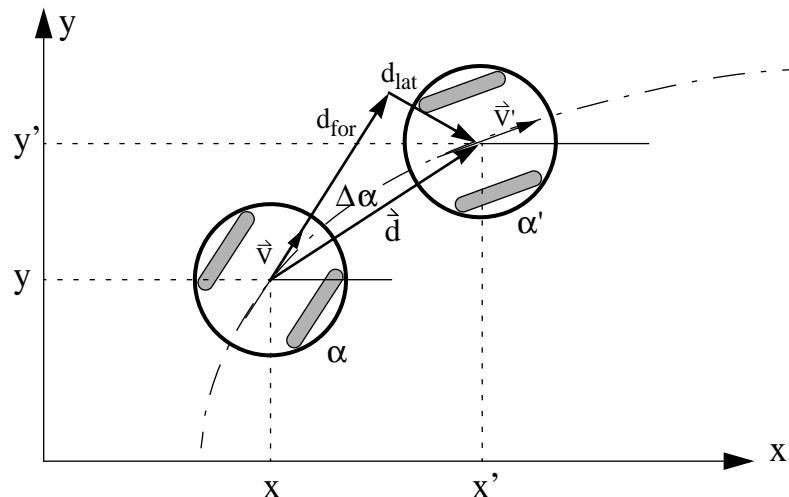
Fig. 5-1: Rough overview of the traditional approach for odometry calculation

1. Four cases to calculate the $\Delta\alpha$ from α_{aim} and $\alpha_{present}$: $\Delta\alpha = \begin{cases} \alpha_{present} - \alpha_{aim} \\ \alpha_{aim} - \alpha_{present} \\ 360^\circ - (\alpha_{present} - \alpha_{aim}) \\ 360^\circ - \alpha_{aim} - \alpha_{present} \end{cases}$

5.1.2 Odometry calculation describing the direction of the robot by a vector

A normalized vector can be used in order to avoid the non-linearity of odometry calculation by using the robot direction α . This idea is not new, but less intuitive and therefore rarely used.

First, the forward d_{for} and lateral d_{lat} displacement is calculated based on the turning angle and radius which is calculated in a similar way as described in the section before. Fig. 5-2 shows how the displacement vector \vec{d} can be calculated by the normalized direction vector \vec{v} stretched by d_{for} added with the perpendicular normalized vector $\perp\vec{v}$ stretched by d_{lat} . Fig.



$$\vec{d} = \vec{v} \cdot d_{for} + \perp\vec{v} \cdot d_{lat} \quad \perp\vec{v} = \begin{bmatrix} \cos(90) & \sin(90) \\ -\sin(90) & \cos(90) \end{bmatrix} \vec{v} = \begin{bmatrix} 0 & 1 \\ -1 & 0 \end{bmatrix} \vec{v}$$

$$\Delta x = (x' - x) = V_x d_{for} + V_y d_{lat} \quad \Delta y = (y' - y) = V_y d_{for} - V_x d_{lat}$$

Fig. 5-2: Displacement calculation based on the normalized direction vector of the robot

The following program written in C shows the kernel of the odometry routine:

```
// Global variables: position_x, position_y, norm_vec_x, norm_vec_y
void odometry(double delta_left_speed, double delta_right_speed)
{
    double angle, radius, forward, lateral, help_norm_vec_x, help_norm_vec_y;
    if (delta_left_speed != delta_right_speed) /* Robot is turning */
    {
        /* divide by dist between wheels measured in increments */
        angle = (double)(delta_left_speed - delta_right_speed) / KHEP_WHEEL_DIST;
        radius = (KHEP_WHEEL_DIST / 2) * (delta_left_speed + delta_right_speed) /
            (delta_left_speed - delta_right_speed);
        forward = radius * sin(angle);
        lateral = radius * (1.0 - cos(angle));
    }
    else /* Robot moves straight ahead */
    {
        angle = 0.0;
        forward = delta_left;
        lateral = 0;
    }
    position_x += forward * norm_vec_x + lateral * norm_vec_y;
    position_y += forward * norm_vec_y - lateral * norm_vec_x;
    /*
    \begin{matrix} x \\ y \end{matrix} = \begin{bmatrix} \cos(a) & \sin(a) \\ -\sin(a) & \cos(a) \end{bmatrix} * \begin{bmatrix} x0 \\ y0 \end{bmatrix}
    */
    help_norm_vec_x = norm_vec_x;
    help_norm_vec_y = norm_vec_y;
    norm_vec_x = cos(angle) * help_norm_vec_x + sin(angle) * help_norm_vec_y;
    norm_vec_y = -sin(angle) * help_norm_vec_x + cos(angle) * help_norm_vec_y;
}
```

Thanks to Laurent Tettoni for his advice ...

5.2 Levenshtein simulators on the WEB

In many applications, it is necessary to determine the similarity of two strings. A widely-used notion of string similarity is the Levenshtein (or edit) distance: the minimum number of insertions, deletions, and substitutions required to transform one string into the other. More information can be found on under <http://www.cs.princeton.edu/~ristad/papers/ps-532-96.html>.

There are some very good and instructive Levenshtein simulators on the web. Therefore it is worth to mention it in this section:

- http://www.cut-the-knot.com/do_you_know/Strings.html
Very nice Java applet which measures the Levenshtein distance between two strings. The applet is very intuitive and can even compare the result between Levenshtein and Hamming distance.
- <http://www.cedar.buffalo.edu/Linguistics/Forms/Strmatch.html>
This simulator allows to experiment with an extended version of the Damerau-Levenshtein string similarity metric. It matches a user supplied corrupted string with a lexicon of 21299 words.
- <http://lithwww.epfl.ch/~fbuchs/Leven/src/publ/Grid.html>
This Java applet performs pattern classification of numbers based on Levenshtein. The Java applet offers a graphical user interface to draw a number which will be recognized.

5.3 PPD simulation with Mathematica

```

(* ===== *)
(*          PPD SIMULATOR FOR THE KHEPERA ROBOT          *)
(* ===== *)
(*          September 1996 at LAMI-DI EPF Lausanne by Philip Maechler          *)
(* ===== *)
(*          Tips:                                          *)
(* - Never use var. names starting with capital letter or containing '_'! *)
(* - Type '<<filename' to start the script in Mathematica. *)
(* - Don't stop a turning calculation or you have to restart the kernel. *)
(* - Un-Tab this file before exporting to a text processor *)
(*          Last change: September 11, 1996              *)
(* ===== Variable definitions ===== *)
fieldDimension=200; (* Size of the 2-dimensional array field in pixels *)
v1=100.0;           (* speed of the left wheel (1) in mm/sec *)
v2=100.0;           (* speed of the right wheel (2) in mm/sec *)
base=52;            (* wheelbase. Wheel distance in mm (Khepera 52mm) *)
slip=1;             (* slip of the wheels in +/-% *)
time=40.0;          (* experiment duration in sec. *)
iterationStep=1;    (* iteration steps for the calculation (default=1) *)
resolution=10;     (* # of calculation for each dimension *)
timeSnapShot=10;  (* snapshot of the LPF every x seconds 0=off *)
turnTime1=20; turnAngle1=180 Degree //N (* Turn: timedelay 0=off *)
turnTime2= 0; turnAngle2=-90 Degree //N (* Turn: timedelay 0=off *)
turnTime3= 0; turnAngle3= 90 Degree //N (* Turn: timedelay 0=off *)
startAngle=0 Degree //N; (* start angle direction of the robot *)
modeTraceWay=False; (* mark the covered way by the robot on the floor *)
modeTraceLine=True; (* mark only the covered trajectory if slip=0 *)
modeRecursiveDirect=True; (* Recursive or direct calculation mode *)
modeOnlyCorner=False; (* show not the whole PPD, only 4 corners *)
(* ===== Initialization ===== *)
Clear[field];
field=Table[ 0, {fieldDimension}, {fieldDimension}];
If[modeRecursiveDirect, modeTraceWay=False];
borderZmax=0;
borderXmin=100000; borderXmax=-100000; borderYmin=100000; borderYmax=-100000;

(* ===== plotPoint ===== *)
(* plotPoint sets a point in the field array *)
(* modeCalibrate : =True: just adapt border limits / =False: set point *)
(* modeFixOrInc : =True: the height of the point is fix or increment =False *)
(* value : value for the fixed point or for the increment steps *)
Clear[plotPoint];
plotPoint[posX_, posY_, modeCalibrate_, modeFixOrInc_, value_] := Block[
  {arrayX, arrayY}, (* local variables *)
  If[modeCalibrate,
    If[borderXmin>posX, borderXmin=posX,];
    If[borderXmax<posX, borderXmax=posX,];
    If[borderYmin>posY, borderYmin=posY,];
    If[borderYmax<posY, borderYmax=posY,];
  ], (*else*)
  arrayX=Round[(fieldDimension-4)*(posX-borderXmin)/borderWidth]+2;
  arrayY=Round[(fieldDimension-4)*(posY-borderYmin)/borderWidth]+2;
  If[(arrayX>=1) && (* Boundary check *)
    (arrayY>=1) &&
    (arrayX<=fieldDimension) &&
    (arrayY<=fieldDimension)
  ],
  If[modeFixOrInc,
    If[field[[arrayY, arrayX]]<value, field[[arrayY, arrayX]]=value,];
  ], (* Else *)
  field[[arrayY, arrayX]]+=value;
]; (* EndIf *)
If[field[[arrayY, arrayX]]>borderZmax, borderZmax=field[[arrayY, arrayX]],];
(* Else *)
Print["*** ERROR in function plotPoint *** Point out of range:"];
Print["field index: x=", arrayX, "; y=", arrayY, " dV1=", runV1, " dV2=", runV2];
]; (* EndIf *)
]; (* EndIf modeCalibrate *)
]

(* ===== recursiveCalc ===== *)
Clear[recursiveCalc];
recursiveCalc[runV1_, runV2_, modeCalibrate_, modeTraceWay_] := Block[
  {help, runTime, angle}, (* local variables *)
  x=0.0; (* start point x *)
  y=0.0; (* start point y *)
  angle=startAngle; (* start angle *)
  For[runTime=iterationStep, runTime<=time, runTime+=iterationStep,
    help=((runV2 + (runV1-runV2)/2)*iterationStep);
    x+= ( (Sin[angle]) help ); (* new x position of the robot *)
    y+= ( (Cos[angle]) help ); (* new y position of the robot *)
    angle+=(runV1-runV2)/base //N; (* new direction of the robot *)

    If[runTime==turnTime1, angle+=turnAngle1,];
    If[runTime==turnTime2, angle+=turnAngle2,];
    If[runTime==turnTime3, angle+=turnAngle3,];

    If[timeSnapShot!=0,
      If[Mod[runTime, timeSnapShot]==0,
        plotPoint[x, y, modeCalibrate, False, 1];
      ], (* EndIf without else *)
    ], (* EndIf without else *)

    If[modeTraceWay,
      plotPoint[x, y, modeCalibrate, True, 1]
    ], (* EndIf without else *)
    If[modeTraceLine,
      If[(runV1==v1 && runV2==v2),
        plotPoint[x, y, modeCalibrate, True, borderZmax/3];
      ], (* EndIf without else *)
    ], (* EndIf without else *)
  ]; (* For *)
  plotPoint[x, y, modeCalibrate, False, 1];
]; (* Block *)
]

(* ===== directCalc ===== *)
Clear[directCalc];
directCalc[runV1_, runV2_, modeCalibrate_] := Block[
  {x, y}, (* local variables *)
  If[runV1!=runV2,

```

```

    x=(base/2+runV2*base/(runV1-runV2))*(1-Cos[(runV1-runV2)/base*time]);
    y=(base/2+runV2*base/(runV1-runV2))*Sin[(runV1-runV2)/base*time];
    ,(*else*)
    x=0;
    y=runV2*time;
    ];(*EndIf*)
    plotPoint[x, y, modeCalibrate, False, 1];
  ] (* Block *)
  (* ===== main function ===== *)
  For [runV1=v1-(v1/100*slip), runV1<=v1+(v1/100*slip), runV1+=(v1/100*slip*2)/resolution,
    If [modeRecursiveDirect,
      recursiveCalc[runV1, v2-(v2/100*slip), True, modeTraceWay];
      recursiveCalc[runV1, v2+(v2/100*slip), True, modeTraceWay];
    ], (*else*)
    directCalc[runV1, v2-(v2/100*slip), True];
    directCalc[runV1, v2+(v2/100*slip), True];
  ];(*EndIf*)
  ];
  For [runV2=v2-(v2/100*slip), runV2<=v2+(v2/100*slip), runV2+=(v2/100*slip*2)/resolution,
    If [modeRecursiveDirect,
      recursiveCalc[v1-(v1/100*slip), runV2, True, modeTraceWay];
      recursiveCalc[v1+(v1/100*slip), runV2, True, modeTraceWay];
    ], (*else*)
    directCalc[v1-(v1/100*slip), runV2, True];
    directCalc[v1+(v1/100*slip), runV2, True];
  ];(*EndIf*)
  ];
  If [modeTraceWay, plotPoint[0, 0, True, True, 1],]; (* consider start point in field *)
  If [borderXmax-borderXmin>borderYmax-borderYmin,
    borderWidth=borderXmax-borderXmin;
    borderYmin=(borderWidth-(borderYmax-borderYmin))/2;
    borderYmax=borderYmin+borderWidth;
  ], (*else*)
    borderWidth=borderYmax-borderYmin;
    borderXmin=(borderWidth-(borderXmax-borderXmin))/2;
    borderXmax=borderXmin+borderWidth;
  ];
  If [modeOnlyCorner,
    runV1=v1-(v1/100*slip);runV2=v2+(v2/100*slip);recursiveCalc[runV1, runV2, False, modeTraceWay];
    runV1=v1+(v1/100*slip);runV2=v2-(v2/100*slip);recursiveCalc[runV1, runV2, False, modeTraceWay];
    runV1=v1-(v1/100*slip);runV2=v2;recursiveCalc[runV1, runV2, False, modeTraceWay];
    runV1=v1+(v1/100*slip);runV2=v2;recursiveCalc[runV1, runV2, False, modeTraceWay];
    runV1=v1;runV2=v2+(v2/100*slip);recursiveCalc[runV1, runV2, False, modeTraceWay];
    runV1=v1;runV2=v2-(v2/100*slip);recursiveCalc[runV1, runV2, False, modeTraceWay];
    runV1=v1+(v1/100*slip);runV2=v2+(v2/100*slip);recursiveCalc[runV1, runV2, False, modeTraceWay];
    runV1=v1-(v1/100*slip);runV2=v2-(v2/100*slip);recursiveCalc[runV1, runV2, False, modeTraceWay];
  ], (*else*)
  For [runV1=v1-(v1/100*slip), runV1<=v1+(v1/100*slip), runV1+=(v1/100*slip*2)/resolution,
    For [runV2=v2-(v2/100*slip), runV2<=v2+(v2/100*slip), runV2+=(v2/100*slip*2)/resolution,
      If [modeRecursiveDirect,
        recursiveCalc[runV1, runV2, False, modeTraceWay];
      ], (*else*)
        directCalc[runV1, runV2, False];
    ]; (* EndIf *)
  ]; (* For *)
  ]; (* For *)
  ];(*EndIf*)
  If [modeTraceWay, plotPoint[0, 0, False, True, borderZmax],]; (* Mark start point *)
  ListContourPlot[field,
    Axes -> True,
    AxesLabel -> {"x-Axis (mm)", "y-Axis (mm)"},
    (*ColorFunction -> (GrayLevel[1-#*2]&),*)
    ColorFunction -> (GrayLevel[Which[##==0,1,
    #<0.15,0.5,
    #>=0.15,0]]&),
    ContourLines -> True,
    Contours -> 30, (* chooses n equally spaced contours between the maximum and minimum z values.*)
    ContourShading -> True,
    Frame -> False,
    Ticks -> {{1,borderXmin}, {fieldDimension/2,borderXmin+(borderXmax-borderXmin)/2//N},
    {fieldDimension,borderXmax}},
    {{1,borderYmin}, {fieldDimension/2,borderYmin+(borderYmax-borderYmin)/2//N},
    {fieldDimension,borderYmax}}
  ];
  ListContourPlot[field,
    Axes -> True,
    AxesLabel -> {"x-Axis (mm)", "y-Axis (mm)"},
    ColorFunction -> (GrayLevel[ If[##==0, 1, 0]]&),
    ContourLines -> True,
    Contours -> 30, (* chooses n equally spaced contours between the maximum and minimum z values.*)
    ContourShading -> True,
    Frame -> False,
    Ticks -> {{1,borderXmin}, {fieldDimension/2,borderXmin+(borderXmax-borderXmin)/2//N},
    {fieldDimension,borderXmax}},
    {{1,borderYmin}, {fieldDimension/2,borderYmin+(borderYmax-borderYmin)/2//N},
    {fieldDimension,borderYmax}}
  ];
  ListPlot3D[field,
    AxesLabel -> {"x-Axis (mm)", "y-Axis (mm)", ""},
    Ticks -> {{1,borderXmin}, {fieldDimension/2,borderXmin+(borderXmax-borderXmin)/2//N},
    {fieldDimension,borderXmax}},
    {{1,borderYmin}, {fieldDimension/2,borderYmin+(borderYmax-borderYmin)/2//N},
    {fieldDimension,borderYmax}},
    {1,borderZmax/2//N,borderZmax}},
    PlotRange -> {1,borderZmax},
    Mesh -> True,
    ClipFill -> None
  ];
  If [modeRecursiveDirect, Print["Recursive calculation"], Print["Direct calculation"]];
  Print["WheelSpeed left=", v1, "mm/sec right=", v2, "mm/sec; Slip=", slip, "%"];
  Print["Simulated duration =", time, "sec, iteration steps =", iterationStep, "sec."];
  Print["Visualisation field =", fieldDimension, "x", fieldDimension, " ; ", resolution, " values for each
  ΔV1 and ΔV2."];
  Print["Start point: 0,0 with ", startAngle/Pi*180//N, "Degree angle.];
  If [modeRecursiveDirect,
    If [turnTime1>0, Print["1. turn point after ", turnTime1, "sec. Correction angle=", turnAngle1/Pi*180//N,
    "Degree."],];
    If [turnTime2>0, Print["2. turn point after ", turnTime2, "sec. Correction angle=", turnAngle2/Pi*180//N,
    "Degree."],];
    If [turnTime3>0, Print["3. turn point after ", turnTime3, "sec. Correction angle=", turnAngle3/Pi*180//N,
    "Degree."],];
  ];(*EndIf without else*)

```

5.4 Simplified Fuzzy ARTMAP

```

/*****
/*                               Simplified Fuzzy ARTMAP                               */
/*                               from Tom Kasuba, described in AI Expert November 1993   */
/*****
#include <math.h>
#include <stdlib.h>
#include <stdio.h>
#include <float.h>
#define CATEGORY_NUMBER 2
#define INPUT_WIDTH 2
#define INPUT_2WIDTH (2*INPUT_WIDTH)
#define PATTERN_MAX 5
#define NODE_MAX 40
#define MIN(a,b) ((a)<(b)?(a):(b))
#define TRUE -1
#define FALSE 0
float vigilance, vigilance_baseline=0.4, epsilon=0.0001, alpha=0.000001;
float W[NODE_MAX][INPUT_2WIDTH], AND[NODE_MAX][INPUT_2WIDTH];
float NODE_ACTIVATION[NODE_MAX], NODE_MATCH[NODE_MAX];
int CATEGORY_USED[CATEGORY_NUMBER], NODE_CAT[NODE_MAX];
int _mismatch_vigilance=FALSE, _mismatch_category=FALSE;
int mode_used=0;
/*****
void createOutputNode(float *input, int supervisor)
{
    int i;
    if (mode_used+1>=NODE_MAX)
    { printf("No place for more nodes\007\n"); return;}
    printf("Creating new output node nr. %d for input: ", mode_used);
    for (i=0; i<INPUT_2WIDTH; i++)
    { W[mode_used][i]=input[i]; printf("%5.3f ", input[i]); } printf("\n");
    NODE_CAT[mode_used]=supervisor;
    mode_used+=1; /* Increment output node counter */
}
/*****
void outputNodeActivation(float *input)
{
    int i, node;
    float div_upper, div_lower;
    for(node=0; node<mode_used; node++)
    {
        for(i=0;i<INPUT_2WIDTH;i++)
            AND[node][i]=MIN(input[i], W[node][i]);
        div_upper=0.0;
        for(i=0;i<INPUT_2WIDTH;i++)
            div_upper+=AND[node][i];
        div_lower=0.0;
        for(i=0;i<INPUT_2WIDTH;i++)
            div_lower+=W[node][i];
        NODE_ACTIVATION[node]=div_upper/(alpha+div_lower);
        NODE_MATCH[node]=div_upper/INPUT_WIDTH;
    }
}
/*****
void display(void)
{
    int i, node;
    printf("OutputNodeActivation [%d]:", mode_used);
    for (node=0; node<mode_used; node++)
        printf("%5.3f ", NODE_ACTIVATION[node]); printf("\n");
    printf("Match function [%d]:", mode_used);
    for (node=0; node<mode_used; node++)
        printf("%5.3f ", NODE_MATCH[node]); printf("\n");
    printf("Vigilance :%f\n", vigilance);
    for (node=0; node<mode_used; node++)
    {
        printf("Top-Down weights [%d]:", node);
        for (i=0; i<INPUT_2WIDTH; i++)
            printf("[w=%5.3f a=%5.3f] ", W[node][i], AND[node][i]);
        printf("\n");
    }
}
/*****
void search_winner_node(int supervisor)
{
    int node_disabled[mode_used], node, node_win;
    for (node=0; node<mode_used; node_disabled[node++]=FALSE);
    for (;;)
    {
        node_win=-1;
        for (node=0; node<mode_used; node++)
            if (node_disabled[node]==FALSE)
                if (node_win==-1)
                    node_win=node;
                else
                    if (NODE_ACTIVATION[node]>NODE_ACTIVATION[node_win])
                        node_win=node;
        if (node_win==-1)
        { printf("!!! No node could be found. Create a new one...\n"); break; }
        printf("Winner node %02d: ", node_win);
        if (NODE_MATCH[node_win]<vigilance)
        { /* start if2 */
            mismatch_vigilance=TRUE; /* Suppress activation of the output node*/
            node_disabled[node_win]=TRUE;
            printf("!!! VIGILANCE MISMATCH! Match of node %d = %f (vig=%f). Try again...\n",
                node_win, NODE_MATCH[node_win], vigilance);
        } /* ende if2 */
        else

```

```

    { /* start else2 */
      if (NODE_CAT[node_win]==supervisor)
      { /* start if3 Winning output node encode the same category */
        int i;
        for(i=0;i<INPUT_2WIDTH;i++)
          W[node_win][i]=AND[node_win][i];
        _mismatch_vigilance=FALSE; /*there was no mismatch*/
        _mismatch_category=FALSE;
        printf("!!! Winner node %d found! Correspond to supervisor category %d. Weights addapted.\n",
          node_win, supervisor);
        break;
      } /* ende if3 */
      else
      { /* start else3 */
        vigilance=NODE_MATCH[node_win]+epsilon;
        _mismatch_category=TRUE; /* There is a category mismatch */
        node_disabled[node_win]=TRUE;
        printf("!!! CATEGORY MISMATCH! Node %d is not cat. %d.", node_win, supervisor);
        printf(" Set vigilance to %f. Try again...\n", vigilance);
      } /* ende else3 */
    } /* ende else2 */
  } /* ende for */
}
/*****
void training(float *input, int supervisor)
{
  if (CATEGORY_USED[supervisor]==FALSE) /* Never used this category */
  { /* start if1 */
    printf("\007*****Categorie was never seen... create a new one*****\n");
    createOutputNode(input, supervisor);
    CATEGORY_USED[supervisor]=TRUE;
  } /* ende if1 */
  else
  { /* start else1 */
    vigilance=vigilance_baseline;
    outputNodeActivation(input);
    search_winner_node(supervisor);
    display();
    /*****
    if ((_mismatch_vigilance==TRUE) || (_mismatch_category==TRUE)) /* Create an output node (if4)*/
      createOutputNode(input, supervisor);
    */
  } /* ende else1 */
  /*****
  printf("press RETURN to continue...\n"); getchar();
  */
}
/*****
int use(float *input)
{
  int node, node_win;
  outputNodeActivation(input);
  for (node=node_win=0; node<mode_used; node++)
    if (NODE_ACTIVITION[node]>NODE_ACTIVITION[node_win])
      node_win=node;
  return(NODE_CAT[node_win]);
}
/*****
void main(void)
{
  int i, pattern;
  float input[INPUT_2WIDTH];
  float INPUTS[PATTERN_MAX][INPUT_WIDTH], CATEGORY[PATTERN_MAX];
  for(i=0;i<CATEGORY_NUMBER;i++) CATEGORY_USED[i]=FALSE;
  INPUTS[0][0]=0.3 ; INPUTS[0][1]=0.3 ; CATEGORY[0]=0; /* inside */
  INPUTS[1][0]=0.7 ; INPUTS[1][1]=0.7 ; CATEGORY[1]=0; /* inside */
  INPUTS[2][0]=0.1 ; INPUTS[2][1]=0.1 ; CATEGORY[2]=1; /* outside */
  INPUTS[3][0]=0.9 ; INPUTS[3][1]=0.9 ; CATEGORY[3]=1; /* outside */
  INPUTS[4][0]=0.1 ; INPUTS[4][1]=0.8 ; CATEGORY[4]=1; /* outside */
  printf("=====\n");
  for(pattern=0; pattern<PATTERN_MAX; pattern++) /*start work loop*/
  {
    for(i=0;i<INPUT_WIDTH;i++)
      { input[i] = INPUTS[pattern][i];
        input[i+INPUT_WIDTH]=1-INPUTS[pattern][i]; }
    printf("%d. sample: ( " pattern);
    for(i=0;i<INPUT_2WIDTH;i++)
      printf("%5.3f ", input[i]);
    printf(") -----\n");
    training(input, CATEGORY[pattern]);
  }
  printf("RESULT:\n");
  for (i=0; i<mode_used; i++)
    printf("Node %d with weights [%5.3f,%5.3f] is allocated to category %d\n",
      i, W[i][0], W[i][1], NODE_CAT[i]);
  printf("Learning of the 5 pattern finished. Try now the network (CTRL-C to stop):\n");
  for (;;)
  {
    printf("Enter a point (x and y position between 0 and 1):");
    scanf("%f %f", &input[0], &input[1]);
    input[2]=1.0-input[0]; input[3]=1.0-input[1];
    printf("The pos(%f,%f) belongs to category %d\n", input[0],input[1], use(input));
  }
}

```

6 References

- [Abidi et al., 1992] Mongi A. Abidi, Rafael C. Conzalez, *Data Fusion In Robotics and Machine Intelligence*, Academic Press Inc., San Diego, 1992, 546 pages, ISBN 0-12-042120-8
- [Bauer R., 1995] Rudolf Bauer, *Active manoeuvres for supporting the localisation process of an autonomous mobile robot*, Robotics and Autonomous Systems Nr. 16, 1995, 8 pages (39-46)
- [Borenstein J., 1994] Johann Borenstein, *The CLAPPER: a Dual-Drive Mobile Robot with Internal Correction of Dead-reckoning Errors*, Proceedings of the 1994 IEEE International Conference on Robotics and Automation, San Diego, CA, May 8-13, 1994, pp. 3085-3090.
- [Borenstein J., 1995] Johann Borenstein, *Internal Correction of Dead-reckoning Errors With the Compliant Linkage Vehicle*, Journal of Robotic Systems, Vol. 12, No. 4, April 1995, pp. 257-273
- [Borenstein et al, 1996] J. Borenstein, H.R. Everett, L. Feng, *Where am I? Systems and Methods for Mobile Robot Positioning*, Wellesley, MA: A.K. Peters, March 1996, 282 pages, <http://www-personal.engin.umich.edu/~johannb/position.htm>
- [Borenstein, Feng, 1996] Johann Borenstein, Liqiang Feng, *Measurement and Correction of Systematic Odometry Errors in Mobile Robots*, IEEE Transaction on Robotics and Automation, Vol. 12, No. 6, December 1996, 12 pages
- [Brooks, 1993] Rodney A. Brooks, Lynn Andrea Stein, *Building Brains for Bodies*, MIT AI Lab Memo #1439, 15 pages, August 1993
- [Brown et al, 1992] C. Brown, H.F. Durrant-Whyte, J. Leonard, B. Rao, B. Steer, *Distributed Data Fusion Using Kalman Filtering: A Robotics Application*, In M. A. Abidi and R.C. Gonzalez *Data Fusion in Robotics and Machine Intelligence*, chapter 7, pages 267-309, Academic Press, 1992 [Abidi et al., 1992]
- [Braitenberg, 1984] V. Braitenberg, *Vehicles. Experiments in Synthetic Psychology*, MIT Press, Cambridge, 1984
- [Campion G., 1996] Guy Champion et al., *Structural Properties and Classification of Kinematic and Dynamic Models of Wheeled Mobile Robots*, IEEE Transactions on Robotics and Automation, Vol. 12, No. 1, February 1996
- [Canudas C., 1992] C. Canudas de Wit and O. J. Sørдалen, *Exponential Stabilization of Mobile Robots with Nonholonomic Constraints*, IEEE Transactions on Automatic Control, Vol. 37, No. 11, November 1992
- [Carpenter et al, 1987a] Gail A. Carpenter and S. Grossberg, *ART 2: Self-Organization of Stable Recognition Codes for Analog Input Patterns*, First IEEE Conference on Neural Networks, San Diego, California, June 21-24, 1987. In Conference proceedings, vol. II (1987), page 727-735

-
- [Carpenter et al, 1987b] Gail A. Carpenter and S. Grossberg, *ART 2: Self-Organization of Stable Category Recognition Codes for Analog Input Patterns*, Applied Optics, vol. 26 (1987), no. 23, page 4919-4930
- [Carpenter et al, 1991c] Gail A. Carpenter, S. Grossberg, D. Rosen, *Fuzzy ART: Fast Stable Learning of Analog Patterns by an Adaptive Resonance System*, Neural Network, 1991, Vol. 4, page 759-771
- [Catling I., 1994] I. Catling, *Advanced Technology for Road Transport: IVHS and ATT*, Artech House, Boston, MA, 1994
- [Collins L, 1975] Lyndhurst Collins, *An Introduction to Markov chain analysis*, Concepts and techniques in modern geography no.1, University of Edinburgh, Norwich, ISBN-0-902246-43-7, 1975, 36 pages
- [Crowley et al., 1993] J. L. Crowley, Y. Demazeau, *Prinziples and Techniques for Sensor Data Fusion*, Signal Processing, Vol. 32 Nos 1-2, pages 5-27, May 1993
- [Descartes R., 1637] René Descartes, Discours de la méthode, 1637, http://www.ambafrance.org/HYPERLAB/PEOPLE/_descart.html
- [Devy M., 1995] Michel Devy, *On autonomous navigation in a natural environment*, Robotics and Autonomous Systems Nr. 16, 1995, 12 pages (5-16)
- [Dubrawski et al, 1994] Artur Dubrawski, Patrick Reignier, *Learning to Categorize Perceptual Space of a Mobile Robot Using Fuzzy-ART Neural Network*, IROS'94, Laboratory of Adaptive Systems, Warsaw, Poland & Laboratoire d'Informatique Fondamentale et d'Intelligence Artificielle, Grenoble, France, 6 pages.
- [Duda & Hart, 1973] Richard O. Duda, Peter E. Hart, *Pattern Classification and Scene Analysis*, Stanford Research Institute, California, John Wiley & Sons, Inc, 1973, 480 pages, ISBN 0-471-22361-1
- [Durieu C., 1995] Cécile Durieu, *A Data Fusion Application for Location of a Mobile Robot Using an Odometer and a Panoramic Laser Telemeter*, Intelligent Autonomous System, 1995
- [Everett H., 1995] H. R. Everett, *Sensors for Mobile Robots*, A K Peters Ltd, Wellesley, Massachusetts, 1995, 530 pages, ISBN 1-56881-048-2
- [Fausett L., 1994] Laurene Fausett, *Fundamentals of Neural Networks; Architectures, algorithms, and applications*, Prentice-Hall International Inc., New Jersey, 1994, 440 pages, ISBN 0-13-042250-9
- [Floreano et al., 1996] Dario Floreano, Francesco Mondada, *Evolution of Homing Navigation in a Real Mobile Robot*, IEEE Transactions on Systems, Man and Cybernetics, Part B, Vol 26, Number 3, June 1996, pages 396-407.
- [Freriks L.W. et al, 1992] L.W. Freriks, P.J.M. Cluitmans, M.J. van Gils, *The Adaptive Resonance Theory Network: (Clustering-) Behaviour in Relation With Brainstem Auditory Evoked Potential Patterns*, Eindhoven University of Technology Report 92-E-264, November 1992, ISBN 90-6144-264-8, 102 pages
- [Fritzke B., 1997] Bernd Fritzke, *Some Competitive Learning Methods*, Inst. for Neural Computation, Ruhr-Universität, Germany, 1997, <http://www.neuroinformatik.ruhr-uni-bochum.de/ini/VDM/research/gsn/JavaPaper>
- [Grewal et al., 1997] M. S. Grewal, A. P. Andrew, *Kalman Filtering Theory and Practice*, Prentice Hall, 5th Printing, June 1997. Original publication February 1993
- [Grossberg S., 1982] Grossberg, *Studies of mind and brain: Neural principles of learning, perception, development cognition and motor control*, Reidel, Boston, 1982
-

-
- [Grossberg S., 1988] S. Grossberg, *The ART of adaptive pattern recognition by self-organizing neural network*, Computer, Vol. 21, Mar., 1988, page 77-88
- [Hall et al., 1990] D. L. Hall, R. J. Linn, *A Taxonomy of Multi-Sensor Data Fusion Techniques*, Proceedings of the 1990 Joint Service Data Fusion Symposium, Vol. 1, May 1990, page 593-610
- [Hartmann G. et al, 1995] Georg Hartmann, Rüdiger Wehner, *The ant's path integration system: a neural architecture*, Biological Cybernetics 73, Springer Verlag, 1995, 15 pages (483-497)
- [Harvey D., 1967] D. Harvey, *Models of the evolution of spatial patterns in human geography*, Models in Geography, London, 1967, chapter 14
- [Hertz J. et al, 1991] John Hertz, Anders Krogh, Richard G. Palmer, *Introduction to the theory of neural computation*, Addison-Wesley, Redwood City CA, 325 pages, 1991, ISBN 0-201-50395-6
- [Kampmann P., 1991] Peter Kampmann, *Ein topologisch strukturiertes Weltmodell als Kern eines Verfahrens zur Loesung von Navigationsaufgaben bei mobilen Robotern*, Diss. Techn. Univ. Muenchen, 1991, 109 pages, ISBN 3-18-147808-3
- [Kasuba, 1993] Tom Kasuba, *Simplified Fuzzy ARTMAP*, AI Expert, November 1993, page 18-25
- [Knieriemen T., 1991] Thomas Knieriemen, *Autonome mobile Roboter; Sensordateninterpretation und Weltmodellierung zur Navigation in unbekannter Umgebung*, 1991, 258 pages, ISBN 3-411-15031-9
- [Kohonen T., 1981] T. Kohonen, *Automatic formation of topological maps of patterns in a self organizing system*, Proc. 2nd Scandinavian Conf. on Image Analysis, Espoo, Finland, 1981, page 214-220
- [Kohonen T., 1990] T. Kohonen, *The self-organizing map*, Proc. of the IEEE, Vol. 78, No. 9, Sep. 1990, page 1464-1481
- [Kuipers B., 1991] Benjamin J. Kuipers, Yung-Tai Byun, *A robot exploration and mapping strategy based on a semantic hierarchy of spatial representations*, Journal of Robotics and Autonomous Systems, 8:47-63, 1991
- [Kuipers B., 1997] Benjamin J. Kuipers, David Pierce, *Map Learning with Uninterpreted Sensors and Effectors*, University of Texas at Austin, Austin TX 78712 USA, appeared in Artificial Intelligence Journal, 1997, 60 pages
- [Kurz A., 1994] Andreas Kurz, *Lernende Steuerung eines autonomen mobilen Roboters*, Diss. Techn. Hochschule Darmstadt, 1994, 192 pages, ISBN 3-18-342808-3
- [Levenshtein V.I., 1975] V.I. Levenshtein, *On the minimal redundancy of binary error - correcting codes*, Information and Control, Vol 28, Nr. 4, August 1975, 23 pages (268-291), <http://theory.lcs.mit.edu/~iandc/Authors/levenshteinvladimiri.html>
- [Lin et al., 1993] Long-Ji Lin and Stephen Jose Hanson, *On-line learning for indoor navigation: Preliminary results with RatBot*, NIPS93, Robot Learning Workshop, 1993
- [Lin L.J., 1993] Long-Ji Lin, *Reinforcement Learning for Robots Using Neural Networks*, PhD thesis, Carnegie Mellon University, Pittsburgh, PA, 1993
- [Mächler Ph., 1995] Philip Maechler, *Map creator for variable natural landmarks*, report: Federal Institute of Technology, Switzerland, 1995, 14 pages
-

-
- [Maechler Ph., 1996] Philip Maechler, *Navigation and path planning, based on automatic recognition of significant experiences from the environment*, Thesis Intermediary report, June 1996
- [Maechler Ph., 1997] Philip Mächler, *Robot odometry correction using grid lines on the floor*, M.C.P.A Pisa, Italy, 1997, 10 pages
- [Mallot H.A. et al, 1995] H. A. Mallot et al., *View-based cognitive map learning by an autonomous robot*, ICANN '95, Max-Planck-Institut für biologische Kybernetik, Tübingen
- [Mataric M., 1990] Maja J. Mataric, *Navigating with a Rat Brain: A Neurobiologically-Inspired Model for Robot Spatial Representation*, J-A. Mayer, & S.W. Wilson (eds), 1990, From animals to animats, MIT
- [Michel O., 1996] Olivier Michel, *Khepera simulator package version 2.0*, Freeware mobile robot simulator downloadable from <http://diwww.epfl.ch/lami/team/michel/khep-sim/index.html>
- [Nake F., 1974] F. Nake, *Ästhetik als Informationsverarbeitung*, Springer, 1974
- [Newman, 1982] E. A. Newman, P. H. Hartline, *The Infrared "Vision" of Snakes*, Sci. Amer. 246(3), 1982, page 116-127
- [Nicoud et al., 1995a] J.D. Nicoud, Ph. Mächler, *Robots for Anti-Personnel Mine Search*, IAV'95 Conference, Espoo, June 1995
- [Nicoud et al., 1995b] J.D. Nicoud, Ph. Mächler, *Pemex-B, a Low Cost Robot for Searching Anti-Personnel Mines*, WAPM'95, Lausanne, June 1995
- [Pandya A.S. et al, 1996] Abhijit S. Pandya, Robert B. Macy, *Pattern Recognition with Neural Networks in C++*, CRC Press, Florida, 1996, ISBN 0-8493-9462-7
- [Pau L. F., 90] L.F. Pau, *Mapping and spatial modeling for navigation*, 1990, 355 pages, ISBN 3-540-52771-7
- [Pfeifer R., 1996] Rolph Pfeifer, *Building "Fungus Eaters": Design Principles of Autonomous Agents*, SAB96, From Animals to Animats 4, 1996, page 3-12, pfeifer@ifi.unizh.ch
- [Piaget J., 1970] Jean Piaget, *L'épistémologie génétique*, Que sais-je, Presses Universitaires de France, 126 pages, 1970, ISBN 2-13-040208-9
- [Piasecki M., 1995] Marek Piasecki, *Global Localization for Mobile Robots by Multiple Hypothesis Tracking*, Robotics and Autonomous Systems Nr. 16, 1995, 12 pages (93-104)
- [Pruski A., 96] Alan Pruski, *Robotique mobile, la planification de trajectoire*, 1996, 236 pages, ISBN 2-86601-549-5
- [Recce & Harris, 1998] Michael Recce, Kenneth D. Harris, *Memory for places: A navigation model in support of Marr's theory of hippocampal function*, Dep. of Anatomy and Developmental Biology, University College, London, UK, 40 pages.
- [Reuhkala E., 1983] E. Reuhkala, *Recognition of strings of discrete symbols with special application to isolated word recognition*, Acta Polyt. Scand. Ma 38. Dr. Tech. dissertation, Helsinki Univ. of Tech. 1983
- [Revuz D., 1984] D. Revuz, *Markov chains*, North-Holland, 1984, 374 pages, ISBN 0-444-86400-8
- [Ross S. M., 1997] Sheldon M. Ross, *Introduction to probability models*, Academic Press, ISBN 0-12-598470-7, 1997, 669 pages
-

-
- [de Sa, Ballard, 1998] Virginia R. de Sa, Dana H. Ballard, *Category Learning Through Multimodality Sensing*, MIT, Neural Computing 10, 1998, pages 1097-1117
- [Scheier Ch., 1996] Christian Scheier, *Categorization in a real-world agent using haptic exploration and active perception*, SAB96, From Animals to Animats 4, 1996, page 65-74, scheier@ifi.unizh.ch
- [Schmidhuber J., 1997] Jürgen Schmidhuber, *What's interesting?*, Technical report, IDSIA-35-97, Lugano, Switzerland, 23 pages, 1997, <http://www.idsia.ch/~juergen>
- [Schürmann J., 1996] Jürgen Schürmann, *Pattern Classification*, John Wiley & Sons, Inc, Canada, 1996, 370 pages, ISBN 0-471-13534-8
- [Song H., 1996] Ho-Keun Song, Jong-soo Choi, *Edge detection method for range image using pseudo reflectance images*, Dep. of Electronic Engineering, Chung-Ang University, Seoul, Korea
- [Wehner R., 1992] Rüdiger Wehner, *Homing in Arthropods*, Chapter 3 of *Animal Homing*, ed. by F. Papi, London, Chapman and Hall, 1992, pages 45-144
- [Welch S., 92] Sharon S. Welch, *Sensors and sensor systems for guidance and navigation II*, 1992, 323 pages, ISBN 0-8194-0859-X
- [Wolfram, 1992] S. Wolfram, *Mathematica Reference Guide*, Addison-Wesley, 1993
- [Xu H. et al., 95] H. Xu et al., *Sensor fusion and positioning of the mobile robot LiAS*, IAS'95, 1995

C
H
A
P
T
R
E

7 Remerciements aux dieux de l'Olympe



Merci à Jean-Daniel, Zeus souverain veillant sur nous tous, immortels et mortels, dieux et sémi-dieux, héros et farfelus assistants que nous sommes, la foudre de tes expériences incroyables et le pouvoir de ton savoir sans frontières, nous ont tous fait vivre au rythme de tes passions scientifiques. Merci de m'avoir offert une place sur le char du LAMI.

Merci à Laurent, véritable Poséidon, le dieu de la mer. Tu m'as encouragé à voyager au-delà des évidences, le long des chemins de la pensée abstraite au cours de débats surprenants. Ton humour m'a emporté dans les profondeurs de la langue française et tu m'as permis de hisser les voiles de l'amitié et du partage. Merci pour ton soutien.

Merci à Jelena, personnification d'Athéna, déesse de la sagesse et de l'intelligence. Mariant force et indulgence, ta présence s'accompagne toujours de conseils judicieux. Tu prends pitié des faibles et tu aides les héros (je fais partie de ces derniers, n'est-ce pas?). Tu as su d'une main de fer, me redresser en tout temps et tes jugements sages m'ont toujours remis sur la bonne voie. Merci pour ta lance et ton bouclier.

Merci à Yuri, incarnation de Dionysos, dieu du vin et de l'allégresse! Tu as égayé les quelques heures de liberté de nos vies de pauvres thésards trop responsables par des orgies culturelles, des expéditions alpines et de folles aventures. Tu m'as permis de découvrir la Suisse Romande avec un regard Hispanique. Merci pour les rires et ta bonne humeur.

Merci à Paolo, qui tel Ulysse est doté de talents incroyables. Grâce à ta patience et à ton courage, tu as franchi le gouffre séparant un Italien d'un Suisse Allemand. Tu as su, avec beaucoup de dextérité, me secouer les méninges et me souffler quelques unes de tes brillantes idées (on parle de pasta bien sûr!). Bien qu'étant à l'étranger, merci d'avoir toujours gardé un oeil de cyclope sur mon travail.

Merci à Marie-Jo, notre Iris, messagère des dieux, personnification de l'arc-en-ciel, union entre la Terre et l'Océan. Toujours présente, toujours prête à secourir, toujours un mot d'encouragement. Tu es le pilier du LAMI, l'arc sans lequel il ne peut voler. Au service de Zeus, tu sais pourtant créer le lien entre nous tous.

Merci à Jean-Bernard, qui tel Prométhée, ami de l'homme et pour moi, collègue de bureau inoubliable. Tu es doté de la logique, et tu as tenté, parfois avec grande peine, de me l'inculquer. Tu nous as donné à tous le feu de ta joie, les flammes de tes projets scientifiques mais aussi ton savoir-vivre et ton expérience de la vie. Chaudement Merci.

Merci à Olivier C., un Apollon sincère, dieu de la lumière, de la musique et de la mantique. Un dieu magnifique aux talents multiples dont celui de l'art pastoral. Toujours à l'écoute, ouvert et prêt à dépanner tes amis. Tu n'as jamais hésité à donner un coup de main en cas de besoin. Tes conseils et mille formules se jouent comme les notes émises de ta lyre de l'amitié et de la disponibilité.

Merci à Dario, un Hélios, dieu de la lumière. La nuit, tu traverses la voûte céleste et le jour, tu parcoures la surface de la terre. Tu es à la recherche de nouvelles cultures et de nouveaux spécimens robotiques. Tu m'as encouragé à m'étendre au delà des frontières Helvétiques et tu m'as poussé à regarder dans l'au-delà des univers de réseaux neuronaux. Merci pour cet initiation au voyage.

Merci à Alcherio, Esculape modernisé, dieu de la médecine, aux talents rares, tu ressuscites les assistants morts d'overdose de folie scientifique. Ton calme, ton regard scrutateur, tes conseils, tes idées méticuleuses et ton don de l'organisation nous remettent tous sur le chemin de la guérison. Merci pour toute ton aide et pour toute tes critiques constructives.

Merci à Sonya, telle Perséphone, ta présence fait reflourir le monde et apporte le bonheur à ceux qui t'entourent. Mais, enlevée par la sombre et lointaine puissance d'Hadès-Mikado, tu nous replonges chaque année dans l'affliction, nous abandonnant au seul espoir de ton retour, synonyme de floraison et de moissons. Le soutien décidé que tu m'as accordé dans les moments difficiles et l'attachement qui nous lie ont été comme l'eau et la terre qui ont fait éclore mon coeur et ma thèse; ils me donneront demain la force d'affronter sans crainte les nouveaux assauts d'Hadès.

Finalement je remercie l'ensemble du jury, Héros sauvant l'humanité par leur analyse sévère et leur combat pour la victoire de la juste cause, par leur divers odyssées sur la terre, au ciel ou dans les enfers de nos laboratoires.... Entreprenant des missions dangereuses à des fins morales, pour la conquête d'un royaume de formules et dévoués à la quête du bien rationnel. L'Humanité se débat avec les monstres de la technologie bidon, la maladie de l'imperfection et les créatures de foire... Votre esprit d'invention, vos éternels combats et exploits pour l'accomplissement méthodique d'une science sans limites nous incitent à vous considérer comme mes mentors. Hercule, Thésée, Orphée, Achille, Dédale ou Icare, qui que vous soyez, une grand Merci à tous.

8 Publications

- Ph. Mächler, Looking for concepts: Unsupervised map construction with unknown sensor configuration, IROS'98, Victoria, Canada, October 1998
The navigation method presented here allows a robot to find its position and its route without prior information about its environment. What is going to be used as landmark is initially unknown, and no specific preprocessing of the sensor signals is done. The robot nevertheless extracts meaningful information from its environment and uses it to build a map. Both landmark definitions and map are continuously adapted. Working without pre-defined categorization takes full advantage of the sensor abilities.
The result shows that a robot can reliably recognize self-defined landmarks suitable for its sensor capability and create a map of its environment.
- Ph. Mächler, *Without a clue: Unsupervised robot navigation with unknown sensor configuration*, IAV'98, Spain, March 1998
Abstract: The aim of this research project is to perform robot localization and navigation without prior information about the environment. A method for the interpretation of different kinds of unknown but significant sensor signals as "landmarks" is proposed. These landmarks are used to create a topological map containing navigation information to get from one landmark to another. A navigator steers the robot toward "learned" landmarks to synchronize the actual robot position with the position previously associated to the landmark. The results show that a robot can learn to stabilize its odometry error without any prior information about the environment or its sensor ability. It extracts only adequate environment features which make the algorithm suitable to new and unknown surroundings.
- Ph. Mächler, *Robot odometry correction using grid lines on the floor*, M.C.P.A. Pisa, Italy, February 1997
This paper presents an algorithm to correct the odometry error of an autonomous mobile robot only by using a painted grid on the floor. The estimated robot position is calculated by odometry and is matched with the grid lines, whose existence is known to the robot. A new "position probability function" was developed and used by the correction algorithm.
The correction of the odometry error is also based on interactive trajectory modification, in order to reduce the error when crossing a line.

Publications of other topics:

- Ph. Mächler, *Detection Technologies for Anti-Personnel Mines*, AVMC'95 Symposium, California, April 1995
- J.D. Nicoud, Ph. Mächler, *Robots for Anti-Personnel Mine Search*, IAV'95 Conference, Espoo, June 1995
- J.D. Nicoud, Ph. Mächler, *Pemex-B, a Low Cost Robot for Searching Anti-Personnel Mines*, WAPM'95, Lausanne, June 1995
- J.D. Nicoud, Ph. Mächler, *Demining Robots*, IAS-4 Conference, Karlsruhe, March 1995

C H A P T E R

9 Curriculum Vitae

Personal details

Name	Philip Mächler	
Date of Birth	March, 17 th 1967	
Marital Status	Bachelor	
Nationality	Swiss	
Contact address	rue Pichard 11 CH-1003 Lausanne http://diwww.epfl.ch/lami/team/maechler/	Priv.: +41 21 312 90 72 Prof.: +41 21 693 39 07 philip.maechler@epfl.ch

Educational background

1994 to 1998	Ph.D. candidate in computer science under Prof. Nicoud in the Microprocessor and Interface lab at the Federal Institute of Technology, Lausanne.
1987 to 1993	Dipl. El. Ing., Federal Institute of Technology, Zurich.
1983 to 1987	Gymnasium, Zurich, type C (mathematics, natural science).

Professional Experience

1995 to 1998	Working on a Ph.D. thesis on autonomous robot navigation.
1994 to 1995	Developing of a Robot for Anti-Personnel Mine searching, EPF Lausanne.
1992 to 1998	Founding a small company (PHI-Systems GmbH) for the development and production of a CD-Changer system, containing 777 CD's and 35 CD-readers.
1990 to 1991	Co-development of a board-computer for an electric mobile vehicle ETHOS, created by students at EPF Zurich.
1988 to 1989	Assistant teacher in the first official computer course in Zurich to introduce programming to school-teacher.

Linguistic skills

German	Mother tongue (Swiss-German as well)
French	Fluent
English	Fluent

Others

Sport	Mountain-biking, swimming, water-skiing.
Computer	Knowledge of several office applications and computer languages incl. HTML.