TECHNISCHE
UNIVERSITÄT
WIEN

VIENNA
UNIVERSITY OF
TECHNOLOGY

DIPLOMARBEIT

Geometric World Model Repository and Localization for Autonomous Mobile Robots

ausgeführt am Institut für
Computertechnik Inst.-Nr. E384
der Technische Universtiät Wien
unter der Anleitung von

UNIV.PROF. DIPL.-ING. DR.TECHN. HERMANN KAINDL
und DIPL.-ING. DR.TECHN. GREGOR NOVAK
als verantwortlich mitwirkendem Assistenten

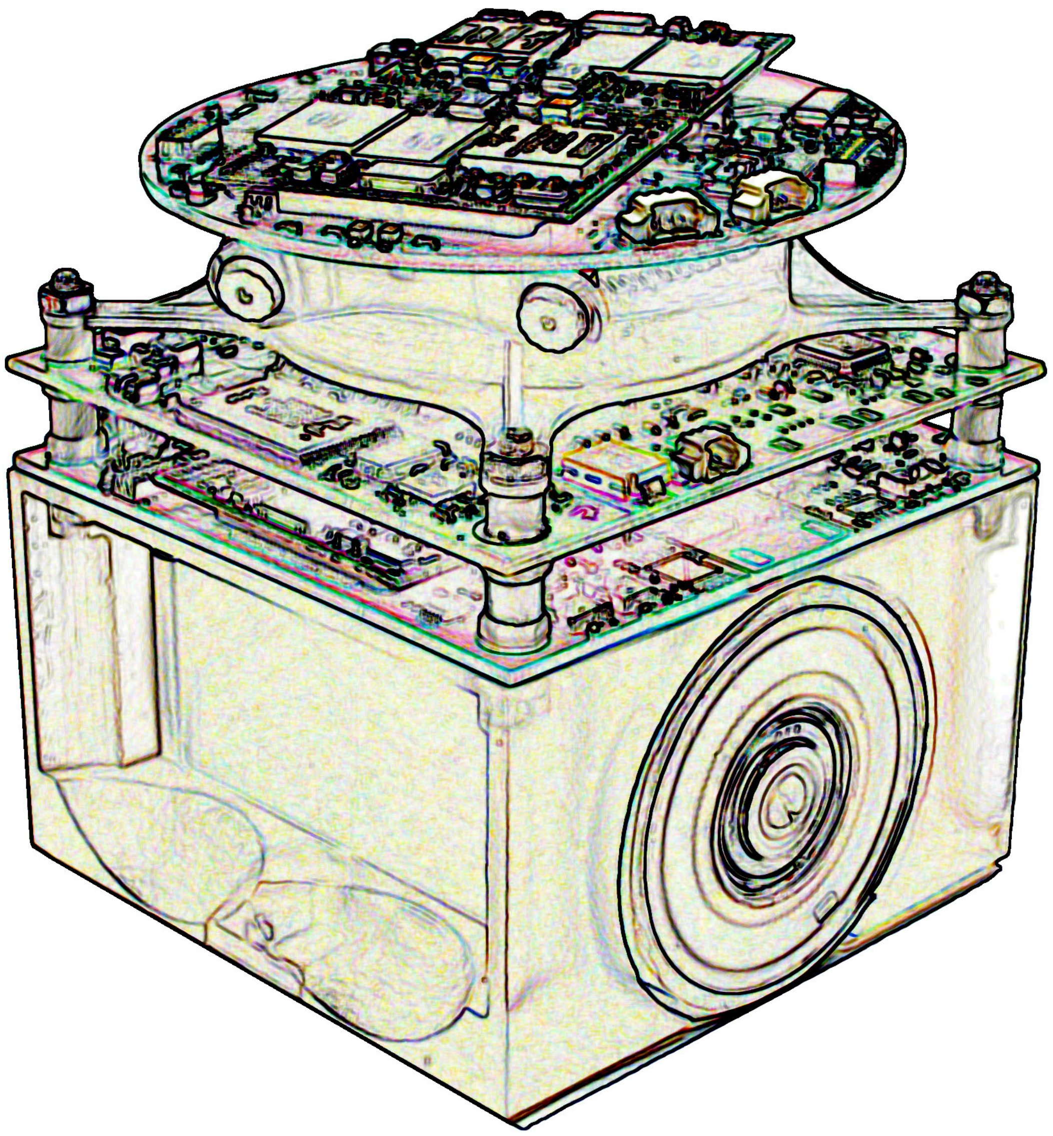
durch

TOBIAS DEUTSCH

Matr.-Nr. 9625115

Borschkegasse 1/9
1090 Wien

Wien, 11.Februar 2007



Zusammenfassung

Wesentliche Voraussetzung für einen autonom agierenden Roboter ist die Fähigkeit, Wissen über seine Position in der Welt zu sammeln. Dieses Wissen basiert auf Sensordaten und einem Weltmodell zusammen, welches wiederum aus einer Karte von statischen Elementen und Daten über mobile Objekte besteht. Die Sensordaten werden gemeinsam mit dem Weltmodell zur Bestimmung des wahrscheinlichsten Aufenthaltsorts des Roboters in der Welt verwendet.

Diese Diplomarbeit behandelt die Implementierung eines geometrischen Weltmodells und eines Lokalisierungssystems. Als Zielplattform dient Roboterfußball, gespielt von kleinen, autonomen Robotern, die eine Kamera, einen Prozessor und eine Antriebseinheit benötigen. Da es keine zentrale Kontrolle gibt, müssen sie in der Lage sein, ihre eigenen strategischen Entscheidungen zu treffen und ihre Position in der Welt selbst zu bestimmen.

Für das Weltmodell und die Lokalisierung wurde jeweils ein einfacher und ein komplexer Lösungsansatz entwickelt. Aufgrund der geringen Größe des Roboters (7,5×7,5×9 cm^3) sind die verfügbaren Ressourcen stark beschränkt. Abhängig von den vorhandenen Ressourcen und den Anforderungen an Strategie, Lokalisierung und Weltmodell kann durch die zwei Varianten ein geeignetes System realisiert werden.

Das geometrische Weltmodell umfasst eine Karte und einen Positionsverlauf für die beweglichen Objekte (Ball und Roboter). In der einfachen Variante besteht die Karte aus einer Menge von Liniensegmenten; die Objekte werden auf einen Punkt simplifiziert. Im Positionsverlauf werden nur die Positionshypothesen (gegenwärtige und vergangene) gespeichert. In der komplexen Version bestehen Karte und alle Objekte aus Polygonen. Dies hat den Vorteil, dass strategisch wichtige Fragen, wie zum Beispiel, ob sich der Roboter hinter dem Ball befindet, präziser beantwortet werden können. Zusätzlich wird neben der Speicherung der Positionshypothesen noch eine Positionsvorhersage angeboten.

Der einfache Lösungsansatz für das Lokalisierungssystem basiert auf einem linearen Näherungsverfahren und stellt eine Weiterentwicklung des bisher für Tinyphoon verwendeten Systems dar. Es können nur Sensordaten vom Odometer, vom Kompass und die von der Kamera erkannten Fixpunkte verwendet werden. Von diesen Fixpunkten muss sowohl die Richtung, als auch die Distanz erkannt werden. Im Gegensatz dazu kann der verwendete Partikelfilter beim komplexen Lösungsansatz sämtliche Sensordaten, die der Roboter aufnehmen kann, in die Hypothese integrieren.

Der in dieser Arbeit durchgeführte Vergleich der Lokalisierungsmethoden zeigt deutlich, dass der komplexe Ansatz bessere Ergebnisse erzielt als der einfache. Die Standardabweichung unterscheidet sich um den Faktor drei (Partikelfilter: 5 cm, lineares Näherungsverfahren: 15 cm). Im Falle der zwei Varianten für das Weltmodell kann ein Vergleich mangels objektiver Kriterien nicht formuliert werden. Die Entscheidung, welche Variante verwendet werden soll, ist eine Designentscheidung und hängt maßgeblich von den verfügbaren Ressourcen und der weiteren Verwendung der gespeicherten Daten ab.

Abstract

For autonomous behavior, a robot requires knowledge about its position in the world. This knowledge is gained through collecting relevant information on the environment with sensors and by comparison of this data with a model of the world. The model consists of a map of the static environment and information about moving objects. Based upon this model, the sensor data is used to generate a hypothesis of the position of the robot in the real world.

This thesis focuses on the implementation of a geometric world model and a localization system. A robot soccer game played by small, autonomous robots is the test-bed for this work. To be autonomous, each robot has to be equipped with a vision system, a processor, and a locomotion unit. Further, it has to be able to calculate its own strategic decisions and localize itself. The used robot system is Tinyphoon.

Due to the small size of the robots — 7.5 cm by 7.5 cm with a height of 9 cm — resources are limited. This problem is taken into account by providing a simple and a complex approach to the world model and the localization. Depending on available resources and requirements for the strategy, the localization, and the world model, an appropriate approach can be chosen.

The geometric world model repository (WMR) is a container storing a map and the position histories for moving objects (the ball and the robots). In the simple approach, the map is realized as a set of line-segments. Just the current and past estimated positions are stored for moving objects. In the complex approach, every object — including the playground — is stored as a polygon. The advantage to the simple approach is that each object has a shape. Thus, the question whether a robot is behind the ball can be answered more precisely. Another feature of the complex approach is the inclusion of a position predictor for each moving object.

The two approaches to the localization part are a simple position estimator based upon the linear least squares filter and a particle filter. The simple approach only uses odometric data, the compass, and landmark sightings with known distance and direction for the localization. It is an enhancement of the localization algorithm used for the robot Tinyphoon. The particle filter is able to integrate odometric data, the compass, the three distance sensors, and landmark sightings with or without a known distance.

The comparison between the two self-localization algorithms carried out in this thesis has shown that the complex self-localization algorithm produces better results than the simple approach. The measured standard deviations are: 15 cm for the simple algorithm and 5 cm for the complex algorithm. Thus, the particle filter enhances the overall performance.

Such a comparison cannot be defined for the WMR. The decision, which one to use is a design decision which depends on the available resources and the intended further use of the stored data.

Acknowledgments

For I but now declared that Odysseus should suffer many woes ere he reached his home, though I did not wholly rob him of his return when once thou hadst promised it and confirmed it with thy nod; yet in his sleep these men have borne him in a swift ship over the sea and set him down in Ithaca, and have given him gifts past telling, stores of bronze and gold and woven raiment, more than Odysseus would ever have won for himself from Troy, if he had returned unscathed with his due share of the spoil.

— *Ομηρος* (Homer), *Οδυσσεια* (Odyssey),
Book 13, Lines 131–138

Like Odysseus in Homers Odyssey, it took me a long time to finish my studies and to finalize my diploma thesis. I took numerous detours along the way and never took the short way home. But unlike Odysseus, I was never alone. My family and my friends were always around and supported me. The gifts I brought home are experiences, which cannot be expressed by grades. For the support and these gifts, I am grateful.

First, I want to express my special thanks to my parents who made my studies possible and supported me during my entire journey. Also, I want to thank Tatjana for her support and her patience — it has not always been easy with me during the last twelve months.

I thank my advisor Professor Hermann Kaindl for his support. Further, I want to thank Gregor Novak for infecting me with robot soccer.

Finally, I would like to thank the following people for their support (in alphabetical order) Abdul, Alex, Anna, Chris, Doc Joe, Markus, Matthias, Mr. — West-Coast-LaTeX-Customs — Biely, Roland, Rosi, Stefan, Thomas, Zwettler, and anyone else who I unfortunately forgot to include.

Tobias Deutsch, in February 2007

List of Figures

List of Tables

List of Algorithms

Contents

1 Introduction

Well begun is half done.

— *Αριστοτελης* (Aristoteles)

Robots will invade — like personal computers did twenty years ago — private homes in the near future. There are already many robots in industrial environments today. They are operating under well specified conditions and humans that work with robots must adapt to these conditions (i.e. staying clear of robots performing automated tasks). In private homes, this is undesirable. Here the robots have to cope with a dynamic — often chaotic — environment.

Other differences in the requirements between industrial environments and private homes are the ability to act autonomously and the user interface. For industrial applications, the installation and maintenance of a host computer controlling the robots is an acceptable cost factor. To be successful as a commercial product for private homes, the consumers should be able to install the robot without any previous knowledge and without the need for special installations in the building. Thus, a robot should be able to act autonomously as soon as it has been unpacked.

Professional programmers are needed to configure industrial robots for any future tasks. This is impossible for private homes. Here, the robots have to have an interface which is easy to use.

Autonomous behavior and an easy to use interface are advantageous in both worlds. In private homes a robot would not be accepted without them. For industry these improvements will result in a cost reduction. Autonomous behavior is the core requirement for autonomous cooperating robots. If several robots are assigned to the same task, they are then able to team up and to cooperate without further human interaction.

To develop autonomous cooperating robot teams, a competition was formulated — robot soccer tournaments. In this test-bed, the robots face similar problems like those in private homes. They have to localize themselves in an environment where objects are randomly repositioned. Without cooperation they cannot score and win the game. And they have to analyze the behavior of their opponents. These three abilities enable a team of robots to fulfill a complex task with minimal disturbances for the human inhabitants of the private home.

1.1 Problem Statement and Methodology

The main focus of this thesis is: "How can a robot localize itself in a known environment". To solve this problem, the robot first has to be able to store information about the environment. The combination of this knowledge with actual sensor readings results in an estimation of the robot's location at any given time.

To determine which localization algorithm to choose, a comparison between the most important algorithms is carried out. Based upon the results, the best means of locating itself — here called localization — is implemented. The resulting system is then compared with an enhanced version of the for the robot already existing localization approach.

The composition of the data storage depends on the intended further use. The minimum components required for the localization are: the shape of the playground and the position of the robot. It is not possible to formulate a comparison like the one for the localization algorithms. Thus, the methodology for the data storage consists only in the literature research and the formulation of requirements.

1.2 Outline

To provide background information, Chapter 2 ("Robot Soccer") gives an overview about the world of robot soccer. Based upon this chapter, the basic requirements for a robot capable of playing soccer are presented in Chapter 3 ("Autonomous Mobile Robots for AMiroSOT").

The next two chapters deal with the basics needed to solve the above stated problem. Chapter 4 ("World Model Repository") deals with storage of environmental information. The localization algorithms and their comparison are presented in Chapter 5 ("Localization").

Chapter 6 ("Tinyphoon") describes the target platform for this thesis. It is a robot fitting the requirements in Chapter 3.

In Chapter 7 ("Software System Description"), the a data storage and a localization system resulting from the previous chapters is presented. Further, the implementation is described. A comparison between the new localization approach and the enhanced version of the approach used by the Tinyphoon is conducted in Chapter 8 ("Implemented Self-Localization Systems in Comparison").

This work concludes with Chapter 9 ("Conclusion and Further Work"). It is a summarization of the results of this thesis and gives an outlook on further work.

2 Robot Soccer

Der Ball ist rund und das Spiel dauert 90 Minuten.

— Sepp Herberger

The ball is orange and the game lasts 10+80 minutes.

— Tobias Deutsch, after he had to referee for one and a half hours a RobotSOT match (which has a net play time of ten minutes).

To force progress it is wise to set up a competition — like the U.S.A. raced with the U.S.S.R. for the first man on the moon. In the field of artificial intelligence (AI) during the 80's and 90's of the last century, this competition was to build a chess computer able to beat the greatest human chess players (world champions). After Deep Blue[1] had beaten Garry Kasparov in 1997 this task shifted its aim from competitions with humans towards performance optimization. Deep Blue was a special high end computer consisting of more than 500 processors and was solely built for this task (and dismantled afterwards). 480 of these processors were special purpose chips designed for computer chess [Hsu06]. Nowadays, programs like Deep Fritz[2] using almost standard PCs[3] with much less calculation power can be victorious over human world champions.

With the chess competition won, AI needed a new — more complex — goal. A goal within a domain where progress can be — like in chess — easily defined. The idea for robots playing

[1] `http://www.research.ibm.com/deepblue`

[2] `http://www.chessbase.com`

[3] The last match between a human world champion — Wladimir Kramnik — and the computer chess program Deep Fritz was done by using a computer equipped with eight Intel Xeon processors each operating at a frequency of 900 MHz. It took place in December 2006 and was won by the computer (`http://www.rag.de/microsite_chess_com`).

soccer first appeared in 1992 in the paper “On Seeing Robots”[Mac93]. In the same year, in Tokyo a workshop on grand challenges in AI discussed independently about soccer as a possible domain. The game of soccer played by robots opened a large field of research topics like real-time sensor fusion, reactive behavior, decision making, learning, multi agent systems, and vision.

A competition with similar goals like robot soccer is the race of self guided cars through the Mojave Desert [Pat05]. This race is hosted by the U.S. government’s Department of Defense Advanced Research Project Agency (DARPA). The goal is to navigate through the desert along a 100-plus mile track along predefined track points without any interference by human operators or any other external computer. In the first year (2004), no car was able to complete the path, moreover, no one was able to go further than 7.36 miles. In 2005, all but one contestant were able to get far past that mark. From twenty-three teams, five completed the 132-mile track, and the best four were within the 10 hour limit. The winner — Stanford Racing Team — finished in 6 hours 53 minutes tightly followed by CMU’s Red Team (+11 minutes), and CMU’s Red Team Too (+21 minutes). Compared to 2004, this result is astonishing. The problem with this competition is money — the need for a real size vehicle and a lot of hardware makes a large budget unavoidable. Furthermore, a perfect test site like the Mojave Desert is not always around the corner.

Thus, robot soccer is not the only competition set up to enforce progress in robotics. But unlike other competitions like the DARPA challenge, robot soccer is resource friendly. Depending on the available skills, space, and money, a team can participate at different leagues. These are ranging from simulation leagues (with no robot-hardware involved), over leagues dedicated to one commercially available robot (e.g. Sony Aibo), to leagues where everything is built by the team (mechanics, electronics, and software).

Robot Soccer is a game similar to human soccer restricted in size and rules to the abilities of robots.

Next, the two world organizations[4] for robot soccer — RoboCup (Section 2.1) and FIRA (Section 2.2) — and their leagues are introduced. Finally, the new league AMiroSOT is described in Section 2.3.

2.1 RoboCup

> “RoboCup is an international research and education initiative. Its goal is to foster artificial intelligence and robotics research by providing a standard problem where a wide range of technologies can be examined and integrated.”[5]

[4]Since 1997, there are two robot soccer world organizations. Due to the fact that their leagues are relatively orthogonal, optimistic observers believe that a unification into one organization may be possible. (Compare `http://www.heise.de/tp/r4/artikel/19/19792/1.html`)

[5]`http://www.robocup.org/Intro.htm`

Founded in June 1993 first as the Robot J-League (Japanese Professional Soccer League), it transformed within a month to the international organization RoboCup[6]. Until 1996, mainly conferences and workshops were held. The Pre-RoboCup-96 in Osaka was a test-run for the First Robot World Cup Soccer Games in 1997 in Nagoya. While the test was held with 8 teams from Japan, the World Games had almost 100 participating teams and over 5,000 spectators.

Although robot soccer is a very broad and complex competition — the fun is an add on — they soon introduced a second type of competition — RoboCupRescue — many autonomous agents in a hostile unknown area carry out rescue tasks. Finally, to support young scientist and schools RoboCupJunior was added. A RoboCup@home league is planned, but the definition of the tasks is not finished. This league will focus on real-world applications and human-machine interaction with autonomous robots.

To enforce the promotion of science and technology, RoboCup formulated an ambitious goal:

The Landmark Project By mid-21st century, a team of fully autonomous humanoid robot soccer players shall win the soccer game, comply with the official rule of the FIFA, against the winner of the most recent World Cup.[7]

While 50 years (until 2050) seems to be a very short period, FIRA argues that from the first flight of the brothers Wright to the first industrial produced and soled yet engine it took also only 50 years (1903-1947), and by 1969 man flew to the moon. As another example they mention chess: from ENIAC to Deep Blue beating Kasparow it took exactly 50 years (1947-1997). Also the completion of the human genome sequence was done in less than 50 years (1953-2001).

In [BM03] several scientists — like Bernhard Nebel from the team CS Freiburg — stated, that they believed in the basic reachability of this goal. They believe that at least a demonstration game between humans and robots will be possible.

2.1.1 Leagues

A common rule for all different types of leagues and robots is that the robots/agents have to act autonomously.

The RoboCup organization holds tournaments in the following three major categories: soccer, rescue, and junior.

RoboCupSoccer The vanilla soccer competitions, as originally formulated.

Simulation League 2D — Simulation of independently acting agents on a virtual field. 5-minute halves.

[6]`http://www.robocup.net`

[7]`http://www.robocup.org/overview/22.html`

Simulation League 3D — Enhancement of the 2D simulation league. A complex physic model adds the third dimension and thereafter more complex game situations. 5-minute halves.

Small Size Robot League f-180 — Up to five small robots per team not larger than 18 cm in diameter using an orange golf ball play on an area slightly larger than a ping-pong table. 10-minute halves.

Middle Size Robot League f-2000 — Up to four middle-sized robots per team not larger than 50 cm in diameter using an orange soccer ball play on a field of a size of 12×8 meters^2. 10-minute halves.

Four-Legged Robot League — Is played by four dog-shaped robots (SONY's AIBO[8]) per team using an orange golf ball on a field the size of 5×3 meters^2. 10-minute halves.

Humanoid League — Biped autonomous humanoid robots. This league is still very young and thereafter design rules like size, sensor, etc. are in draft mode. Instead of soccer games, they compete by solving challenges (penalty kick, one vs. one, sprint, etc.).

RoboCupRescue The idea behind RoboCupRescue is to promote the development of robots/agents for search and rescue operations, human assistance, etc.

Rescue Simulation League — A generic urban disaster simulator is used to evaluate the team with the best rescue strategy.

Rescue Robot League — Duplicated sites of actual disasters are the playground in which robot teams compete for speed and reliability in search and rescue missions.

RoboCupJunior While the target of RoboCup and RoboCupRescue is scientific progress, RoboCupJunior is designed to be fun and educational. Young people and students are the target participants.

Soccer Challenge — A simplified game of soccer played 1 vs. 1 or 2 vs. 2.

Dance Challenge — The dancing judged by performance, creativity and choreography makes this league the most creative one.

Rescue Challenge — In an artificial disaster scenario victims have to be rescued.

2.2 FIRA

Like RoboCup, FIRA[9] (Federation of International Robot-soccer Association) also pushes scientific and technological progress via direct competition using the game of soccer. Among

[8] `http://www.sony.net/Products/aibo/`

[9] `http://www.fira.net`

the objectives of the association are support of young scientists, promote the development of autonomous multi-agent robotic systems, bringing scientists from different backgrounds like robotics, sensor fusion, intelligent control, communication, etc. together using the annual FIRA Robot World Cup and Congress.

FIRA was founded by Prof. Jong-Hwan Kim, KAIST, Daejeon, Korea, in 1995 and in the following year the first international championship was held there. While the FIRA Cup and the associated congress are the flagships of FIRA, they also organize — among other meetings — the International Robot Olympiad. The aim of this olympiad is similar to existing Mathematics-, Physics- and Chemistry Olympiads — offering young promising pupils a place to compete and to exchange (social networking).

2.2.1 Leagues

In contrast to RoboCup, the FIRA rules are much more adopted towards robot soccer. Especially the playgrounds are equipped with a border like in ice hockey and the corners are beveled.

HuroSOT — Humanoid Robot World Cup Soccer Tournament. Biped humanoid robot with a maximum size of 150 cm and a maximum weight of 30 kg. The playground is up to 3.5×4.3 meters^2 and an orange soccer ball is used. 5-minute halves.

KheperaSOT — One fully autonomous robot per team based upon Khepera robot equipped with an on board vision system. The playground is 1.3×0.9 meters^2, a yellow tennis ball is used. 5-minute halves.

MiroSOT — Micro Robot World Cup Soccer Tournament. Two teams, each equipped with 3 to 11 robots and a host computer connected to a vision system installed 2 meters above the playground (Figure 2.1). The robots are limited to a cube smaller than 7.5 cm×7.5 cm×7.5 cm. An orange golf ball is used. This league is divided into four leagues. They differ in the number of robots per team and size of the playground. The smallest league has 3 robots and a field size of 1.5×1.3 meters^2. These numbers increase up to 11 robots and a dimension of 4.0×2.8 meters^2. 5-minute halves.

NaroSOT — Five robots, one host computer, and a vision system installed above the playground per team. Each robot is limited to 4 cm×4 cm×5.5 cm. The playground is 1.3×0.9 meters^2. An orange ping-pong ball is used. 5-minute halves.

RoboSOT — Up to three robots per team, each robot equipped with its own vision and decision system. A team may use an external host computer to process the vision information of each robot. Each robot is limited to 20 cm×20 cm. No limit in height is given. The playground is 2.6×2.2 meters^2. A yellow tennis ball is used. 5-minute halves.

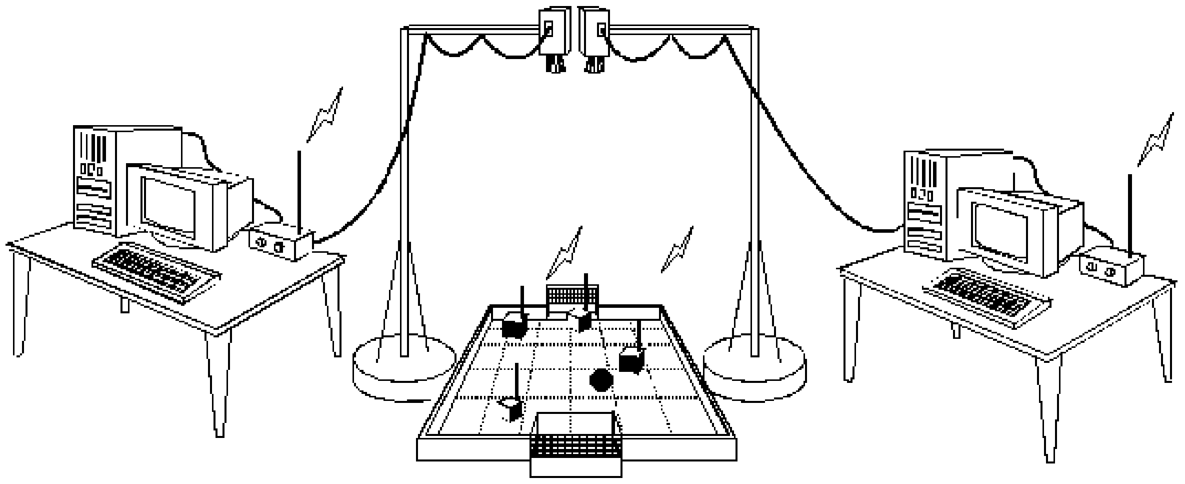


Figure 2.1. MiroSOT System Overview [Nov02]

SimuroSOT — SimuroSOT is like playing a virtual game of MiroSOT — hence, the rules are the same like for the middle and the large league of MiroSOT. 5-minute halves.

2.3 AMiroSOT

MiroSOT is a challenging league. The robots are one of the smallest in both organizations and move at a high velocity (up to 3 m/s). High electrotechnical, mechanical, and programming skills are necessary to build a competitive team. The smallest league needs about 2 m^2 of space and only three robots are needed for a start. Some teams sell their robots to finance the development of new ones. Thus, this league is a perfect place to start with robot soccer and grow with the skills gained. It is also an active community with more than 20 participating teams at the world cup 2006 in Dortmund[10].

What is missing in this league is the autonomous action. Most of the robots have a one-way communication with the host computer. They only receive movement commands which they execute. The simplest commands are the velocities for the left and the right wheel. More sophisticated approaches receive relative target coordinates, which are then transformed by the robot into wheel velocities. The external host computer has a complete world view through the atop mounted camera and is in control of all robots of the team. The only competitive approach in this setup is to calculate the strategy and the coordination between the robots at the host. Thus, introducing more autonomous actions to the robots is counterproductive.

Autonomous cooperative behavior between robots is useful in environments without globally available information and the lack of a central processing unit. If each robot perceives the world only partly through its local sensors, communication among the team mates is advantageous to generate a more complete world view. This is then done by collecting of

[10]`http://www.firaworldcup.de`

the incomplete world views of each team mate. These views are then combined into a single world view. The coordination among the team mates can be done by assigning roles for each robot in advance or by agreement and dynamic reallocation of the roles.

To establish such an environment, which enforces the usage of autonomous cooperative robots, some of the basic concepts of MiroSOT have to be altered. The global vision system has to be banned. The host computer is only allowed to send referee commands to the robots (e.g. start or stop the game). The severity of the changes leads towards a new league. This league should be designed in such way that existing MiroSOT teams can upgrade their robots to the new set of rules.

The — yet still to be officially founded — resulting league is named Autonomous Micro Robot World Cup Soccer Tournament (AMiroSOT) [KDB$^+$06]. The rules are basically the ones for MiroSOT[11] with the following major adaptions:

- No central vision system — compare with MiroSOT Law 4
- No communication with an external host computer — compare with MiroSOT Law 3
- Robots are allowed to be slightly higher (90 mm instead of 75 mm) — compare with MiroSOT Law 2.b
- The team-colors are attached to the sides and not on the upper side of the cube — compare with MiroSOT Law 2.b
- The robots have to move to the predefined position for penalty kick, free kick, free ball, and kick off autonomously — compare with MiroSOT Laws 10, 11, 12, 13

Additions to these rules are:

- The cameras have to be mounted parallel to the floor (within some tolerance). This rule avoids the usage of omni-vision cameras which provide a 360°round view of the field. A typical camera has a field of view of approximately 45°. Thus, the omni-vision provides up to eight times more information on the environment. Resulting, such a round view device reduces the necessity for environment information exchange.
- Restricted amount of battery capacity[12]. Analysis of the MiroSOT matches has shown that the robots move all the time over the whole field. They usually travel faster than the ball. Strategic positioning is — except for the goalkeeper — not necessary. With a reduced capacity of the battery, the robots are forced to use their resources economically. To enforce the development of such robots for a longer period, this reduction has to be done annually. Otherwise, economy would be in the focus of the participating universities only in the first few years.

[11]The official MiroSOT rules can be found at the FIRA home page at `http://fira.net/soccer/MiroSOT/overview.html`.

[12]The capacity of a battery is difficult to measure. Thus, the rule for a restricted amount of battery capacity is more or less a gentleman's agreement.

Similar to MiroSOT, the robot is allowed to overlap the ball up to 30%. The ball itself is an orange golf ball[13], [14] (see Figure 2.2) with a diameter of 42.7mm and 46g weight. The orange color simplifies the task of detecting the ball in the vision system. In the future, the color may change to less distinct colors like white.

Figure 2.2. Orange Golf Ball.

The playgrounds are the same as in MiroSOT. Figure 2.3 shows a field for the smallest league. The marked predefined positions FK (Free Kick), PK (Penalty Kick), and FB (Free Ball) are important for the robots to position themselves according to the actual referee command.

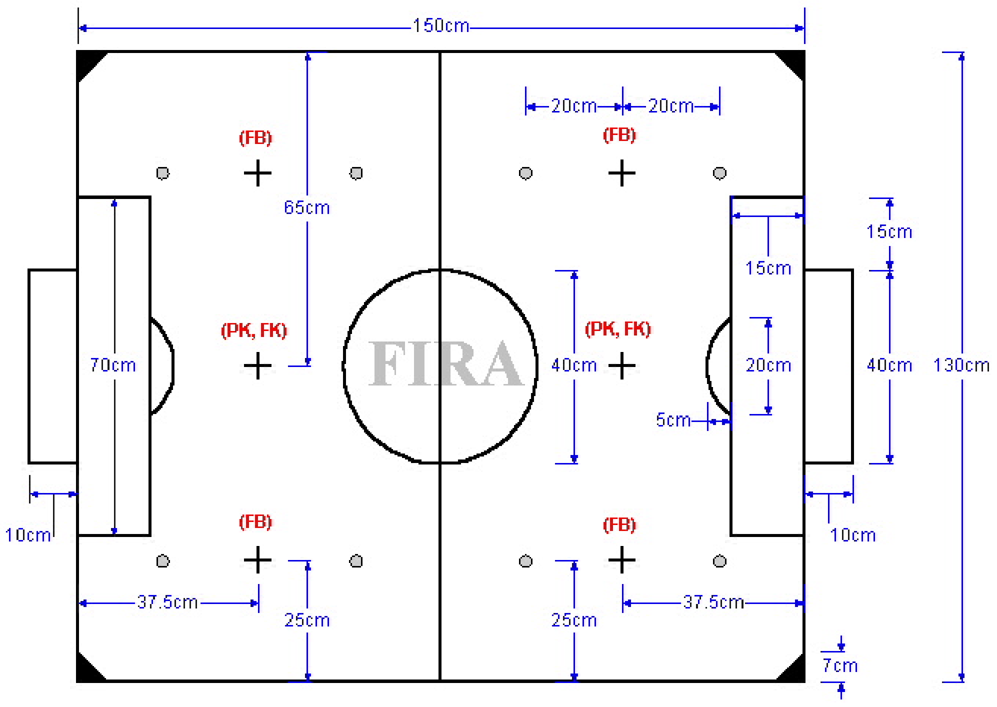


Figure 2.3. MiroSOT Small League Playground [FIR06]

[13]Golf balls are — according to the official rules (`http://www.usga.org/playing/clubs_and_balls/guide/book/appendix3ball.html`) — greater than 42.62mm in diameter with no upper bound specified. The United States Golf Association (USGA) provides a list of conforming golf balls (including diameter and weight) at `http://www.usga.org/equipment/conforming_golf_ball/conforming_golf_ball.asp`

[14]The number of dimples does not matter, but it should be mentioned that there is only one official golf ball with an odd number of dimples (333).

2.4 Summary

Robot soccer is a competition set up to enforce development in the fields of robotics and artificial intelligence. The basic idea reaches back to 1992, and the first tournament was held in Japan in 1993. Today, there are two major organizations in the world of robot soccer: RoboCup and FIRA. Both organize annually world championships in their leagues. These leagues range from tiny robots not larger than four by four centimeters up to large robots with a diameter of 40 cm and a hight of more than one meter. Biped humanoid robots are still in the beginning but they are another step towards the landmark project as denoted by RoboCup:

> By mid-21st century, a team of fully autonomous humanoid robot soccer players shall win the soccer game, comply with the official rule of the FIFA, against the winner of the most recent World Cup.

Another step leading towards this goal is to transform the successful MiroSOT league from semi-autonomous into a fully autonomous league called AMiroSOT. The small robots, with a dimension of 7.5 cm by 7.5 cm, are equipped with cameras and communication systems. They navigate through the playground based solely on their own sensor readings and the sensor information exchanged with their team mates.

.

3 Autonomous Mobile Robots for AMiroSOT

Law I A robot may not harm a human or, by inaction, allow a human being to come to harm.

Law II A robot must obey orders given it by human beings except where such orders would conflict with the first law.

Law III A robot must protect its own existence as long as such protection does not conflict with the first or second law.

— Those are the "Three Laws of Robotics" by Isaac Asimov

To be able to play AMiroSOT — which is the test environment for this thesis — autonomous mobile robots according to the rules are needed. A mobile robot is capable to move in a given environment. It has locomotion, optional sensors, and energy resources on board. It can communicate with a host to receive movement commands and send back sensor information. To be autonomous, this robot also has to have a processing unit on board, and it has to be equipped with sensors. This unit helps to transform the sensor data into a world view. The robot now can calculate strategic decisions on how to fulfill its tasks which are predefined or given by an operator online. The communication with the operator can be realized with radio transmissions, but also infrared and other vision-based systems like gestures. Based upon the strategic decisions, motion paths are planned. Through sensor feedback the execution of theses paths is monitored. For cooperative behavior with other robots, wireless communication is needed. Through this communication channel, the robots exchange sensor information and strategic decisions. This chapter gives a basic overview about the requirements and the components of an autonomous mobile robot for AMiroSOT.

"An autonomous agent is a system situated within and a part of an environment that senses that environment and acts on it, over time, in pursuit of its own agenda and so as to effect what it senses in the future." [FG97]

A robot is a real world agent. Thus, the term "agent" can be replaced by "robot" in the above given definition. Figure 3.1 translates this definition into a basic work flow. Through the sensors information about the world is perceived. This data has to be transformed into information processable by the further units. After the strategic decisions have been made, the actions which are necessary to fulfill these decisions are selected. Finally, the robot influences the environment by executing the actions using its actuators.

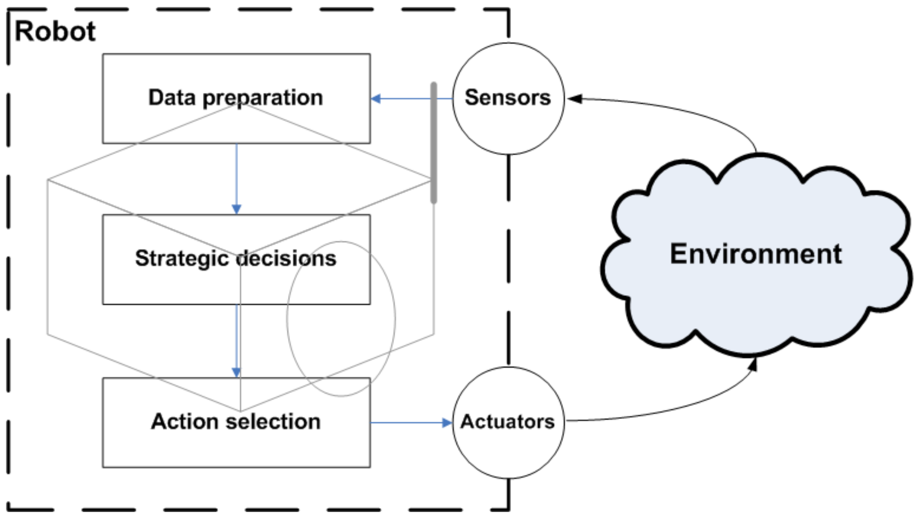


Figure 3.1. Basic Concept for an Autonomous Mobile Robot.

In AMiroSOT exists one further restriction to this definition of autonomy. During gameplay, the robot is only allowed to communicate with its team mates. The communication with the host computer is only allowed to transmit referee commands to the robots.

The chassis, power supply, actuators, and processors will be handled in the hardware section. Next, a short overview about sensors will be given. After some fundamental issues about communication are raised, different software architectures are introduced.

3.1 Hardware

The fast and sometimes rough play demands a strong and stable chassis. It is the platform where everything else is mounted. The occupied area should not exceed 7.5×7.5 cm^2. For higher stability in fast turns, heavy elements like motors and battery packages are mounted as low as possible.

At the front and at the back, the chassis has to have a special gear to be able to lead the ball. The small size allows no additional shooting device for additional ball handling.

Due to the high speed of the game, a two-wheeled locomotion driven by two motors is preferable. Three- and four-wheel systems would have too small wheel diameters. The resulting loss of momentum is a serious disadvantage in the competition. Within the small game fields, acceleration is as important as maximum speed.

The battery package has to be designed to be able to serve the motors, the sensors, and the processors for at least one half-time of the game. Furthermore, to fulfill the requirement to be mounted as low as possible, it has to be small. Ideally, it is positioned at a place where it is easily exchangeable.

Part of the hardware is also at least one processor with enough ports to read all sensor information and to control the wheel encoders. Preferably, instead of being directly connected to the processor a bus system is used to communicate with the sensors and actuators. For processing video data an additional processor is recommended.

Finally, a radio communication system has to be implemented. This enables communication among the robots. Furthermore, tasks can be sent to the robot by an operator.

3.2 Sensors

For autonomous navigation and strategy planning, the robot needs to perceive its environment through different sensors. Like humans, who have five senses, the robot then builds upon these various sensor readings its view about the world. The following list gives a short overview about possible sensors for AMiroSOT.

Odometry Odometry is position estimation during wheeled navigation; e.g. by using wheel rotation encoders as data source (see [Wik06c]). This position estimation system — or relative movement estimation system — is treated as a sensor.

Distance Typical types of distance sensors are infrared and supersonic. Their task is either to provide the distance to the next obstacle or a boolean signal if an obstacle is closer than a given threshold. To cope with faulty distance sensor readings, out-of-range errors, and unexpected objects, probabilistic algorithms to determine whether the value returned should be rejected or not have to be used (see [FT06] Chapter 6).

Vision The task of a vision system is to generate out of the perceived images abstract information about the environment — the so called landmarks, other robots, etc. Omni-vision camera systems[1] are not allowed in AMiroSOT.

Tactile These types of sensors are providing boolean information if a sensor has physical contact with an obstacle (e.g. bump sensor).

[1]Omni-vision camera systems are single cameras looking skywards with a mirror mounted at the top. Through this mirror, the camera produces spherical images. After post-processing, the outcome of this system is a 360°panoramic view.

3.3 Communication

As mentioned above, one purpose of communication is to cooperate with other robots or to communicate with the operator. The other purpose is to exchange data internally between different units — the sensors have to send data to the processor, the processor has to send motion commands to the wheel encoders. Thus, two different types of communication exist: internal and external.

Internal Internal communication is typically wired, reliable, and has a high transmission rate. The transmitted data is predictable (in the sense of what type of data is transmitted at what time). If there is only one processor and no sensor bus, internal communication is not needed.

External The radio controller mentioned in "Hardware" (Section 3.1) is used for external communication. Because this communication is done via radio, it is neither reliable nor fast (at least much slower than the internal communication). The type of data needed to coordinate strategic decisions with other robots is usually determined when the corresponding event occurs. The point of time when this event occurs and a message has to be sent may be unpredictable. Thus, external communication is asynchronous.

As a result, internal and external communication are recommending different types of communication. Due to the a priori predictability of the internally transmitted data, a data-oriented time triggered protocol is preferable. On the other hand, the event triggered communication pattern of the external communication suggests a message-oriented protocol.

Data-oriented In data-oriented communication predefined data packages are transmitted at predefined points in time. This transmission also takes place if the values of the data packages have not changed.

A priori to using data-oriented communication, the data which has to be transmitted (typically a reference to a variable) and the time slot in the communication cycle when it has to be transmitted have to be specified. The initialization establishes a stateful communication channel between all participants. Each participant is provided with the a priori known time schedule containing the information when which data will be updated. To keep these updates synchronous, precautions to guarantee a global time base have to be established.

Message-oriented A communication is initialized if a new message has to be sent. The system has to wait until the communication channel is free for transmission. If a collision occurs during the transmission, the process has to be restarted until the message has been successfully transmitted. This method of communication is asynchronous and stateless (at least at the data transmission level). Usually, no guarantees for the maximum duration until successful delivery of the message can be given.

The main differences of these two methods are the higher payload of the message-oriented protocol (messages are only transmitted if they contain new data) and the guaranteed transmission of data in the data-oriented protocol. Chapter 5 explains why data-oriented transmission is preferable for communication — localization can only be done with data from the same time base. This time base cannot be guaranteed with the message-oriented protocol.

3.4 Software Architectures

Software architectures for robot soccer range from simple monolithic systems to sophisticated layered architectures. The German national team (GermanTeam) participating in the RoboCup Sony Aibo league has such a layered approach [RBD+05]. The qualification matches prior to the world championships do not select the best team. Instead, they select the best approach for a module. After integration tests, they have the best solutions of all teams combined into one very competitive solution. GermanTeam won the world championships in 2004 and 2005.

The rules for AMiroSOT give no restrictions regarding the software architecture a robot can be equipped with. This section gives a general introduction into architectures for autonomous robots and a short overview about two selected architectures: the Three-Layer architecture and CLARAty.

3.4.1 Overview

According to [Gat97], the need for a layered software architecture for autonomous mobile robots was recognized in the early 1980's. Separation between robot control and logic resulted in better software designs and an easier decomposition of complex tasks. Additionally, it resulted in the ability to reuse the logic for several robots or one robot working as a platform for different tasks without the need of reprogramming all the low-level robot control units.

The reasons for introducing layered software architectures are ranging from decomposing a complex task to creating robot-independent reusable software. Each new research project introduces new (more or less different) architectures and it is very likely that an ideal general solution will never be found.

3.4.2 Three-Layer Architecture

Three-Layer architectures as proposed in [Gat97] are divided vertically into the control layer, the sequencer layer, and the deliberative layer (see Table 3.1). The control layer is the lowest layer and the deliberative layer the highest. The sequence layer is located between them.

The flow of information is strictly upwards and the flow of commands — the other way round — strictly downwards. Each layer is able to operate independently of the layers atop of it. Due to the flow of information, commands, and the nature of each layer, during one

Table 3.1. Three-Layers Architecture

Layer	Task	Constraints
Deliberative	Costly computations like long-term path planning and other strategic decisions.	All time-consuming algorithms should be placed here.
Sequence	Long term goals provided by the deliberative layer are decomposed into a set of primitive behaviors of the control layer.	Prediction should only be made in context of how to reach the long term goals.
Control	Reactive control algorithms or primitive behaviors like wall-following, moving to a target, avoiding collision, etc.	Internal states should be avoided whenever possible and — if unavoidable — expire after a short period.

run of the deliberative layer, several rounds of the sequence layer and many rounds of the control layer are processed.

> “By the standards of AI, the deliberative layer was trivial and uninteresting, which is precisely what makes the three-layer architecture non-trivial and very interesting.” [Gat97]

As shown in [Gat97], each layer should be simple and easy to handle. Complex behavior emerges from the combination of them.

A problem of this approach is that it is up to the system designer which functionality accounts to which layer. This results usually in the fact that one of the three layers is over-dominant. Further, several modules which are logically located at the control layer are duplicated in the deliberative layer due to algorithmic needs [VNE$^+$01].

3.4.3 CLARAty

Other than the Three-Layer architecture, CLARAty[2] [NWB$^+$03] is a two dimensional approach. One dimension — similarly to the Three-Layer architecture — is divided into a functional and a decision layer. The functional layer matches the control layer, and the decision layer is a combination of the sequence and the deliberation layer. The second dimension is the granularity of the software modules.

The granularity describes the partonomy (part-whole-relationship) of a robot. The lower the granularity, the more detailed access to the modules is possible (e.g. robot arm vs. joint motor). According to [VNE$^+$01], the concept of granularity is inspired by the object oriented programming concept. In Figure 3.2 an additional dimension called abstraction has been

[2]Coupled Layer Architecture for Robotic Autonomy — CLARAty

added to visualize the generalization structure of the elements of the functional layer. For example, the arm of the robot `Rover` is a specialization of the class `Appendage` which is itself specialized from the `Coordinated System` class. Atop of these classes, as the top generalization, resides the class `Robot`. At the granularity axis is the decomposition of the robot into its parts. In Three-Layer approaches, this granularity is distributed to the different layers, although the distributed elements originally belonged to the same layer.

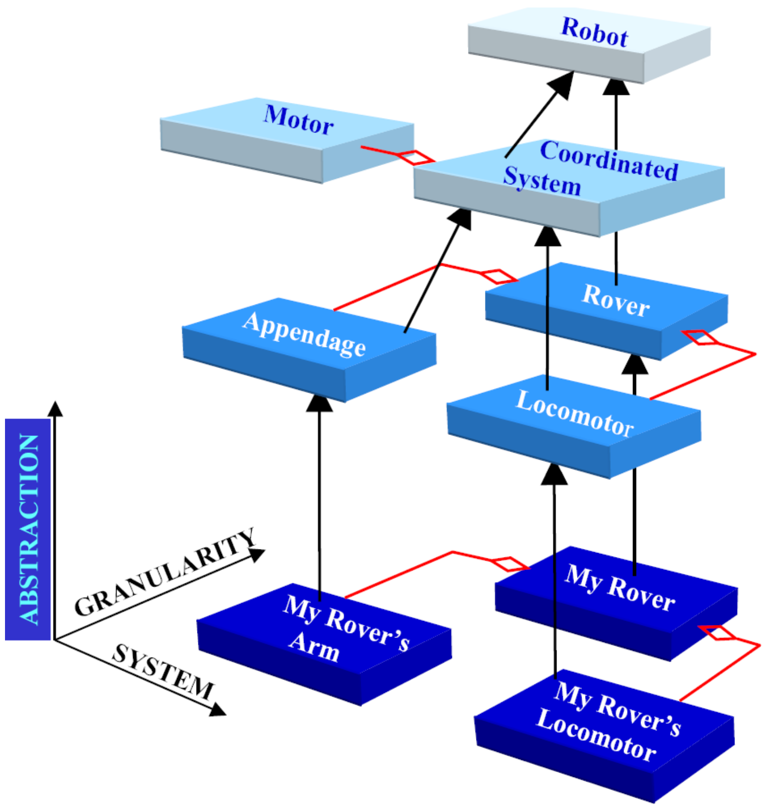


Figure 3.2. Granularity in CLARAty [VNE^{+}01]

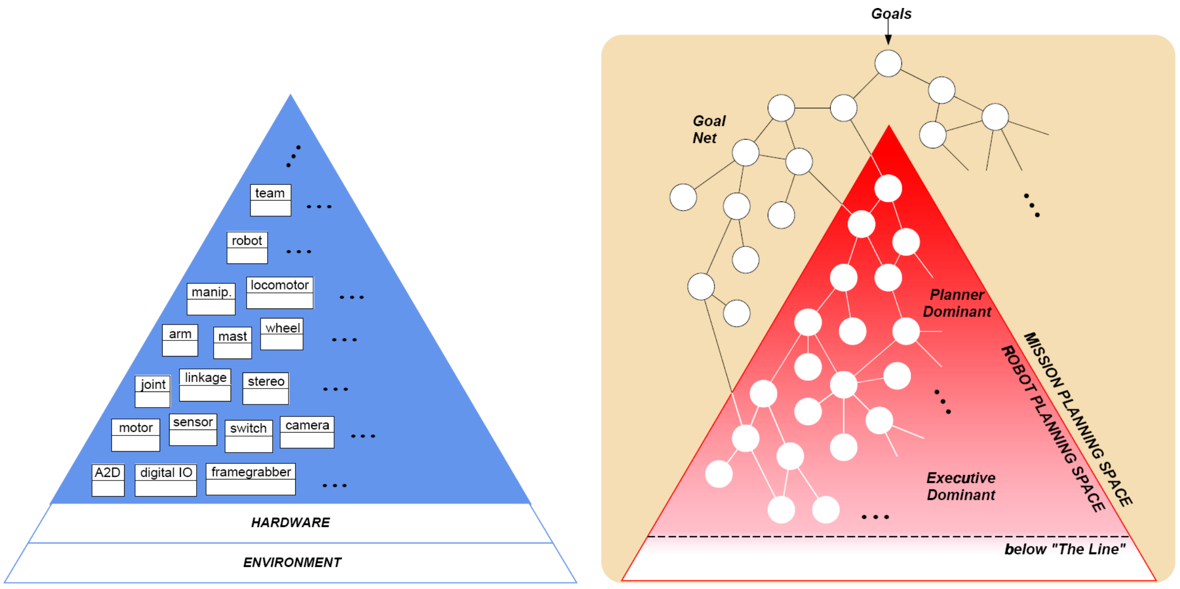
Next to these principal design issues, special care was taken during design to keep the CLARAty platform independent and reusable. Robot projects based upon CLARAty have to write their own hardware interfaces. For higher level functionality, existing code can be reused (without any adaption). This also simplifies the use of third party software modules.

Functional Layer The functional layer (Figure 3.3(a)) is a hardware abstraction and provides procedural robot control methods in different levels of granularity — e.g. move ahead, turn left/right, move to position, and move to position as fast as possible. This granularity is reached through an object oriented inspired approach. The highest level of granularity could be represented by the robot. The robot is then decomposed into smaller elements like locomotor and manipulator. These are themselves divided into their various parts. Finally, the direct motor control level is the one with the lowest granularity.

Other features are decoupling system limitations from algorithmic limitations, which enables the use of general algorithms and the separation between behavioral definitions and their implementation. The general algorithms can be replaced, if they are inefficient or obsolete, through newly added hardware. Each unit in the functional layer can also provide information about its energy consumption for certain actions. For simulation purposes, a unit gives feedback about the estimated outcome of the demanded action, without really executing it.

Decision Layer The decision layer (Figure 3.3(b)) plans, schedules and carries out the decisions resulting from planning. The decision layer is divided into two main areas: the goal oriented area and the planning oriented area. Goals are modeled by the goal net located in the mission planning space. This net realizes the break down of large goals into their intermediate goals. In CLARAty, a goal is a constraint like 'a joint angle should not exceed 30°' [VNE+01]. Inside the robot planning space are the task. A task is a set of parallel of sequential activities which specify what should be done (e.g. 'the joint should be at 20°').

In the decision layer, the granularity is the decomposition of tasks into subtasks. These subtasks may belong to several tasks.



(a) Functional Layer (b) Decision Layer

Figure 3.3. CLARAty Layers [VNE+01]

The interaction between the functional layer and the decision layer is designed as a client-server model with the latter acting as client [NWB+03]. The decision layer selects appropriate actions from the functional layer at different levels of granularity. Dependent on the algorithmic needs, a robot arm can be directed either by providing target coordinates or by controlling the joint motors directly. In Figure 3.4, a sequence of goals and tasks leads to a terminal state. This state represents a tasks which accesses the appropriate function in the lower level. This function is decomposed until the finest granularity is reached. Finally, the such found functions are accessing the hardware. In Figure 3.4 is the resulting set of goals, states, and functions marked green.

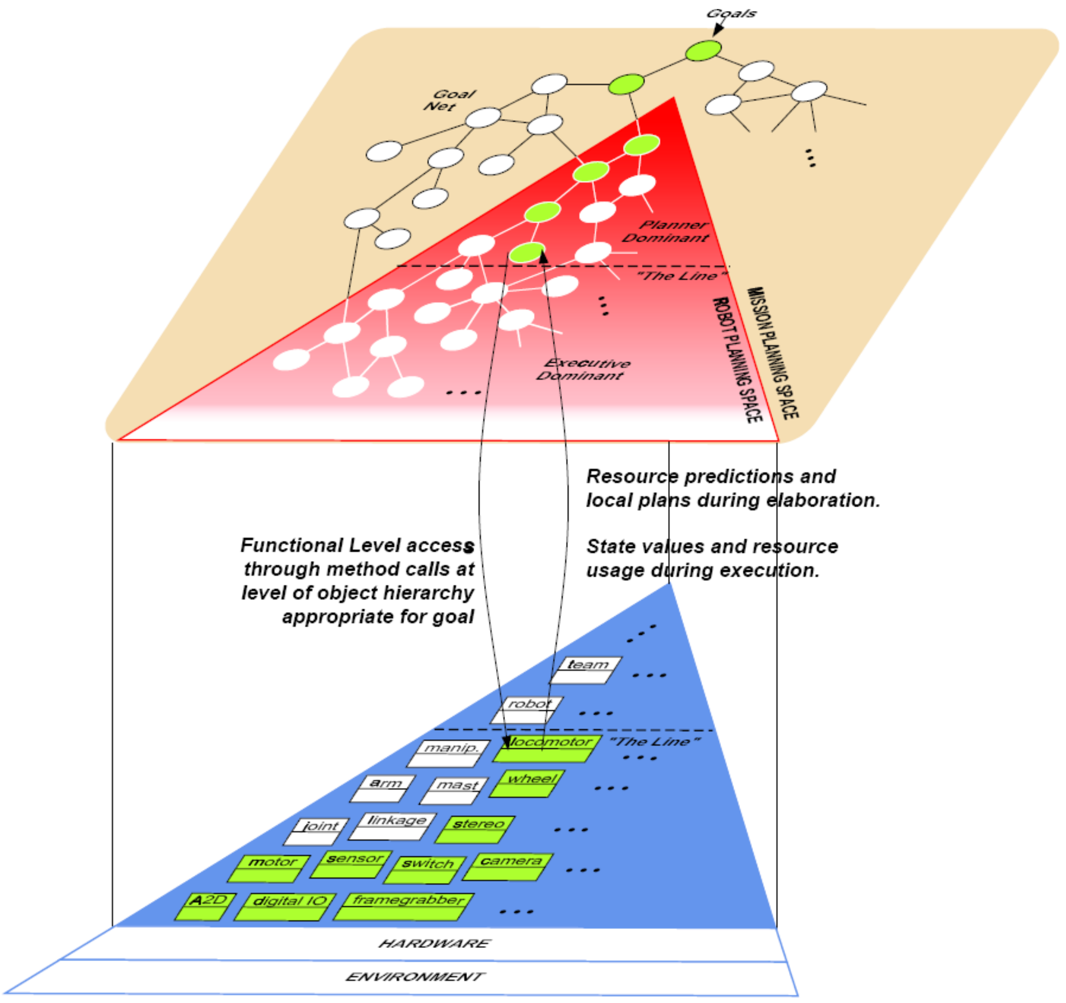


Figure 3.4. The Decision Layer Interacting with the Functional Layer at Different Levels of Granularity [VNE+01].

As an additional benefit, the CLARAty project has developed a simulator where the robot platform can be modeled in 3D. This helps — for example — in developing and optimizing of complex leg movements of hexapods (six legged robots), which are usually built after the model of insects.

CLARAty has been successfully implemented and tested by NASA. It has been used for the development of several rovers (Rocky 7, Rocky 8, K9, and FIDO) and for a Mars mission [NWB+03, SHP+03].

3.5 Summary

This chapter has shown some basic requirements and concepts for certain autonomous mobile robots. To fit the rough world of robot soccer and the strict rules of AMiroSOT, a robot has to be compact, robust, and fast. Further, to be independent, it has to be equipped with a processor, an energy source, and sensors. To be able to cooperate with its teammates, a radio communication module has to be provided.

4 World Model Repository

Man is the microcosm: I am my world.

— Ludwig Wittgenstein

The World is its own best model [Bro91][1]. Obviously, the world is best represented by itself containing every bit of information naturally — why do we need an abstracted representation? First, the world does not store its history — if we want to find out about e.g. the preferred attack situation of our opponents, we need a history of their former positions and actions. Second, sensor data is only limited liable — thus, for reducing the influences of measurement errors, we need algorithms like Kalman filter or particle filter (Chapter 5 Localization). Third, information which is exchanged among the team mates has to be processed and stored. And fourth, such an abstraction is needed to establish consistency of information for every module. As a result, a module is needed where information is processed and stored — the world model repository (WMR).

World Model Repository A collection of data, which is either (in)directly observable or generated by the robot or a-priori knowledge. The world model repository (WMR) represents the state of the world as perceived by the robot.

This chapter gives an overview about the data stored in a WMR. This data can be classified in two different ways: static vs. dynamic (Section 4.1), and categorized by content (Section 4.2).

[1]In the publication [Bro91], the sentence "The World is its own best model" never appears, although the meaning of it fits with the content of the publication. However, many publications like [Gat97] are referring to this publication using this sentence.

4.1 Static vs. Dynamic Data

Depending on the application, the developer determines which information is based on a-priori knowledge and, thus, static, and which information can change or be collected during runtime, using e.g. sensor data, and is, thus, dynamic. Dynamic data, which does not represent internal states (e.g. statistics calculated by the Decision Unit), has to be based on symbols which are generated from data from the sensors of the robot.

The design decision which data to model static and which dynamic has great impact on the performance and the possibilities of the robot, especially on the Decision Unit. Static data is usually more reliable but cannot be corrected if the deviation from the reality is too large. E.g. if the size of a playground is predefined with a width of one meter instead of three meters, the wrong value will still be trusted. Hence, the data of the stereo vision system will be rejected as out of bounds although it had been correct. Dynamic data can adapt to wrongly perceived/defined data, but the data is of lower quality, and additional calculation time is needed to process and store the perceived data.

Examples in the domain of robot soccer for static data are: maps; shape, color and number of the robots; shape and color of the ball; special positions on the playground like free kick point. The positions and the history of the positions of the robots and the ball are dynamic.

4.2 Categories of Data

Ideally, all data is stored in a generic way in the WMR. This includes environment maps, objects, positions, histories, rules, tasks, goals, semantic information, etc. The downside is an increase in complexity. Robot soccer data like rules, tasks, and strategic goals is better stored implicitly in the strategic instruction set of the decision unit.

The WMR as used for robot soccer contains the following categories of data:

Environment Map A map of the room/playground. Environment maps are discussed in Section 4.2.1.

Predefined Position The rules define several positions on the playground for special situations. E.g. the positions of the robots when a penalty kick is carried out.

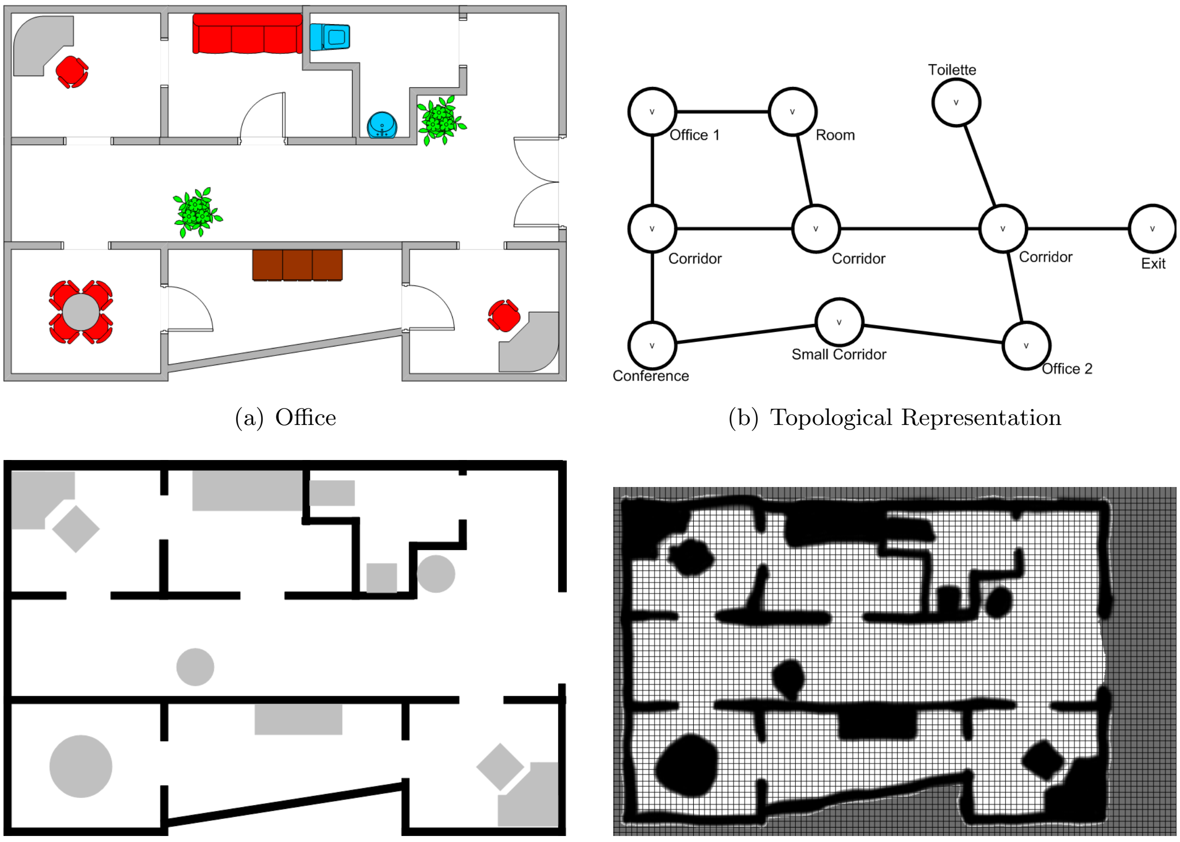
Robot Each robot is represented by its shape, direction, and position. The direction and the position are stored in a history to calculate speed, acceleration, and possible future positions. For team mates (includes the robot itself), additional information like their last action and their strategic role (e.g. striker, goalie) is stored.

Ball Shape, position and position history of the ball.

4.2.1 Environment Maps

Environment maps are used to store static information about the area like walls or doors. Sometimes, also moving objects are included in the representation (e.g. occupancy grids).

The environment — e.g. an office (Figure 4.1(a)) — can be represented using either topological maps (Figure 4.1(b)) or metrical maps which are divided into: (1) geometrical maps (Figure 4.1(c)), and (2) grid maps (Figure 4.1(d)).



(a) Office (b) Topological Representation

(c) Geometrical Representation (d) Grid Representation

Figure 4.1. An Office Map and Its Different Types of Representation.

Metric maps capture the geometric properties of the environment and store the location in a set of coordinates in cartesian space. Topological maps describe the connectivity of different places ([Thr02b] and [Bai02]). Thus, the properties of topological and metric maps are complementary.

Topological Maps

> “Topological representations aim at representing environments with graph-like structures, where nodes correspond to “something distinct” and edges represent an adjacency relationship between nodes.” [CLH+05]

In robotics, this broad definition is usually applied to generate *roadmaps* of the environment. As shown in Figure 4.1(b), the rooms are the nodes and the edges are passages between them. Enriched with additional information like distance between the nodes, node type, and direction to the node, these roadmaps can be used for navigation and inter-room path planning. For intra-room path planning, the nodes have to store additional information about the shape of the room (see Section 4.2.1).

While roadmaps are easy to use, they are difficult to generate automatically. The problem is to define what a room is (e.g. is this just a cupboard or a passage) and to detect cycles. If the robot should generate the map by itself, and the concept of rooms is not needed, the *generalized Voronoi graph (GVD)* is the better choice.

A Voronoi diagram is a surface divided into an arbitrary number of sections each containing a point [Wik06d]. The sections are generated in such a manner that each point of the border between two adjacent sections is at the same distance to the two points in the sections. The GVD [CWEAB00] is constructed similarly. Here, the points are not points, but the outer hull of obstacles like walls. The graph is drawn in such a manner that it is equidistant to its tangential borders. If the graph is divided (e.g. in case of a crossroad), a "virtual" border is risen between the two resulting graphs. The points of these graphs are again placed in equidistance between the real borders and the virtual border. A robot equipped with distance sensors to the front and to the side can easily build this graph automatically to map the environment. The resulting map (Figure 4.2) is ideal for path planning, due to the fact that the graph can directly be used as a path.

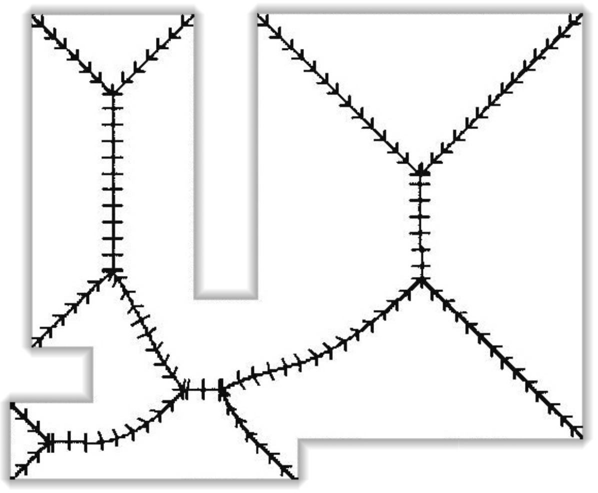


Figure 4.2. Generalized Voronoi Graph. [CWEAB00]

Geometric Maps

Geometric maps — also called feature-based maps or landmark maps — describe the environment using geometric primitives like points, lines, and circles or using polygons (see Figure 4.1(c)). To auto-generate geometric primitives out of sensor data is a difficult and time-consuming task. Thereafter, these maps are usually predefined by humans. Another approach to use geometric maps to represent dynamic/unknown environments is to connect sensor readings with a pre-defined set of object types (see Section 6.2.3). These object types can be represented in the map using primitives.

The advantage of geometric maps is that they are storage efficient. They can also be easily converted into grid maps which will be discussed in the next section.

Cell Decompostion

The environment is divided into simple cells which have a physical meaning (e.g. occupied). Other than topological maps which are abstract representations, cell decomposition represents occupied/unoccupied areas within the chosen granularity in a geometrically exact way.

The simplest approach to define the cells is to use adjacent squares. Other, more complex methods are trapezoidal decomposition where the area is divided into large trapezoids, and morse decomposition where the cells can be of arbitrary shape (e.g. circular slices). Also the areas defined by a GVD can be used. A more detailed explanation of these methods can be found in [CLH$^+$05].

Grid Maps The basic idea is a cell decomposition where the environment is divided into a grid of similar quadratic cells (see Figure 4.1(d)). Each cell is either marked as occupied or as free. An extension of the cell decomposition is the occupancy map [BFHS96]. Here the cells are not marked with the boolean values occupied and free. They are marked with the belief that it is occupied. The value ranges from 0 to 1, where 0 represents free and 1 represents occupied. This makes the grid map more robust against sensor failures and environment changes. Especially the problem of cell flickering[2] can be avoided.

The implementation of grid maps is simple — a two dimensional array representing the map storing at each array element the occupancy belief. The downside of this approach is that for large areas either the array gets too large or the resolution of the cells is too rough. Based upon the assumption that adjacent areas of the grid are marked with the same value, hierarchical approaches like quadtrees help to reduce size and search cycles.

Potential Field Map Similarly to grid maps, the area is divided into cells. But, other than in grid maps — where the cells are filled with absolute values for the occupancy belief — the potential field map stores a vector for each cell. This vector shows the gradient and the direction towards the highest value of the eight surrounding cells. (see Figure 4.3[3]). The result is a 3D map where the peaks represent obstacles and the valleys unoccupied areas. The downsides are the same as for grid maps, but the advantage is that robot

[2]In grid maps using boolean values for the cells, the belief whether a certain cell is occupied or not may vary between the two extremes free and occupied due to sensor errors, self positioning imprecisions, and different angles of view. The closer the belief is to the threshold above which the cell is marked as occupied, the more likely it is that the state of the cell will change between the opposite values rapidly back and forth — thus, making navigation more difficult.

[3]Here the direction of the vector is inverted — it points towards the "valleys".

navigation can be implemented very easily. The robot tries to reach its target without "climbing to the top of the peaks".

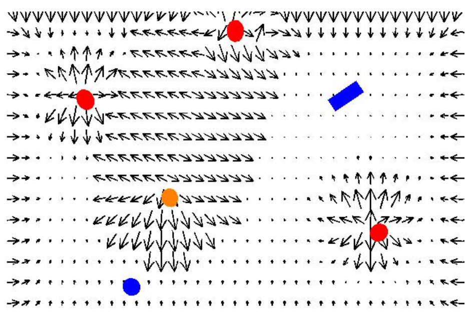


Figure 4.3. Potential Field. [RBD$^+$05]

The potential field in Figure 4.3 not only uses the vectors to avoid collisions but also to enforce strategically important areas. The robot is naturally attracted towards or distracted from these areas during path planning. To avoid collisions with moving objects, not only the area where the object is at the moment but also the predicted position in the future is raised — the gradient of the vector is increased. After a strategic evaluation of the situation, important areas are raised or leveled. The outcome is a reduction of possible target positions and pathes. This results in a speed up for the strategy unit.

Hybrid Approaches

Sometimes it is preferable to use different types of maps in one application. For the two different tasks of calculating strategic decisions and short term path planning, two different views of the world are advantageous. The strategic view should be reduced to a minimalistic abstract model which is best represented by a geometric map. For short term planning (how to reach the strategic position), detailed information about obstacles is needed. A potential field map is capable of providing detailed information enriched with a plausibility grade for this information.

In [YJ99], a combination between a grid-based map and a roadmap is proposed. Each room is represented by its own grid and possible connection points (doors, passages, etc.). The room grids are connected with transition graphs. The main advantages are the reduced size of the actually used grid map and the ease of inter-room path planning. Only when moving to another room, the next detailed map has to be loaded. The room transition graph itself is very simple to implement and can be extended with the distances from one door to another in one room to optimize inter-room path planning. When a robot maps the environment (Figure 4.4(a)), it collects the data in a large grid (Figure 4.4(b)) until enough data is collected to recognize and distinguish between the rooms. The rooms are then stored in smaller grids, and connections between the rooms are represented as graphs (Figure 4.4(c)).

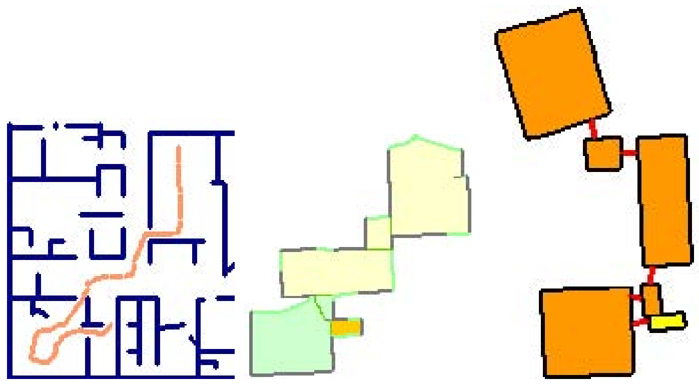


Figure 4.4. Hybrid Approach to Environment Maps. [YJ99]

4.3 Summary

The world model repository (WMR) is the module where all data representing the current state and the history of the world is stored. This data can either be dynamic or static. In the domain of autonomous robot soccer, the WMR contains an environment map, predefined positions, all robots (the robot itself, the team mates, and the opponents), and the ball. The map and the predefined positions are static. For dynamic data (robots and the ball), also a history is stored.

Environment maps can be categorized into three groups:

Topological Maps — represent connections between distinct points in the world (e.g. roadmap).

Geometric Maps — describe the world through an abstraction. This can be the position of a landmark or a square representing the space occupied by a table or a chair.

Grid Maps — decompose the world into cells. Each cell stores the belief that it is occupied by an obstacle.

Table 4.1 summarizes the main differences between the three basic types of environment maps. Geometric maps and topological maps with attached distances store their values in variables. Thereafter the precision of the predefined or measured values does not decline through storage. In grid maps, the precision for these values is lower bounded by the grain size of the grid. Along with the high precision comes the difficulty of building the map autonomously. To be able to mark an obstacle precisely in a geometric map, it has first to be fully measured, or — if dealing with landmarks — the type, the position, and the alignment of the object has to be detected. For topological maps, additionally the relations between the landmarks have to be determined. Due to the fact that grid maps only store the belief that a cell is occupied, only the cells corresponding to the obstacle have to be found and marked. The downside of this easiness of construction is that no information why this cell has been marked is stored. Thus, if faced with an update task, the map has to be rebuilt. In topological and geometrical maps updates are easier. Once the perceived object has been identified, it has to be associated with an already stored entity. Then the attributes of this entity are updated.

Table 4.1. Comparison of the Environment Maps

	Topological	**Geometric**	**Grid**
Precision	n/a[4]	+	−
Automatic Generation Difficulty	+	+	−
Automatic Update Difficulty	−	−	+

For robot soccer, geometrical maps are preferable. The world is a priori known. Robots and the ball do not change their size or shape during game play. Thus, prior to the game, the map can be generated, and during the game only position and alignment updates have to be performed.

[4]Topological maps store links between distinct objects. These links are of boolean type. Thus, precision of these connections cannot be expressed. Only the additional values like distance attached to the links may have a precision.

5 Localization

In theory, there is no difference between theory and practice.
In practice, there is.

— Attributed to Yogi Berra and Jan L.A. van de Snepscheut

It is the mark of an instructed mind to rest satisfied with the degree of precision which the nature of the subject admits and not to seek exactness when only an approximation of the truth is possible.

— *Αριστοτελης* (Aristoteles), Nicomachean Ethics

The last two sections were about the sensors an autonomous robot is equipped with and the types of maps that may be implemented in a world model repository. This section deals with the problem of relating sensor readings to a position — i.e. localization.

A definition of localization which can be found in [Fox98] reads as follows:

Given A model of the environment such as a grid-based geometric description of obstacles or a topological map of the environment.

Task Estimate the location of the robot within the environment based on observations. These observations typically consist of a mixture of odometric information about the robot's movements and information obtained from the robot's proximity sensors or cameras.

In this section, problems are presented which may occur while performing a localization task. After introducing a taxonomy for localization algorithms, four different types of them — linear least squares filter, Kalman filter, Markov localization, and particle filter — are discussed. They are ordered by their complexity (and by historic development). First the simplest filter, the linear least squares filter, is described. Afterwards the Kalman filter and its derivates are shown. Next, Markov localization is presented. Finally, the particle filter, which is based upon the principles of the Markov localization, is explained. It is noteworthy to mention that the Kalman filter can be used within a Markov localization or particle filter algorithm as an extension to get better results. After all algorithms are discussed, they are compared via a simulation example.

5.1 Problems

Localization could be as easy as this: collect all data, merge it, and figure out the perfect position. Unfortunately, one is confronted with problems. These problems can be divided into two groups. The first group consists of “fundamental problems”, which are due to restrictions in hardware or stem from inherent computer science (complexity) limitations. The second group can be roughly termed as “environmental conditions”, which are located in the “real world” the robots have to interact with. In the following the diverse problems are sketched.

Fundamental problems are independent of the concrete applications. They formulate the core problems of localization and are the same for all localization algorithms. The most important fundamental problems are:

Unreliable sensor data Data measured by sensors is not reliable (e.g. reflection of ultra sonic distance sensors).

Restricted field of view It is not possible to have a reliable complete round view with the sensors — similar to the vision of humans. Note that even sensors like omni-vision have uncovered areas.

Computation time Localization algorithms like Markov localization are computationally complex. Hence, selecting the right algorithm for a given application is crucial.

Time base for sensor readings If two sensor readings are made at different local times, they have to be integrated into the model sequentially by running the algorithm twice. Alternative approaches to this problem can be found in [KFM02].

Frequency of sensor updates On the one hand, some algorithms like Kalman filter rely on a fixed time interval between two sensor updates. On the other hand, for certain tasks, some sensors like vision need too much time to be updated in each cycle along with other sensors like odometry. Other algorithms — for example the particle filter — are able to deal with optional sensor data.

Multi-hypotheses It is advantageous if several hypotheses for one estimation problem can be followed until one hypothesis proved to be correct.

In contrast to the fundamental problems above, the environmental conditions are design issues. For every problem it has to be decided in advance, whether the robot should be able to deal with it or not. If a robot is always started at the same position, e.g. the so-called bootstrap problem (see below) needs not to be considered. Amongst others, environmental conditions are:

Kidnapped robot problem The robot is arbitrarily replaced. It has to find out that it has been replaced and thereafter performs a global relocation. [EM92]

Bootstrap problem Special case of the kidnapped robot problem where the robot is informed that it has been replaced (especially after initialization). [EM92]

Moving objects If other robots are not detected as such, they may be handled as a wall and thereafter produce a wrong position belief.

Changing environment Even with a priori known maps, the environment may change. Unmovables, like walls, may be razed or newly constructed resulting in a different map. Thus, map building is necessary for these environments. Map building and localization are strongly related tasks. Therefore, in the literature the combination of both is suggested — simultaneous localization and map building (SLAM) [FT06]. However, map building is outside the scope of this work.

Note that these lists are by no means complete. They reflect the main issues dealt with in literature.

5.2 Taxonomy of Position Estimation Approaches

To be able to compare localization algorithms, they have to be classified. At [Fox98], the following taxonomy of position estimation approaches is introduced:

Local vs. global localization The majority of localization algorithms start at a known position and add internal movement data (e.g. odometric sensor data) and external environment data (e.g. distance sensors) to this position each cycle. If the robot is replaced or the sensor data quality is too low, these algorithms are usually not able to recover to a useful position estimation (Kidnapped robot problem). Members of these so-called local approaches are the linear least squares estimator and the Kalman filter. Robots equipped with global localization algorithms like Markov localization and particle filter are able to localize themselves even under global uncertainty.

Static vs. dynamic environments In static environments, a once generated map can be used for every succeeding run without loss of certainty. While this is sufficient for indoor environments and robot soccer (changes are rare and the robot can be informed), outdoor navigation is much more complex. In these dynamic environments, parking cars could be mapped as unmovable obstacles in one run. The following day, all cars may be relocated to random positions.

Passive vs. active approaches If the robot's position estimation can only be done with the incoming sensor data stream, we talk about passive localization approaches. Active approaches enable the robot to influence this stream. Intentionally, moving towards areas where the next possible landmark is expected increases the quality of the estimation, regardless whether the landmark has been found or not. Active approaches are usually coupled with global localization algorithms.

This chapter focuses on local and global localization, static environments, and passive approaches. Active approaches have to be discussed along with the decision making. To be able to cope with dynamic environments, map (re-)building is necessary. Both topics are not within the scope of this work.

5.3 Linear Least Squares Filter

The linear least squares filter (LLSQ) is a simple state estimator. Based upon an observation history, outliers are muted. The dimensions of the state are not correlated — hence, the x and y values of a position are treated independently resulting in the fact that a faulty value in one dimension has no influence on the trustworthiness of its counterpart in the other dimension. The advantage of the LLSQ for localization is that the interval between the positions does not have to be constant (c.f. Kalman filter). This filter can only be used for local localization.

The position of a robot is represented by the state vector s (5.1). It consists of the two values for x and y and the direction φ.

$$s = [x\ y\ \varphi]^T \tag{5.1}$$

For non-deterministic moving objects like robots, second order polynomial equations are used to calculate the elements of the state vector at the time t (5.2, 5.3, 5.4). For the ball and other non-self steered objects, a polynomial equation of first order can be used (e.g. $x_t = a + bt$).

$$\begin{aligned} x_t &= a + bt + ct^2 && (5.2)\\ y_t &= d + et + ft^2 && (5.3)\\ \varphi_t &= g + ht + it^2 && (5.4) \end{aligned}$$

Equation (5.5) can be used to calculate the three elements a, b, c. Looking at the matrix notation (5.6) for this equation, the parts of the equation can be identified as the time matrix A, the value history p_x for x, and the result vector r_x containing a, b and c.

$$Ar_x = p_x \tag{5.5}$$

$$\begin{bmatrix} 1 & t_0 & t_0^2 \\ 1 & t_1 & t_1^2 \\ 1 & t_2 & t_2^2 \\ \vdots & \vdots & \vdots \\ 1 & t_{n-1} & t_{n-1}^2 \end{bmatrix} \begin{bmatrix} a \\ b \\ c \end{bmatrix} = \begin{bmatrix} x_0 \\ x_1 \\ x_2 \\ \vdots \\ x_{n-1} \end{bmatrix} \tag{5.6}$$

$$r_x = (A^T A)^{-1} A^T p_x \tag{5.7}$$

The transformed equation to calculate r_x (5.7) can be adapted to calculate the result vectors for y and φ by replacing the history-vector. The same time matrix A is used to calculate all three result vectors.

The length n of the history defines the adaption rate of the estimation to new circumstances (e.g. new primary direction). In combination with the degree of the polynomial equations (compare (5.2)), the linear least squares filter can be configured to different tasks like high vs. low update rate, fast vs. slow changing situations.

The values are stored with an increasing index for older measurements — hence, at t_0, the most recent measurement is stored whereas at t_n the oldest one. To optimize the implementation, all elements of the state vector s should be stored in one matrix together with their timestamp t. This results in the data matrix (5.8) as proposed by [Nov02, Section 5.3]. To fulfil the demand that the first row has to be the most recent measurement, the matrix has to be implemented as a FIFO (first in, first out) queue.

$$\begin{bmatrix} 0 & x_0 & y_0 & \varphi_0 \\ t_1 & x_1 & y_1 & \varphi_1 \\ t_2 & x_2 & y_2 & \varphi_2 \\ \vdots & \vdots & \vdots & \vdots \\ t_{n-1} & x_{n-1} & y_{n-1} & \varphi_{n-1} \end{bmatrix} \tag{5.8}$$

The resulting Algorithm 5.1 fills the data matrix with the incoming sensor updates in a FIFO manner. As soon as enough sensor/position updates have been collected (as defined by n), the position estimation can be calculated. Equations (5.9) and (5.10) are filling the new values for time (in this case, the time step is constantly one) and sensor readings (s) into the arrays. The old values are shifted by one position, resulting in the removal of the oldest ones. If the array is filled, and thereafter the number of iterations j is equal or larger than the size of the array n, the state can be estimated. The result vector r_x is determined using the matrix transformation described in (5.7) and the sensor reading history p_x (5.11). The components of r_x are then needed for the state estimation of x_t (5.12). To enhance this algorithm to represent more dimensions like y and φ in the state, the equations (5.10), (5.11), and (5.12) have to be replicated for each dimension.

A detailed example of the LLSQ can be found at [Nov02, pages 50–52]. It shows the localization for a MiroSOT system. At [WPF98, page 671 pp.] a detailed discussion of the LLSQ and annotations regarding its implementation are given.

Algorithm 5.1 Linear Least Squares Algorithm

```
upon initialization do
  $j \leftarrow 1$
  $A_{n,3} \leftarrow (a_{i,k})_{n,3} = \begin{bmatrix} 1 & 0 & 0 \end{bmatrix}_i$
end upon
upon sensor update $s$ do
  for $i = 1$ to $n$ do
    $$(a_i)_n \leftarrow \begin{cases} \begin{bmatrix} 1 & 0 & 0 \end{bmatrix} & \text{if } i = 1 \\ \begin{bmatrix} 1 & a_{i-1,2}+1 & (a_{i-1,2}+1)^2 \end{bmatrix} & \text{otherwise} \end{cases} \quad (5.9)$$
    $$p_{x_i} \leftarrow \begin{cases} s_x & \text{if } i = 1 \\ p_{x_{i-1}} & \text{otherwise} \end{cases} \quad (5.10)$$
  end for
  if $j \geq n$ then
    $$r_x \leftarrow (A^T A)^{-1} A^T p_x \quad (5.11)$$
    $$x_t \leftarrow a + bt + ct^2 \quad (5.12)$$
  end if
  $j \leftarrow j + 1$
end upon
```

5.4 Kalman Filter

The Kalman filter[1] (KF) — or discrete Kalman filter — is a recursive state estimator, which can deal with incomplete and noisy data. This filter was first published by R. E. Kalman [Kal60] in 1960 and thereafter named after him. It is based upon the Hidden Markov Model (HMM)[2] and linear algebra. The algorithm is divided into two update phases: (a) measurement update and (b) prediction update. In (a), new sensor values are used to refine the prediction, whereas (b) predicts the new state estimation using the refined prediction of phase (a). Based upon the HMM, the calculation is always time-constant and can be optimized.

Other than the linear least squares filter, which basically uses the mean of the last n observations to determine the new state, the Kalman filter generates and adapts a world transition model. Over time, its state estimations are getting more robust against measurement outliers.

Traditionally, the KF is used for local localization problems. New works have shown solutions how to add sensor readings of landmark sightings to the world model to enable global localization [FT06]. A more complete discussion about the problems of global localization in the domain of the KF can be found in [Neg03, page 122 pp.].

[1]Good introductions to the Kalman filter can be found in [WB95] and [May79, Chapter 1].

[2]The Hidden Markov Model is a discrete-time stochastic Markov process with unknown parameters. A process based upon the Markov property is called Markov process. The Markov property basically means that if the present is known, no additional information can be gained from the past.

5.4.1 Kalman Model

The model underlying the KF (see Figure 5.1[3]) is divided into a visible and a hidden part. The visible part consists of the system input and the sensor readings whereas the hidden part contains the state vector, the transition models, etc. As can be seen, the estimated state x at time k is based upon the previous state estimation x_{k-1} and the system input or control vector u_k. The sensor observation vector y_k is derived out of the current state x_k.

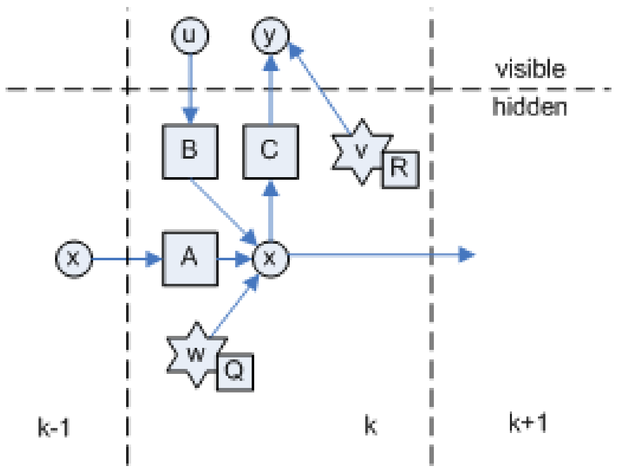


Figure 5.1. Model Underlying the Kalman Filter. [Wik06a]

$$x_k = Ax_{k-1} + Bu_k + w_k \tag{5.13}$$

$$y_k = Cx_k + v_k \tag{5.14}$$

Equations (5.13) and (5.14) are describing Figure 5.1. The first one defines the recursive state estimator $x_k \in \mathbb{R}^n$, where n is the dimension of the state vector and k is the time index representing the time in multiples of Δt. It is a combination of the transition of the last state, the system input, and noise. The transition from one state to the next is defined in the matrix A, which has a dimension of $n \times n$. Next, to determine the relation between the system input u_k and the state, the matrix B with a dimension of $n \times o$ is used. The constant o gives the dimension of u, where $u \in \mathbb{R}^o$. The system input u_k represents for example motion commands. Finally, to model uncertainty, a system noise w_k is added. The noise is white and with normal probability distribution $p(w) \sim N(0, Q)$.

Equation (5.14) gives the relation between the sensor observation vector y_k ($y_k \in \mathbb{R}^m$, where m is the dimension of the sensor vector) and the state vector x_k. This relation is stored in C. This matrix has a dimension of $m \times n$. Similarly to w_k, the variable v_k adds system noise to this relation. This noise is independent of w, but also white and with normal probability distribution $p(v) \sim N(0, R)$.

[3]Circles are vectors, squares are matrices, and stars represent gaussian noise. The stars are attached with the associated covariance matrix at their lower right side.

The matrices Q and R are the process and the measurement covariance matrices. Next to their task to add system noise, they can be used to define a priori known relations between e.g. sensor readings. Their dimensions are $n \times n$ for Q and $m \times m$ for R.

Note that the matrices A, B, C, Q, and R are considered constant in time for the discrete Kalman filter. For other versions of the KF, like the extended Kalman filter, this may change.

5.4.2 Kalman Iteration Process

The model described above is the theoretical background for the iteration process, which is the algorithm used for state estimation (see Algorithm 5.2). This algorithm is divided into the “Time Update” or prediction phase, and the “Measurement Update” or correction phase which are processed periodically (see Figure 5.2). In the first phase, the state is predicted upon the current internal state of the filter. The second phase corrects this prediction using the noisy measurements.

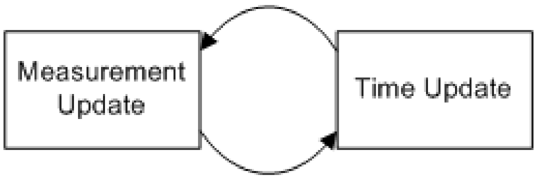


Figure 5.2. Kalman Iteration Process

Algorithm 5.2 Kalman Filter Algorithm

```
upon initialization do
    $k \leftarrow 0$
end upon
upon sensor update $y$ do
    $k \leftarrow k + 1$
    Time Update
        $\hat{x}_k \leftarrow Ax_{k-1} + Bu_k$    (5.15)
        $\hat{P}_k \leftarrow AP_{k-1}A^T + Q$    (5.16)
    Measurement Update
        $K_k \leftarrow \frac{\hat{P}_k C^T}{C\hat{P}_k C^T + R}$    (5.17)
        $P_k \leftarrow (I - K_k C)\hat{P}_k$    (5.18)
        $x_k \leftarrow \hat{x}_k + K_k(y_k - C\hat{x}_k)$    (5.19)
end upon
```

For prediction of the estimation $\hat{x}$ (Equation (5.15)) of the state vector x, only two components of the original equation (compare with Equation (5.13)) are used: The state transition and the system input. The intermediate error covariance matrix $\hat{P}$ with the dimension $n \times n$

describes the estimated accuracy of the state estimation. In Equation (5.16), the trustworthiness of the prediction is lowered by the process covariance matrix Q.

The Kalman gain K (Equation (5.17)) describes the proportion between how much the predicted state can be trusted, and up to which extent the new measurements have to be integrated. Ideally, the predicted state is accurate enough that the noisy sensor data is not needed for precise state estimation. $\hat{P}$ is updated with the Kalman gain in Equation (5.18) resulting in the error covariance matrix P. Thus, K and P are recursively influencing each other. I is the identity matrix with dimension $n \times n$. Finally, the new state x is obtained by adding to the predicted state $\hat{x}$ the deviation of the actual and the predicted measurement in proportion to the Kalman gain (Equation (5.19)).

5.4.3 Example

The following example, which demonstrates the KF, is an adoption of the example found in [Wen00b][4]. The mean resistor value of a set of 100 resistors should be determined[5]. The available measurement device has a standard deviation of 13 Ω, the quality of the mean value should not exceed a standard deviation of 3 Ω.

An intuitive approach towards this problem is to build the average of all measured values. Alternatively filtering techniques like KF can be used. While the average algorithm needs no initial information, the KF needs to be well initialized. The parameters used for this example are: $A = [1]$, $B = [0]$, $C = [1]$, $R = \left[13^2\right]$, $Q = [0]$, $P_0 = \left[3^2\right]$, and $x_0 = [105]$. The values used for A, B, and C are simply describing an average algorithm implemented with the KF. R and P_0 are part of the specification (standard deviations for measurement device and mean). For x_0, a priori knowledge — also known as a good guess — is needed.

As can be seen in Figure 5.3, the KF reaches a stable value for mean earlier than the average algorithm. Stable means in this case that the estimated values are staying within the standard deviation around their average. Additionally, the filter is more robust against outliers, especially during the first 20 steps.

Table 5.1 shows that both algorithms are returning good results for the standard deviation of the mean value and both produce a minimum error of 0 Ω. The maximum error shows the difference between the average algorithm and the KF best: while the first one produces results even outside the standard deviation of the measurement device, the latter one is even with its worst case almost inside the target area of ±3 Ω.

In Table 5.2, the results of 1000 runs of this example are shown. Similarly to the single run experiment, the KF outperforms the average algorithm by filtering out the outliers. Data analysis has shown that the average algorithm needs up to 60 measurements to stabilize while the KF needs at most 15 steps.

[4]The series of publications [Wen00a], [Wen00b], [Wen00c], and [Wen00d] are a good introduction to the Kalman filter in German.

[5]All 100 resistors are of perfect quality and have a value of 100 Ω.

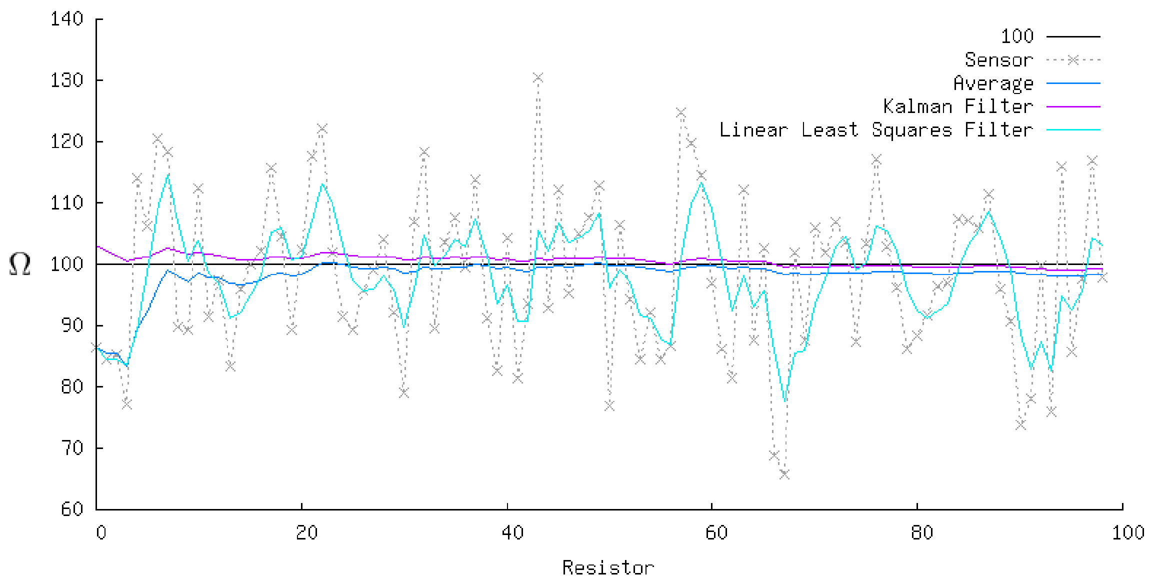


Figure 5.3. Resistor Test Results

Table 5.1. Statistics for the Resistor Test Example

	Max.Error	**Min.Error**	**Avg.Error**	**better than Sensor**
Sensor data	34.4	0.0	10.9	0%
Average	16.7	0.0	1.9	94%
KF	3.1	0.0	0.9	94%
LLSQ	22.5	0.1	6.5	72%

Concluding, the KF and the average algorithm produce results which are within the given range. The advantage of the KF lies in filtering out outliers and in an earlier stabilization of the resulting value. It should be mentioned that, if the KF is initialized with a value for x_0 far from the real value, the filter may not be able to produce a stable, correct result within 100 iterations, while the average filter recovers from bad initialization values.

The linear least squares filter has been added to this example to visualize the differences between it and the KF. As can be seen in Figure 5.3, the LLSQ estimates the next value upon the last three measurements. Thus, the LLSQ is sensitive towards value changes and value change tendencies. As a result, no stable mean can be produced. If we take a completely

Table 5.2. Statistics after 1000 runs

	Max.Error	**Min.Error**	**Avg.Error**	**better than Sensor**
Sensor data	67.8	0.0	10.4	0%
Average	39.3	0.0	2.0	89%
KF	7.2	0.0	1.6	90%
LLSQ	39.3	0.0	5.6	75%

different example — non-linear robot path estimation — this characteristic produces better results than the discrete KF, which can only slowly adopt to a new movement direction. For dealing with the problem of non-linearity, the KF has to be extended.

5.4.4 Non-Linear Kalman Filters

To be able to use the KF in non-linear environments, the data has to be linearized using methods like Taylor expansion. This linearization has to be calculated in each cycle, resulting in a decrease of performance. The most common non-linear Kalman filters are the extended Kalman filter (EKF) and the unscented Kalman filter. A short overview of the EKF is given in this section.

Extended Kalman Filter

The matrices A, B, and C used in Equations (5.13) and (5.14) are constant over time for the discrete case. The EKF replaces the linear transition model $Ax_k + Bu_k$ and the linear measurement model Cx_k with the non-linear functions g and h (Equations (5.20) and (5.21)). Thus, the resulting belief of the estimation is no longer represented by a gaussian distribution.

$$x_k = g(u_k, x_{k-1}) + w_k \tag{5.20}$$

$$y_k = h(x_k) + v_k \tag{5.21}$$

In Algorithm 5.3 — which is similar to Algorithm 5.2 — the non-linearization of the matrices A, B and C is done using Taylor approximation and Jacobians. The matrices A and B are represented by the Jacobian G_k, and the matrix C by H_k.

5.5 Markov Localization

The linear least squares filter and the Kalman filter are only able to represent one state estimation. If e.g. based upon the current sensor readings the robot can be at two different positions but not in between, other approaches have to be used. An intuitive approach is to use topological or grid maps of the environment as discussed in Section 4.2.1. For each possible position, the possibility of the robot to be there is calculated based upon the previous distribution and the actual sensor readings. Finally, the area with the highest belief is selected as estimated position. In case of global uncertainty — several not connected areas; each area with a high possibility — the robot has to continue moving until the possibility for all but one area have decreased, and, therefore, only one possible estimation of the robot position remains. Like the Kalman filter, the Markov localization (ML) is based upon the Markov assumption. Thus, all past information is already inside the model through recursively applying the algorithm and sensor data on one set of locations.

In the resulting Algorithm 5.4, initially no position is known. Thereafter all locations have a uniform belief (Equation (5.27)). In case the starting position is known, the initialization

Algorithm 5.3 Extended Kalman Filter Algorithm

```
upon initialization do
    $k \leftarrow 0$
end upon
upon sensor update $y$ do
    $k \leftarrow k + 1$
    Time Update
        $\hat{x}_k \leftarrow g(u_k, x_{k-y})$   (5.22)
        $\hat{P}_k \leftarrow G_k P_{k-1} G_k^T + Q_k$   (5.23)
    Measurement Update
        $K_k \leftarrow \frac{\hat{P}_k H_k^T}{H_k \hat{P}_k H_k^T + R}$   (5.24)
        $P_k \leftarrow (I - K_k H_k)\hat{P}_k$   (5.25)
        $x_k \leftarrow \hat{x}_k + K_k(y_k - h(\hat{x}_k))$   (5.26)
end upon
```

of the localization beliefs can be adopted to this a priori knowledge. In each cycle, the location probabilities are updated. This is not only done using sensor readings but also — as an extension to the intuitive approach — actions executed by the system. Equation (5.28) applies the actions. E.g. if the robot moves one step right, all position estimations can be moved accordingly. After the sensor inputs have been applied (Equation (5.29))[6], all beliefs have to be normalized[7] (Equation (5.30)). The sum of the beliefs for each location has to be 1. The normalization step can be skipped if no new sensor inputs are available.

For each element that has to be considered for localization, another dimension is added to the state space. Thus, for the typical case of x, y and φ and grid representation of the locations, we have a three dimensional grid (see Figure 5.4).

For a soccer field of 2 by 2.5 meters^2 and a resolution of 2 cm×2 cm×5°, the resulting array has 900,000 cells, each filled with the belief of the actual position. Even with optimization techniques, the time needed to process this large array is problematic. As mentioned above, the space can be reduced using topological maps, but this may not be possible for all applications. Furthermore, even topological maps can get huge.

This localization method can easily be used not only to observe the self position, but also

[6]In case of distance sensors or landmark sightings, the belief $P(s|\ell)$ in Equation (5.29) can be expressed by Equation (5.31) [Fox98].

$$P_m(d_i|\ell) = \frac{1}{\sigma\sqrt{2\pi}} e^{-\frac{(d_i - o_i)^2}{2\sigma^2}} \tag{5.31}$$

Here d_i, denotes the measured distance with a sensor, which has a standard deviation of σ. The measured distance is compared with the real distance o_i at location ℓ to the next known obstacle.

[7]To avoid degeneration problems, the belief should be lower bounded to an ϵ which should be small enough not to distort the result.

Algorithm 5.4 Markov Localization Algorithm

```
upon initialization do
  for all locations $\ell$ do
        $Bel(L_0 = \ell) \leftarrow P(L_0 = \ell)$    (5.27)
  end for
end upon
upon update do
  if action $a$ is executed then
    for all locations $\ell$ do
        $P(L_t|L_{t-1}, a_{t-1}) \leftarrow \sum_{\ell'} P(L_t = \ell|L_{t-1} = \ell', a)Bel(L_{t-1} = \ell')$    (5.28)
    end for
  end if
  if sensor input $s$ is perceived then
    for all locations $\ell$ do
        $Bel(L_t = \ell) \leftarrow P(s|\ell)P(L_t|L_{t-1}, a_{t-1})$    (5.29)
        $Bel(L_t = \ell) \leftarrow \frac{Bel(L_t = \ell)}{P(s|L_t)}$    (5.30)
    end for
  end if
end upon
```

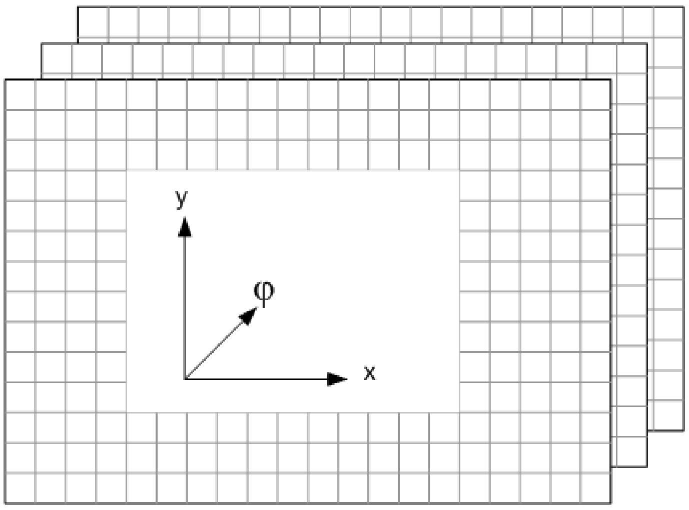


Figure 5.4. Three-Dimensional Grid for Markov Localization

to observe several moving entities like the ball and other robots. Each area which is rated with a high belief represents at least one of these entities. Thus, it is a powerful tool for sensor data integration.

5.5.1 Example

The following example visualizes the Markov localization algorithm very well. It is taken from [FBT99]. In this example, a robot moves along a hall way and has to determine its position starting from global uncertainty.

First, the robot is initialized in the hallway with no information about its whereabout (Figure 5.5(a)). A priori knowledge like the floor plan has been provided. The robot immediately senses a landmark — a door (Figure 5.5(b)). Applying these sensor readings to the Markov algorithm, all positions which are not close to a door are rated with a very low belief. In Figure 5.5(c), the robot starts moving to the right. Applying this action to the state space results first in a shift to the right for all three possible positions, and second, due to the uncertainty of the movement, the gaussian peaks are lowered and widened. Finally, the robot senses a second door (Figure 5.5(d)). With this second landmark the global position of the robot is known — no other than the second door has a door very close to its left.

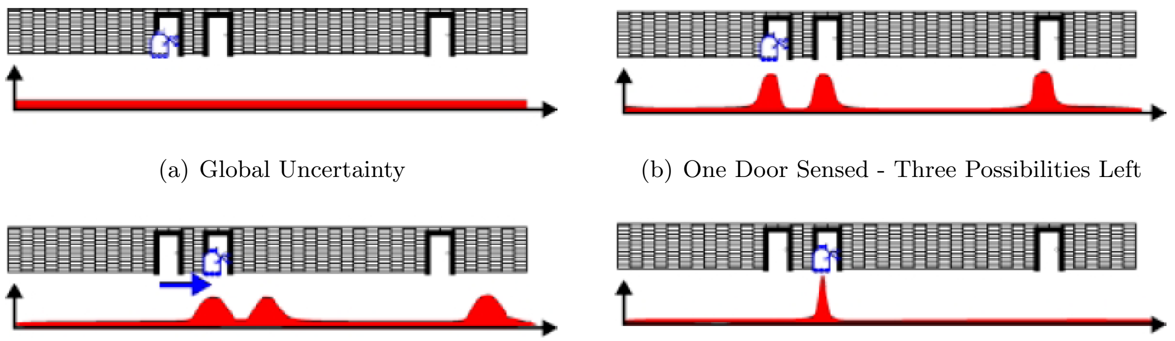

(a) Global Uncertainty
(b) One Door Sensed - Three Possibilities Left
(c) Movement Increases Global Uncertainty
(d) Another Door Sensed - Self Position Found

Figure 5.5. Example for Markov Localization [FBT99].

5.6 Particle Filter

Like mentioned above, the Markov localization can only be optimized to some extent. For larger environments, it is certain that either the precision or the calculation time is getting problematic. To overcome this, the number of cells has to be reduced without deterioration of the desired precision. The Particle filter — or Monte Carlo Localization (MCL) — replaces the dense grid representation by a much lower number of particles. These particles are state vectors representing the belief that the true location is exactly here. Applying various techniques — like resampling — the desired goal of a high precision with a low number of cells can be reached. Particle filters are state of the art in robotics for position estimation.

The basic concept for the algorithm is a Monte Carlo approximation with Sequential Importance Sampling (SIS). Initially, random particles are generated and their weight is set to the reciprocal value of the number of particles. In each cycle, the weight of a particle is calculated by multiplying its old weight with the probability that this particle represents the real state regarding the newest sensor readings. After normalization of the weights, the real state can be estimated using them.

The SIS step — multiplying the old weight with the new probability — leads to degeneration. After several cycles, particles with a low probability are converging towards zero. Even if the sensor readings suggest that one of these particles should increase its weight, the previously strong particles will override this and stay strongly weighted. Thus, once a degree of certainty of the estimation has been reached, the model cannot change its estimation any more.

One possibility to avoid this is to replace SIS with Sampling Importance Resampling (SIR). Here, the weight of each particle is determined each round independently from the previous one. After estimation of the state, a new set of particles is drawn from the actual set. This new set is used as basis for the next cycle. The particles are drawn in such a way that more important ones are more likely to be picked. Thus, areas with a high probability are gaining more weight by replicating their particles, whereas areas with a low probability are thinning out. Once an area has no particle left, another source for degeneration displays — due to the lack of appropriate particles, local maxima will dominate the estimation.

With the introduction of reinjection — some particles are replaced by uniformly distributed new ones — empty areas can be refilled regularly. The decision when and how many particles should be replaced by these reinjected ones can be done in several ways. Usually, either a fixed number of particles is reinjected each round, or, if the quality of the estimation is below a threshold, reinjection is carried out[8].

For additional reduction of the particles, several other methods like dynamically adopted number of particles and Rao-Blackwellization (components with linear dynamics can be estimated using the Kalman filter and feed-back into the particle filter) can be used. All of these optimizations are always a tradeoff between additionally needed computation power and the number of particles. The number for a given application should be selected carefully.

Algorithm 5.5 shows a classical particle filter with SIR as the only enhancement. During initialization, all of the N particles are generated by distributing them uniformly over the state space (Equation (5.32)). At the beginning of every cycle, the cycle index k is increased and temporary particles $\tilde{x_k}^{(i)}$ are generated out of the state space x_k under the conditions of the particle from the previous cycle $\tilde{x_{k-1}}^{(i)}$ and the command control[9] u_k (Equation (5.33)). Using the actual sensor data y_k, the weight $\tilde{w_k}^{(i)}$ for the particles is calculated in Equation (5.34). In (5.36), the normalized weights of Equation (5.35) are used to calculate the state estimation for x_k under the condition of all previous sensor readings $y_{1:k}$. As preparation for the resampling step, all tuples $\left\{\tilde{x_k}^{(i)}, \tilde{w_k}^{(i)}\right\}$ have to be sorted in descending order with the highest weight-value first. During resampling, N times a random number j is drawn in such

[8]Note that re-injecting particles decreases the quality of the estimation. If no counter measures are implemented, after injection of the first set of particles, in every cycle more and more particles are replaced. This results in a complete “breakdown” of the state estimation.

[9]See also Kalman filter Equation (5.13).

Algorithm 5.5 Particle Filter Algorithm

```
upon initialization do
  // generate random particles
  for all $i \in N$ do
      $$x_0^{(i)} \sim p(x_0) \tag{5.32}$$
  end for
  $k \leftarrow 0$
end upon
upon sensor update $y$ do
  $k \leftarrow k + 1$
  for all $i \in N$ do
    // generate particle
      $$\tilde{x_k}^{(i)} \sim p\left(x_k | u_k, x_{k-1}^{(i)}\right) \tag{5.33}$$
    // calculate weight/probability of particle
      $$\tilde{w_k}^{(i)} \leftarrow p\left(y_k | \tilde{x_k}^{(i)}\right) \tag{5.34}$$
  end for
  // normalize particle weights
  for all $i \in N$ do
      $$\tilde{w_k}^{(i)} \leftarrow \tilde{w_k}^{(i)} \left[\sum_{j=1}^{N} \tilde{w_k}^{(i)}\right]^{-1} \tag{5.35}$$
  end for
  // estimate current state
      $$E(g(x_k | y_{1:k})) \leftarrow \sum_{j=1}^{N} g\left(\tilde{x}_k^{(j)}\right) \tilde{w}_k^{(j)} \tag{5.36}$$
  // resample particles
  sort $\left\{\tilde{x_k}^{(i)}, \tilde{w_k}^{(i)}\right\}_{i=1}^{N}$ such that $\tilde{w_k}^{(i)} > \tilde{w_k}^{(i+1)}$
  for all $i \in N$ do
    draw j with probability $\tilde{w_k}$
      $$\left\{x_k^{(i)}, \frac{1}{N}\right\} \leftarrow \left\{\tilde{x_k}^{(j)}, \tilde{w_k}^{(j)}\right\} \tag{5.37}$$
  end for
  // re-inject random particles
  for all $i \in T$ do
    replace $x_k^{(i)}$ with new random particle
  end for
end upon
```

a way that tuples with a higher weight are preferred. For each j, the corresponding $\tilde{x_k}^{(j)}$ is drawn and assigned to the final state $x_k^{(i)}$ (Equation (5.37)). All final state vectors are assigned with a uniform weight. To avoid local maxima, finally, T particles are replaced with newly generated uniformly distributed ones. T is much smaller than N, typically 100 times and more.

5.6.1 Example

This example is in principle similar to the example for the Markov localization (Section 5.5.1) but more complex. It is taken from [Thr02a][10]. The robot knows its environment, but does not know where it is at the moment of initialization.

The four subsequent pictures in Figure 5.6 show a robot, which operates in an office and tries to perform a global self localization. In all pictures, the walls are grey, the robot is green with a small blue line denoting the heading direction, the values of the sensor readings of the 24 distance sensors[11] are represented by the blue rays emitting from the robot, and the multidimensional particles are represented by the red dots.

In Figure 5.6(a), the robot is freshly initialized with all particles uniformly distributed over the area. After moving from one room to the hallway (Figure 5.6(b)), almost every other possibility than being in the hallway has been eliminated. When moving into another room in Figure 5.6(c), the robot has only two — but equally likely — possibilities for its global location left. Only few particles to the right of the robot are disturbing the otherwise perfectly estimated position (Figure 5.6(d)). Once a reliable position estimation has been found, it is unlikely that more disturbances than shown in the last figure will occur.

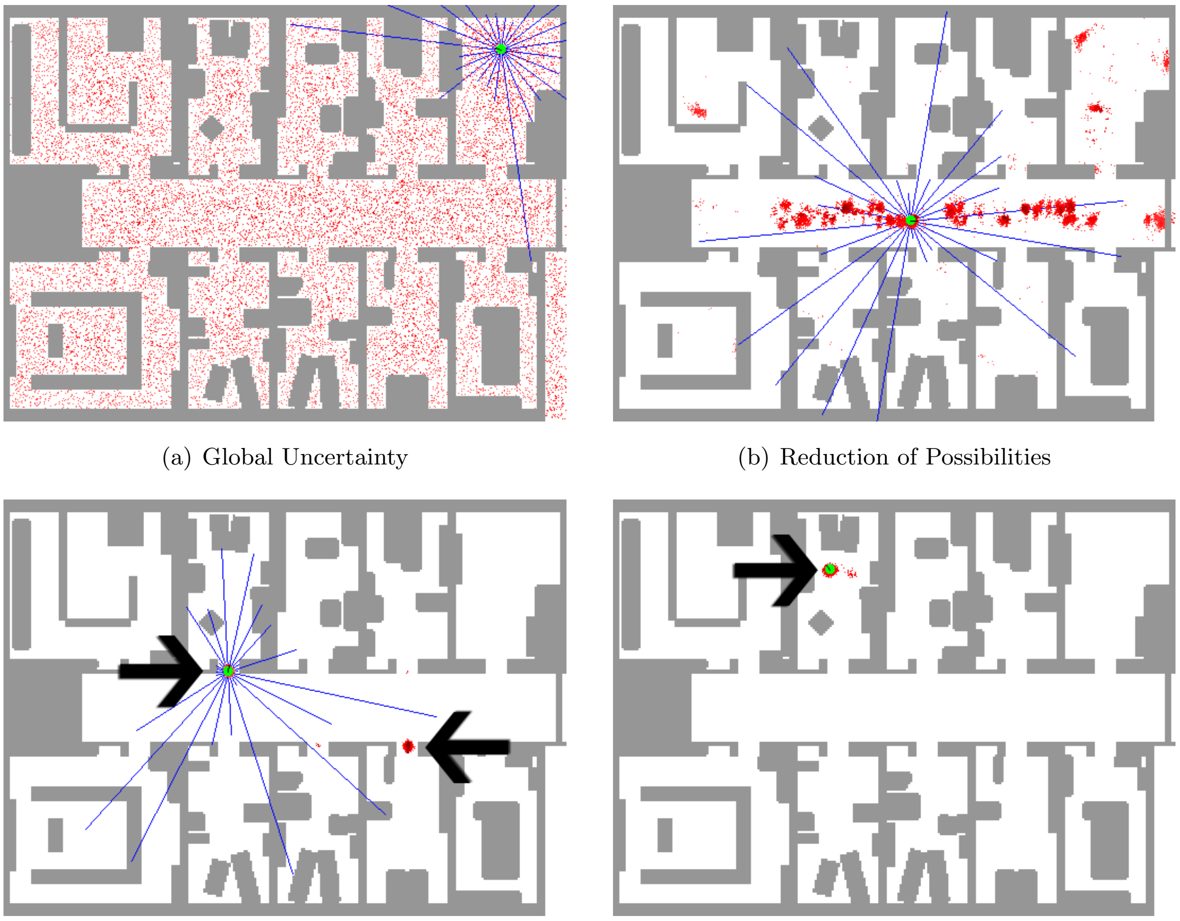
5.7 Comparison Experiment

To compare all four localization algorithms handled in this chapter — linear least squares algorithm, Kalman filter, Markov localization, and particle filter — a simulation has been carried out during this work. The aim of this simulation is to track a single object, which is observed by a non-moving single vision system, which is mounted above the area.

The object to be tracked is a robot, which travels at a constant speed in total two meters. The outputs of the vision system are the absolute position coordinates x and y, but no direction ϕ. The standard deviation of the perceived position is six centimeters from the real position. The robot moves at constant speed, and during the simulation run, 100 data points are collected. Thus, the robot travels 2 cm per time frame. The localization algorithms have to estimate the real robot position using only the few unreliable sensor readings. Additional data like odometry, multiple objects, and motion control have been left out intentionally.

[10]The example as an animation can be found at `https://www.cs.washington.edu/ai/Mobile_Robotics/projects/index.html`

[11]The precision of these sensor readings is very low or even faulty. Some of the blue rays are cutting through walls, others are too short without an obstacle present.



(a) Global Uncertainty (b) Reduction of Possibilities

(c) Two Positions Left (d) Real Position Found

Figure 5.6. Global Localization using Particle Filter [Thr02a]

5.7.1 Configuration

All localization methods use the algorithms described in this chapter. The following list gives only information about the parameters and optional elements for each algorithm.

Linear Least Squares Algorithm Has a queue length of 5 steps. It uses a polynomial equation of second order to approximate the position.

Kalman Filter The matrices are initialized as follows:

$$A = \begin{bmatrix} 1 & 0 & 1 & 0 \\ 0 & 1 & 0 & 1 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}, \; B = \begin{bmatrix} 0 \\ 0 \\ 0 \\ 0 \end{bmatrix}, \; C = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \end{bmatrix}, \; P_0 = \begin{bmatrix} 3 & 0 & 0 & 0 \\ 0 & 3 & 0 & 0 \\ 0 & 0 & 3 & 0 \\ 0 & 0 & 0 & 3 \end{bmatrix},$$

$$Q = \begin{bmatrix} 0.14 & 0 \\ 0 & 0.14 \end{bmatrix},\ R = \begin{bmatrix} 3600 & 0 & 0 & 0 \\ 0 & 3600 & 0 & 0 \\ 0 & 0 & 3600 & 0 \\ 0 & 0 & 0 & 3600 \end{bmatrix},\ u_0 = \begin{bmatrix} 0 \end{bmatrix},\ x_0 = \begin{bmatrix} 0 \\ 0 \\ 1 \\ 0 \end{bmatrix}$$

The state vector x consists of four components: x, y, δx, and δy. Thus, the state transition matrix is filled in such a way that each cycle the new position is approximated with the addition of the delta values to the previous position. The sensor matrix C correlates the sensor readings to the first two elements of the state vector. To guarantee a standard deviation of the estimation of not more than 3, the diagonal of P_0 has been set to this value. The error covariance matrix for the measurement has been set to the variance of the vision sensor. Q has been determined using optimization experiments[12]. The motion control u is set to 0 because no information about it is available. This is also the reason why the KF and not the EKF has been used. For this setup, the EKF would only be a more complex implementation of the KF with no gain of precision.

Markov Localization The grid is configured to have a cell size of 5×5 cm^2. As action update, the probability for the vicinity of the estimated position is increased. This is done applying a multi-variant distribution with a σ much larger than the sensor error.

Particle Filter 1000 particles are used. Further, in each cycle, 10 particles are reinjected. No action update is used.

5.7.2 Results and Interpretation

Figure 5.7 and Table 5.3 show error statistics for the path estimations depicted in Figure 5.8. The path given by the sensor (Figure 5.8(b)) is perceived by the vision system using the real path of the robot (Figure 5.8(a)). Sensor data is also used as a benchmark for the localization algorithms — only if such a localization is on average better than the perceived sensor data, the effort is worth while.

Table 5.3. Statistics for the Comparison Experiment

	Max.Error	Min.Error	Avg.Error	better than Sensor
Sensor data	176.6	3.6	68.6	0%
LLSQ	100.1	9.5	51.1	67%
KF	126.6	13.2	57.7	54%
ML	189.9	28.9	110.8	22%
MCL	153.6	5.5	51.8	76%

[12]The simulation is implemented in AnyLogic. This simulation environment allows to define a function which gives a quality measure for the simulation outcome. This function together with the information which parameters are searched — and, thus, changeable — enables the system to carry out optimization experiments. The outcomes are approximated optimal parameters for the algorithm.

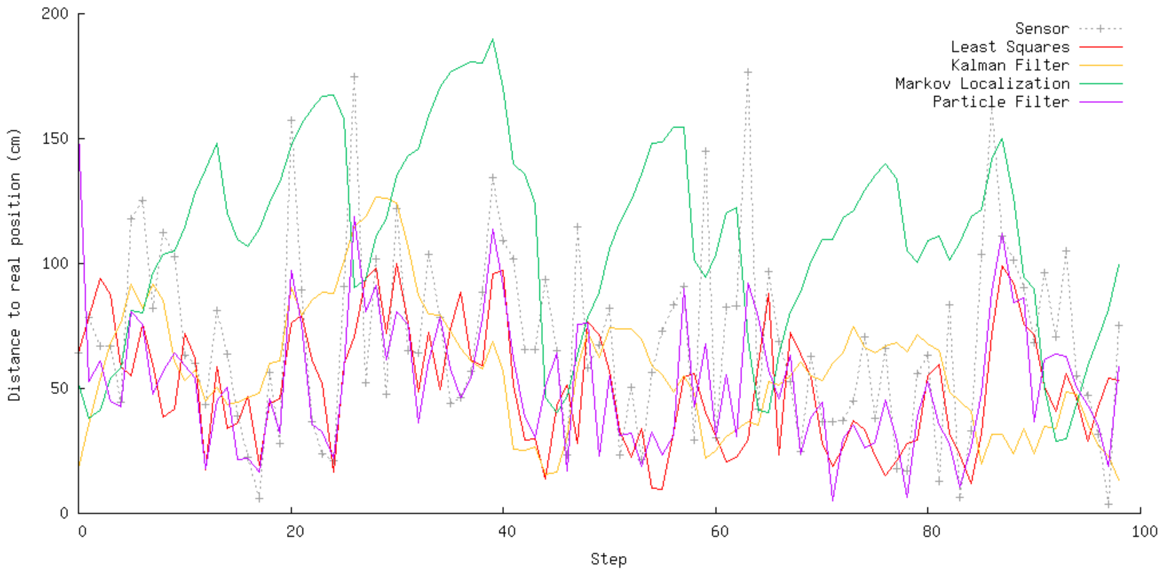
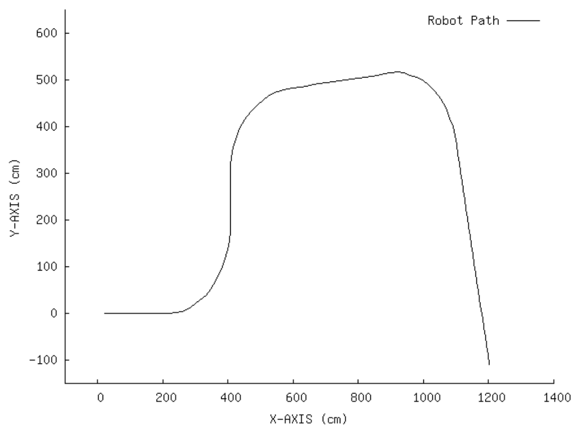


Figure 5.7. Distance Between the Real Position and the Estimated Positions

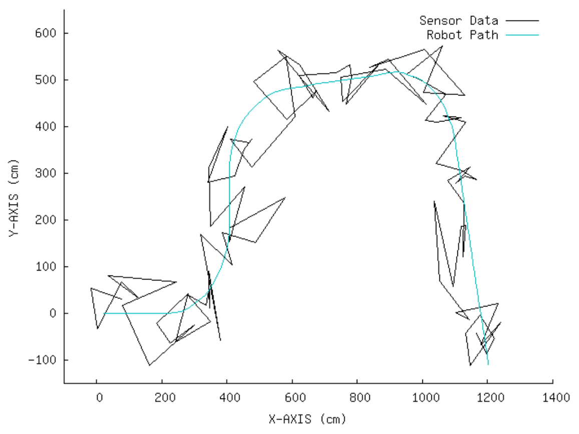
Although the estimated path of the linear least squares algorithm moves sometimes back and forth (see Figure 5.8(c)), the overall performance is reasonably well good. It is second best compared to the benchmark, and in the category of the average distance to the real position it is the best. Also, the maximum error is the lowest in the test.

The Kalman filter follows the real path even with these faulty sensor readings (Figure 5.8(d)). The delayed adaption to course changes is systematic. To overcome it, additional information like motion commands is needed to apply the EKF. According to the table, the KF is second worst in the overall performance, only outperforming the Markov localization. But if the shape of the path is more important than the error distance to the real path, the KF is the best choice.

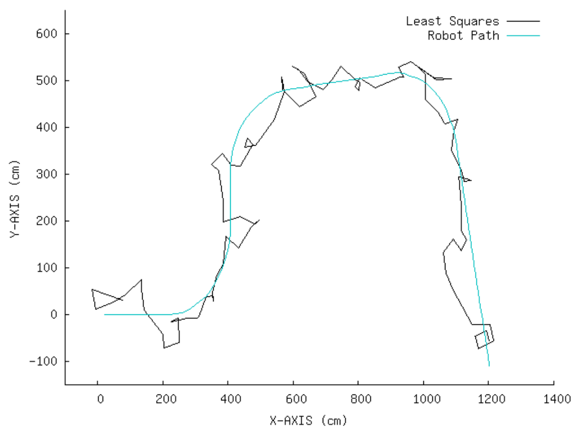
The worst result in this test is the one of the Markov localization. While Figure 5.8(e) suggests the same result interpretations like for KF, the error distances in Figure 5.7 are showing that the estimated path is on average twice as bad as the other estimations. A closer look at Figure 5.8(e) explains what happens — after the last estimated position, the path is still far too short. Hence, the estimated positions were right, but a few cycles too late. This is the result of the degeneration problem. As can be seen in Figure 5.9, a once selected position (dark area) is still active, although the real position (red point) has moved on. An estimated position tends to stay active the next cycle due to the normalization phase at the end of each cycle. After a while, this “the winner takes it all”-situation is ended by a strong difference between the real and the estimated position. Due to algorithmic racing conditions and numeric imprecisions, three new areas claim to be the real position. Once again — the winner takes it all — the area with the highest belief remains. After the resulting jump in the estimation, the real position and the estimated position are matching again. The delay in adapting to the real positions is the reason for the observed large error rate.



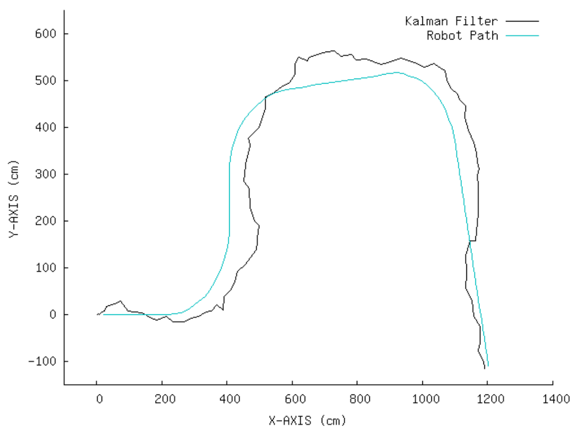
(a) Real Robot Path



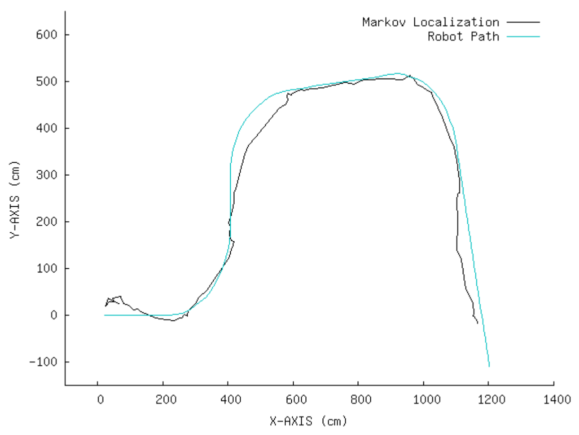
(b) Sensor Data



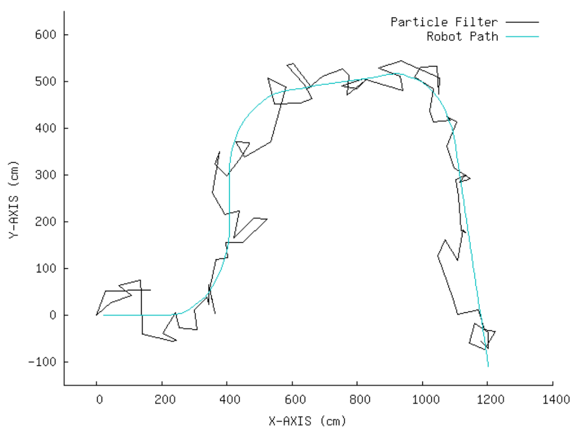
(c) Linear Least Squares Filter



(d) Kalman Filter



(e) Markov Localization



(f) Particle Filter

Figure 5.8. Comparison of Localization Algorithms

Figure 5.9. Three Succeeding Steps in the Markov Localization.

Finally, the particle filter produces similar results as the LLSQ. If the state vector is extended by the direction φ, the MCL and the other two algorithms would outperform LLSQ. In LLSQ, each dimension is treated independently, while the other algorithms are putting them into relation. Thus, if there is a main movement direction, sensor readings, which suddenly change the direction to the opposite are more likely to be discarded.

Because no code optimizations were done, timings of the different algorithms are not shown here. However, it can generally be said that the LLSQ is the fastest, shortly followed by the KF. The MCL is slower by a power of ten but this number varies for different numbers of particles. The slowest algorithm is the ML. Comparison experiments with more optimized implementations can be found in [GBFK98] and [GF02].

In this setup, ML and KF were more robust against data outliers than LLSQ and particle filter. This advantage was gained at the cost of flexibility towards course changes. LLSQ is the winner of this simple setup, while the results of the MCL will increase in quality with the introduction of further state dimensions, action update, and sensor readings.

5.8 Summary

The simulation experiment presented in Section 5.7 showed that there cannot be such a thing as a general optimal localization algorithm. The choice for the best algorithm for a given application depends on several parameters, like available performance, type of objects to be tracked, robustness against outliers, or flexibility towards course changes. Table 5.4 gives an overview of the strengths and weaknesses of the different algorithms. It is based upon the comparison table given at the end of the 8th chapter of [FT06].

The first five rows are showing clearly that the ability to deal with multiple hypotheses is strongly related with the ability to handle raw data with any type of noise. Further, multiple hypotheses are needed for global localization. Optional sensor data is necessary for the case that the vision system needs 5 to 10 cycles to generate new landmarks. If the algorithm demands all sensor data in every round, this either leads to a decrease of cycles to fit the worst case, or the landmarks from the vision system have to be approximated if no new data is available.

The next three rows are about efficiency. LLSQ, KF, and EKF are more efficient than ML and MCL — they only store a small state space. ML with the largest state space also takes longest, and the precision is always lower bounded to the cell size (see Section 4.2.1).

In the last two rows, robustness and flexibility are shown. Here, the EKF is at its best. KF

Table 5.4. Comparison of the Localization Algorithms

	LLSQ	**KF**	**EKF**	**ML**	**MCL**
Measurements	landmarks	landmarks	landmarks	raw	raw
Noise	Gaussian	Gaussian	Gaussian	any	any
Optional sensor inputs	no	no[13]	no[13]	yes	yes
Global Localization	no	no[13]	no[13]	yes	yes
Multi-Hypothesis	no	no[13]	no[13]	yes	yes
Efficiency (time)	++	++	+	−	+
Efficiency (memory)	++	++	++	−	+
Resolution	++	++	++	−	+
Robustness	−	++	++	−	+
Flexibility	++	−	+	−	+

and LLSQ are serving either the robustness or the flexibility, while ML has problems serving any of the two.

In this overview, MCL is always good. Thus, for most applications, it is the algorithm of choice.

[13]There are extensions for the KF and the EKF where the observed landmarks are handled as features and for every feature the measurement update round is initialized. Thus, optional sensor readings and global localization are then possible.

6 Tinyphoon

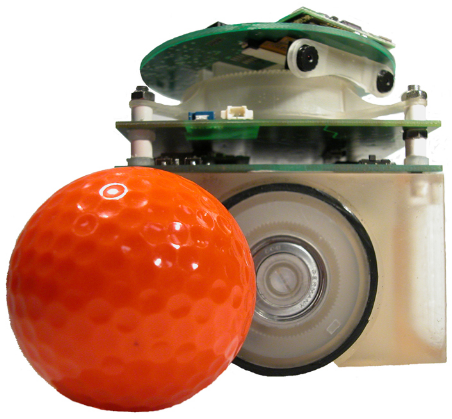


Figure 6.1. Tinyphoon2005 with an Orange Golf Ball.

The Tinyphoon robot (see Figure 6.1) is developed for the FIRA MiroSOT League and the AMiroSOT League [NM04]. The ambitious goal, is to create a team of autonomously cooperating soccer playing robots, each of them not larger than an area of 7.5×7.5 cm^2. To fulfill this goal each robot needs a camera system among other sensors, radio communication, and a decision module. This chapter gives an overview about these components and the modules of the robot.

Tinyphoon is permanently improved. The resulting versions differ more or less in every component. Especially the used processors and the vision system are constantly equipped with increasing capabilities. The Tinyphoon 2006 described in this work is the latest version with a highly advanced vision system. While the hardware design has been finished — except for some minor adaptions — the software is still work in progress. To assist the software development of the different modules, simulators have been written.

In this chapter, the Tinyphoon platform and its control concept [NRB+06] will be introduced. To follow the concept of Chapter 3 — "Autonomous Mobile Robots" — , first the hardware (Section 6.1) and the sensors (Section 6.2) are described. Afterwards, the software architecture (Section 6.3) and the five step pipeline (Section 6.4) are introduced. For communication, the real time data oriented communication system TTP/A is used. TTP/A and the data exchanged will be shown in Section 6.5. Software tools for remote control of the Tinyphoon are listed in Section 6.7. Finally, the simulators developed for Tinyphoon are listed in Section 6.8.

6.1 Hardware

The robot is divided into different units — Vision Unit, Decision Unit, Motion Unit — which are packed into a volume less than $7.5 \times 7.5 \times 9$ cm^3 (Figure 6.2). The small size and the little weight (less than 450 g including batteries) enable the robot to reach a maximum speed of 3.5 m/s with an acceleration of 5 m/s^2. The size of the playgrounds used in robot soccer are ranging from 1.5×1.3 m^2 to 4.0×2.8 m^2. Thus, a Tinyphoon robot can reach almost every position in less than one second. During the design phase of the chassis, special care was taken to keep the center of gravity as low as possible enabling a very high speed when doing turns.

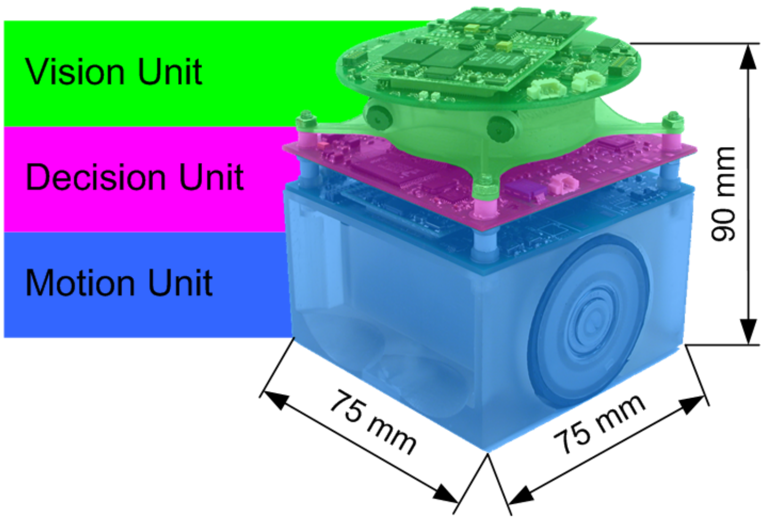


Figure 6.2. The Size and the Three Modular Units of Tinyphoon.

Vision Unit The Vision Unit consists of a Blackfin Dual Core Processor Module @ 2x600 MHz, a Blackfin Single Core Module @ 600 MHz, and a stereo vision camera capable of 30 fps. Additionally, this unit is built upon a turnable turret, which enables it to follow the ball with the "eyes" while moving into a different direction. The outputs of the Vision Unit are global self localization, the relative ball position, and relative positions of visible robots. It accepts absolute head rotation coordinates.

Decision Unit The Decision Unit is a single board, which hosts a XSCALE 32-Bit Embedded Platform @ 400 MHz and the Real-Time Wireless Interface (1 Mbps, 2.4 GHz) for external communication purposes. Additionally, the I^2C bus for sensor communication is placed here. This units hosts three different software modules: decision making, world model repository, and communication. The Decision Unit collects all sensor data — from the Vision Unit, Motion Unit, and other robots — fills them into the world model repository and calculates the next action to be executed by the Motion Unit.

Motion Unit The Motion Unit is the basic platform upon which the other two units reside. It contains two wheels, each equipped with a single stage gear, two DC Motors with magnetic Dual-Channel Encoder (512 Pulses/Rotation), a Li-Ion Rechargeable Battery 2.3 Ah @ 7.2 Vm, and the XC167 Automotive μController including a Blackfin Single Core Co-Processor Module. Additionally, the Motion Unit also hosts the odometry sensors: compass, gyro, acceleration (both directions). For fast ball detection, an additional mono vision camera system is integrated in the chassis. This camera can take pictures at a rate of 60 fps. The Motion Unit provides the local self localization and accepts action commands from the Decision Unit which are executed using trajectories.

The units are connected via serial communication interfaces (UART) among each other. The communication between the units is implemented with TTP/A (see Section 6.5). This communication bus allows adding and removing of nodes. Due to this modular design, new units can be added — e.g. the world model repository is moved from the Decision Unit to a new unit equipped with its own processor to add more calculation power.

6.2 Sensors

To enable the Tinyphoon robot to navigate autonomously, it is equipped with a broad range of sensors. They can be grouped into three categories: odometry, infrared, and vision. The first category focuses on internal sensor data while the other two are collecting external sensor data.

6.2.1 Odometry

As described in Section 3.2, an odometry sensor is not a distinct sensor, but a system which integrates several sensors into one set of data — the relative movement. The Motion Unit provides an interface to fetch this set of data in a manner indistinguishable from other sensor readings.

Currently, the implemented approach to odometry is simple — the steps of the wheel encoders are integrated into Δx, Δy, and $\Delta\alpha$, where $\Delta\alpha$ is derived from Δx and Δy. The encoders have a resolution of 600 steps per turn; data is updated at a rate of 600 Hz.

Because no other sensor-information is used, influences like slip and drift are ignored. Thus, the resulting odometric data is of good quality only for slow movement and short distances. For a planned trajectory of 2.6 m straight ahead, tests have shown an average positioning error of $\Delta x = \pm 2$ cm and $\Delta y = \pm 20$ cm. The strong deviation in y-axis is the result of a bad calibration for the slightly different diameters of the wheels[1]. With increased speed (more than 1 m/s) and/or a lot of turns, the quality drops drastically. Thus, navigation should trust solely on odometric data only for short distances.

From the sensors of the Motion Unit listed in Section 6.1, currently only the Dual-Channel Encoders are used. While problems like variable size of wheels and mechanical differences between left and right side can be solved with calibration, other problems like drift and slip cannot be addressed, making other — more sophisticated — approaches necessary. In [NS04] a trajectory controlling algorithm using all sensors is introduced. [SJ05] additionally introduces neural networks to cope with slip and drift.

6.2.2 Infrared

Two short range and two long range infrared (IR) distance sensors are applied to the Tinyphoon platform. The short range IR-sensors — Sharp GP2D120 — General Purpose Type Distance Measuring Sensors — have a range from 4 cm to 30 cm and are mounted forward with an azimuth of 15°. Due to their position on the robot, the lower bound cannot be reached. Their purpose is collision avoidance and ball detection. To support the localization process, the long range IR-sensors — Sharp GP2Y0A02YK — Long Distance Measuring Sensors — are mounted atop the turnable head providing them with relatively clear view to the borders. They have a range from 20 cm to 150 cm. All IR-sensors are connected using the I^2C bus and have an update frequency of up to 10 kHz.

6.2.3 Vision

The vision system is divided into two parts: (1) a fast and simple mono-vision for ball detection integrated into the body of the robot [MON05], and (2) the turnable stereo-vision head for complex object detection [BAS+06, Bad07]. Both systems produce as output highly abstracted symbols of detected objects and landmarks. These symbols consist of direction and distance towards the object and the type of the object. For some landmarks, only the direction and the type are available. The distance is then not available. This accounts for partly observed landmarks or for objects which are too small to calculate a precise depth information.

Mono-Vision

This system is optimized to recognize a circular shaped object in a predefined color. Thus, edge detection (Figure 6.3(a)) can be reduced to a so called "short line detection" or SLD

[1]The wheels are designed equally. The diameter differences are manufacturing imprecisions.

where only lines with a pixel length of 20 to 50 are considered. Furthermore, only the left and the upper neighbor pixel are used reducing the usually needed nine multiplications and nine additions to two comparisons per pixel. Parallel to edge detection, the color blob detection (Figure 6.3(b)) is executed. Next, the found short lines close to the detected blob are combined during object detection. The resulting algorithm outputs the detected ball position on the picture. Using a priori knowledge — the diameter of the ball and the hight of the camera above the ground — the position of the ball on the playground can be calculated.

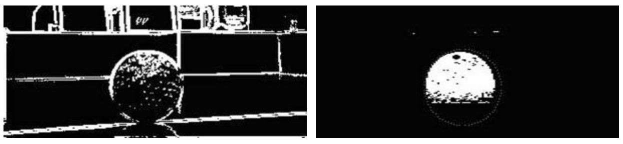

(a) Edge Detection (b) Blob Detection

Figure 6.3. Mono Vision Ball Detection Process [MON05]

The camera used provides a frame rate of up to 60 Hz, a resolution of 320x200 and an opening angle of 55 degrees. As a controller, an Analog Devices BF533 Blackfin DSP with 600 MHz is used.

The mono-vision system is fast — up to 60 frames per second — and accurate in detection of the presence and the direction to the ball. Its downside is its lack of accuracy in the distance to the ball and its restriction to only circular or ball shaped objects.

Stereo-Vision

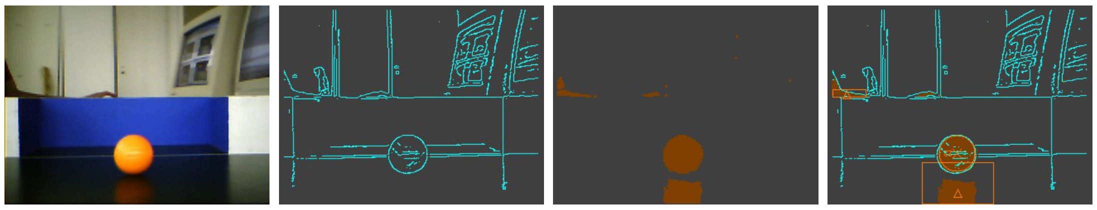
The stereo-vision system, which is mounted on a turnable head is designed to recognize a broad range of objects. These objects are defined using primitive 3D objects like ball and box. Parameters are size, height above floor, and color. As an output, this system produces so called "sight-vectors" — 3D polar coordinates (α,β,r) — of actually observed objects and a quality of the sighting[2]. The quality — which ranges from 0 to 1 — is a combined representation of the accuracy that the primitive object has been detected, the accuracy that the color is matching, and the correctness of the resulting position.

The detection process is divided into four steps. First, each of the two cameras captures an image (Figure 6.4(a)) and does a mono-vision edge detection (Figure 6.4(b)) and blob color segmentation (Figure 6.4(c)) for its own. The resulting two sets of edges are unified into a feature based depth map. Using the feature based depth map, a 3D line detection is carried out. Finally, the objects are detected with the 3D lines and the combined blob color information (Figure 6.4(d)).

Opposite to the mono-vision, the stereo-vision produces precise positions[3] of arbitrary

[2]Sighting is defined as "The act of catching sight of something, especially something unusual or searched for." In robotics it is used for the successfully recognized landmark through the vision system.

[3]The distance to partially observed objects may be — similar to mono-vision — wrong. This error is reduced by using depth information gained from the stereo edge detection.



(a) Captured Image (b) Edge Detection (c) Blob Detection (d) Combined Data

Figure 6.4. Stereo Vision Object Detection Process

objects, but it is relatively slow. The detection process is running at 5 Hz in full resolution of 320×200. To speed-up the process, an area of interest can be selected — e.g. 15 Hz for an area of 200×100 pixels. Thus, the performance is tightly bound to the area of interest. It should be mentioned that the number of predefined objects to be detected has no influence on the performance.

Mono and stereo vision share the same problems: illumination, reflection due to flat perspective[4], correct distance for partially observed objects, calibration of cameras, etc.

6.3 Software Architecture Used for Tinyphoon

The approach to the software architecture used for Tinyphoon is a scaled down CLARAty architecture with an additional unit — the world model repository (WMR) — attached to the Functional Layer and Decision — now called Reasoning — Layer (see Figure 6.5). Decision making is implemented as a rule based fuzzy logic mechanism utilizing XML to store the rules (see [ENW05] and [Web05]).

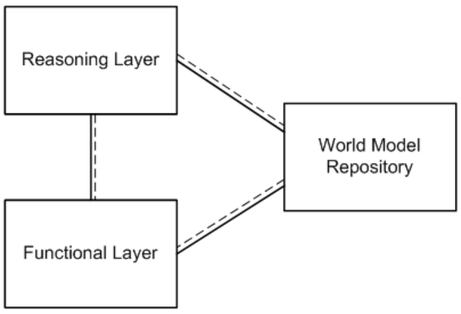


Figure 6.5. Software Architecture With World Model Repository

If the WMR is implemented as part of the Reasoning Layer — as suggested by the CLARAty-architecture — several problems arise. The two layer client-server approach collides with the requirement that the WMR can be utilized by the Reasoning Layer and the Functional Layer. Thus, the WMR acts as its own server with both layers as its clients.

[4]Figure 6.4 is a good example for surface reflection. The blob detection produces two ball objects. Not until the combination of 3D lines and the blob fields, the false positive ball can be eliminated — see boxed orange area at the bottom of Figure 6.4(d).

The Reasoning Layer contains the Decision Unit. The Functional Layer contains two units: the Vision Unit and the Motion Unit. The WMR is distributed to all of these three units with the main part residing at the Decision Unit. The main part of the WMR consists of the localization and the main data storage. The other two units are provided with the data they need for operation.

The distribution of the units across several processors leads to the necessity of another unit — the Communication Unit. It works as an abstraction layer to the communication hardware described in Section 6.5. In combination with WMR, communication provides the functionality needed for a shared world model.

6.4 Five Step Pipeline

The basic control concept drafted in Chapter 3 is parallelized in Tinyphoon using a five step pipeline (Figure 6.6). At the beginning of each step, the newly generated data of the last step is distributed among the units. The maximum time allowed to finish one step is 20 ms. Thus, each component — even computationally expensive parts like decision making — has to finish its processing within this time frame. If a component like the stereo vision cannot fulfill this tight time corset, the algorithms have to be adapted towards the in that case optionally available data.

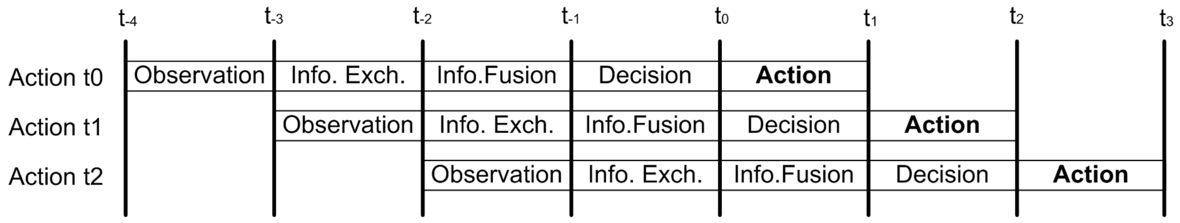


Figure 6.6. Five Step Pipeline.

Step 1 — Raw sensor data collection and preparation The Motion Unit (see Section 6.1) calculates the relative movement (Δx, Δy, and Δα) using its odometric sensors. These three values can be used to determine the local self localization.

The Vision Unit (see Section 6.1) detects the Ball using color matching, meanwhile the stereo image is processed with an edge detection algorithm. Based upon the detected edges other robots are extracted. This may be supported with information about the position of other robots provided by the WMR (see Chapter 4). As a third task, the Vision Unit calculates the global self localization.

Step 2 — Information exchange with other robots (optional) Each robot sends its newly observed and prepared sensor data to its teammates.

Step 3 — Update WMR The WMR integrates all new data (own and shared) into its database and estimates the new position of every object — including currently not observed objects — using e.g. Kalman filter or particle filter.

Step 4 — Strategy selection and distribution The Decision Unit calculates its next action using a two step schema: first, a role — striker, defense, goal keeper, etc. — is selected, second, the best action is chosen. This action command is transferred to the Motion Unit.

Step 5 — Action Execution The selected action is executed using predefined trajectories.

Note that the time-span between fetching the sensor information and executing the responding action is four cycles. Thus, a delay of 80 ms between observation and reaction has to be included into the calculations. For stereo vision based information, this gap is even worse. A frame rate of 15 Hz leads to an additional delay of 40 ms, 120 ms in total. If the delay is not considered, even under perfect localization the possible distance to the true position is up to 28 cm (respectively 42 cm) for a maximum speed of 3.2 m/s.

6.5 Joint Communication Architecture

Communication in Tinyphoon has several duties. It has to transmit data from one unit to another inside the robot, and it has to exchange data with other robots. Thus, communication in Tinyphoon is split up into two different parts:

Internal Communication The task of the internal communication is to ensure the data exchange between the units (Figure 6.7). All three hardware units of Tinyphoon (Vision Unit, Decision Unit, and Motion Unit) are connected via a daisy-chain wire line. The WMR resides at the Decision Unit and is masked by the internal communication. Thus, it appears to all other units as an independent hardware. The internal communication is responsible for passing over packages until the receiver has been reached and to encapsulate logical units.

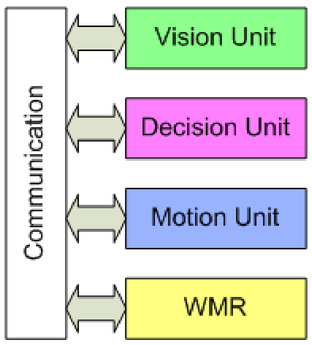


Figure 6.7. Communication Between the Units of One Robot.

External Communication External communication is needed for team coordination and inter-robot data exchange purposes. The task of the external communication is to send and transmit packages wirelessly to the team mates (Figure 6.8).

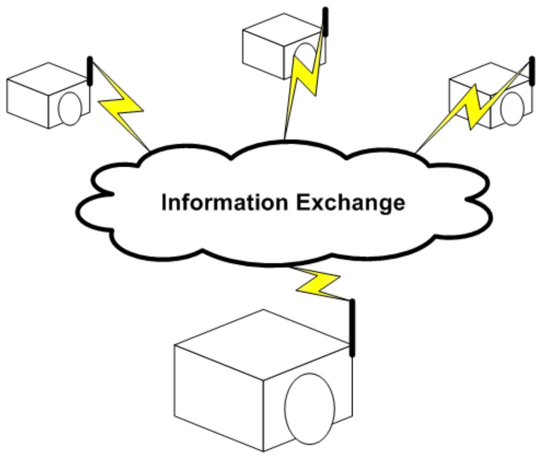


Figure 6.8. Wireless Communication Between Robots.

As already shown in Chapter 3, the internal communication and the external communication are tightly coupled for localization purposes. Internal and external communication are working on different media (wireless and wire line) and are in principle asynchronous. In Tinyphoon, this gap is covered by the implemented joint-communication architecture. This architecture guarantees that internal and external communication are running synchronously and are based upon the same schedule. Thus, for the software system, they appear as one communication channel.

The concept used for the communication is the one from TTP/A [EI03]. This means that the communication happens time triggered, time is synchronized implicitly through communication, and the values are transferred via an Internal File System (IFS). It is capsulated in such a way that TTP/A can be exchanged easily by another communication protocol (e.g. CAN). Because TTP/A provides no error correction, error correction has to be integrated into the communication system. Each value has one distinct sender and one or more receivers. Values, which are not updated in every round (e.g. global self position), need an updated flag. This enables the system to distinguish between old data and data which has been updated but has the same value as before. A more detailed explanation of the TTP/A implementation for Tinyphoon can be found in [Kry06].

6.6 Hardware Tools

For communication with a host PC, a wireless USB dongle (Figure 6.9) is used. It enables a PC to establish a radio communication with the Tinyphoon robots. This dongle provides 16 channels at a frequency of 2.4 GHz. The maximum payload per message is 28 bytes. It is integrated into the system with a transparent UART mode — the application is not aware that the serial connection is interrupted. Two modes of operation are available: master and slave. The master can send either point to point to a slave or broadcast a message to all slaves. A slave can only communicate with the master. Thus, point to point communication between slaves has to be implemented using the master as a relay station.

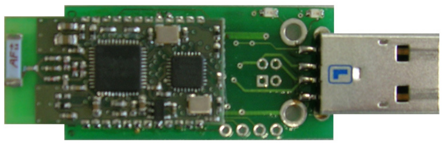


Figure 6.9. Wireless USB Dongle.

6.7 Software Tools

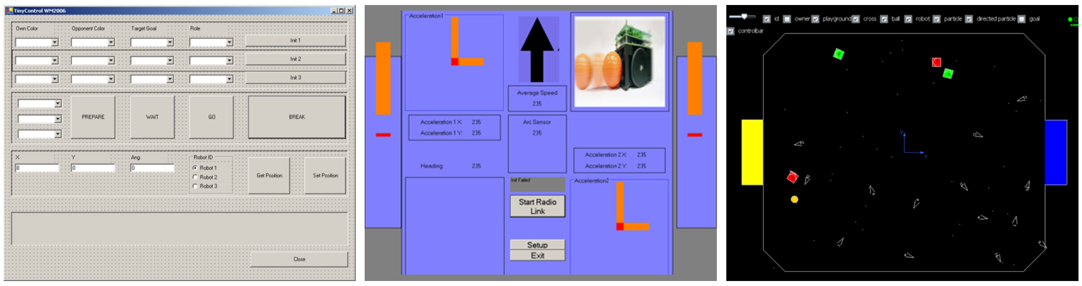
To control Tinyphoon remotely and to collect debugging information, three tools have been developed. Each one focuses on a sub task. For some tasks like displaying vision, debugging data tools are under development by the Tinyphoon team. It is planned to include all these tools into one framework.

TinyControl has been developed to control the robots in a game of AMiroSOT. Each robot can be applied with a strategic role (defender, goalie, or striker). During interruptions the robots are informed about the referee decisions. These can be free ball for one team, penalty kick, and others. All robots can be started and stopped with the large buttons in the middle of the form. For configuration purposes, each robot can be assigned to a position on the playground. Debugging information is displayed at the large field below the form (see Figure 6.10(a)).

TinyTelemetry focuses on the motion unit of one robot. Thus, all sensor information, which can be collected from the motion unit, is displayed here. This includes the acceleration sensors, the odometry, the wheel encoders, and the compass. Furthermore, motion commands can be directly entered using this tool (see Figure 6.10(b)).

TinyVisualizer is able to display all sensor information, the world model repository, and data from the localization process. This is a powerful tool to debug the localization or the strategy during game play. Even the particles of the particle filter (see Section 5.6) can be displayed, resulting in large amount of data, which has to be additionally transmitted via radio communication (see Figure 6.10(c)).

RFModuleGUI can send and receive data as transmitted by the radio transmission module.



(a) TinyControl (b) TinyTelemetry (c) TinyVisualizer

Figure 6.10. Software Tools for Tinyphoon.

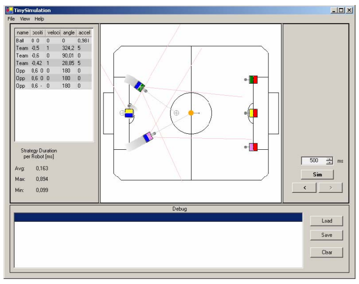
6.8 Simulators

To decouple the development of the hardware and the software, simulators were introduced into the Tinyphoon project. With the advance of the Tinyphoon project, the requirements for the simulators rose. First, only a platform for testing the decision was needed, and TinySimulator was introduced. With the requirement to be able to test the strategy against other teams, a wrapper for TinySimuroSOT was needed. After the split-up of the software into several modules — TinyReasoning, TinyWorldModelRepository, trajectories for the Motion Unit and new projects using the Tinyphoon as a hardware platform — the design of the new simulator MPSim started.

TinySimulation is a simulator (see Figure 6.11(a)), which has been developed solely for testing of the TinyReasoning module. It provides no communication between the robots, and only one team is controlled via the module. The other team can be directed to certain points to test basic strategic setups.

TinySimuroSOT (Figure 6.11(b)) is a wrapper class to be able to use TinyReasoning with the official FIRA simulator SimuroSOT (implemented with Macromedia Director 3D). While in TinySimulation the restricted vision is simulated (only objects, which are within the vision cone are sensed), TinySimuroSOT provides all robot positions to the TinyReasoning module. This is done because SimuroSOT uses the MiroSOT rules where a centralized camera is placed above the playground, and each robot receives a complete world view.

MPSim — Multi Purpose Simulator — is the successor of TinySimulator and still under development by the Tinyphoon team. Other than TinySimulator, MPSim simulates a robot/real world interface. Hence, MPSim provides the sensors and actuators of the robot. Only information about objects within range and the speed of the wheels are exchanged between the robot implementation and the simulator. This enables the simulator to generate a more realistic environment using a physics engine. The rules of the game are not hard coded like in the other two simulators. They can be adopted to many different applications like e.g. automotive applications, BubbleFamilyGame[5], billiard, and, of course, soccer.



name	positi	veloci	angle	accel
Ball	0 0	0	0	0,981
Team	-0,5	1	324,2	5
Team	-0,6	0	90,01	0
Team	-0,42	1	28,85	5
Opp	0,6 0	0	180	0
Opp	0,6 0	0	180	0
Opp	0,6 -	0	180	0

Strategy Duration per Robot [ms]

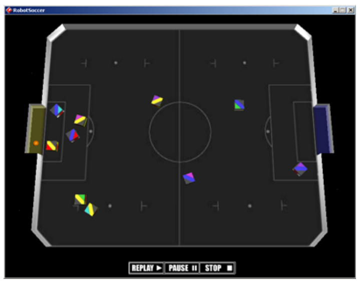
Avg: 0,163

Max: 0,894

Min: 0,099

500 ms

(a) TinySimulation



(b) TinySimuroSOT

Figure 6.11. Simulators for Tinyphoon.

6.9 Summary

Tinyphoon is a tiny robot specially designed for full autonomous activity. It is based upon a solid two-wheeled motion unit which enables it to reach velocities of up to 3.5 m/s at an acceleration of up to 5 m/s^2. For navigation, a broad range of sensors is applied, especially the turnable stereo-vision head for object detection should be mentioned. The Decision Unit is equipped with a fast processor capable of combining the collected sensor data and calculating strategic decisions. Furthermore, the robot can communicate wireless with other Tinyphoon robots, thus, enabling the development of robot swarms.

[5]The ARS-PA project of ICT as a long-term goal plans to implement reasoning strategies into the Tinyphoon robots based upon Sigmund Freuds psycho analysis (refer [DLP$^+$06]). For this reason, a game has been defined where populations (families) of creatures — so called Bubbles — have to deal with restricted resources like energy (the BubbleFamilyGame). `http://ars.ict.tuwien.ac.at`

7

Software System Description

Thoughts without content are empty, intuitions without concepts are blind.

— Immanuel Kant, Critique of Pure Reason

Beware of bugs in the above code; I have only proved it correct, not tried it.

— Donald E. Knuth, (1977)

Figure 7.1(a) shows a typical situation in robot soccer. Each robot observes only a small part of the playground. Possible contents of their field of view are one of the two goals, team mates, opponents, and the ball. The goals can be used for localization purposes. They are the only landmarks available. The other contents have to be stored in the World Model Repository (WMR). When one of them is currently not visible — which is usually the case — the current position has to be estimated based upon the stored observations.

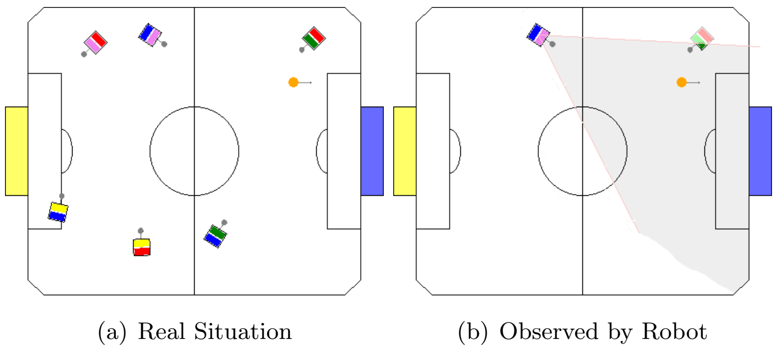

(a) Real Situation (b) Observed by Robot

Figure 7.1. Standard Situation in Robot Soccer

As depicted in Figure 7.1(b), the robot of interest observes its own blue goal, the ball, and parts of another robot. Thus — next to odometry and distance sensor information — one landmark is available for self-localization. The partly observable robot can be identified as an opponent. To identify which one of the three opponents it is, position estimations for all of them are calculated. The observed robot is then assigned to the observation history with the closest estimated position. Due to the fact that there is only one ball in the game, the ball observation can easily be added to the ball observation history in the WMR.

The self-localization integrates sensor data from different sources. Odometry, compass, and distance information is available in each cycle. The resulting position estimation is relatively precise (inside a circle with a diameter of 10 centimeters around the real position). The game field is symmetrical around two axis. It only differs in the coloring of the goals. Thus, the estimated position can also be diagonally opposed. To be able to solve this ambiguity, vision information is needed. The period between capturing an image containing a landmark and the output of the relative position ranges from 20 to 100 milliseconds. This relative position has to be unbiased by the estimated traveled track during this delay. Resulting, the vision data is approximately at the same quality as the combination of the other three sensors, but without it, a correct global localization would not be possible.

Concluding, the WMR has to contain at least a model of the playground (static data) and the position history of all robots and the ball (dynamic data). The self-localization has to be able to integrate the information of all four types of sensor (odometry, compass, distance sensor, and vision) into an estimated position.

This chapter presents a software system. It is the implementation of such a geometric WMR in combination with a self-localization module. As described earlier in this thesis, the Tinyphoon robot has only limited resources. Dependent on the application, the software system has to meet different requirements on complexity, speed, precision, and resource utilization. These may range from cases where a simple world model with a fast and simple but unprecise self-localization system is sufficient up to where a complex world model in combination with a precise self-localization system is necessary. To meet this, the software system consists of two different implementations for the WMR and the self-localization module. The first one — simple implementation — aims at fast algorithms which utilize little resources. The second one — complex implementation — focuses on better self-localization results and a more complex modeling of the environment. Both implementations share the same high-level system design.

In the first section of this chapter, the high-level system design is defined (Section 7.1). Next, the two different approaches to WMR are described (Section 7.2). Afterwards, the self-localization systems and the used sensor models are described in Section 7.3. Finally, the whole software system implementation is described in Section 7.4.

7.1 High-Level System Design

Three actors which use the WMR and the self-localization can be identified: Motion Unit, Vision Unit, and Decision Unit. These units are identical to the units defined in Chapter 6. The tasks of these actors are:

Motion Unit — provides sensor readings of the odometry, the compass, and the infrared sensors. This actor requests no data from the WMR. The only interaction between Motion Unit and WMR is that the Motion Unit stores data into the WMR.

Vision Unit — provides landmark sighting vectors and information about detected objects. Requests information about the static world (e.g. shape of the playground) and the current self position from the WMR.

Decision Unit — needs information about the static world, dynamic objects in this world, and the current self position for decision making. No information is stored into the WMR by this actor.

Figure 7.2 shows the use cases in which these actors participate in. The main system of interest is the WMR. The localization system is a sub-system of WMR (see Chapter 4). The WMR is the system where all sensor data from different units is stored. The localization system combines all available sensor data into a position estimate. If the localization system is placed elsewhere, this data has to be transferred from the WMR. Thus, the design decision that the localization is a part of the WMR reduces the communication overhead. The five identified use cases are:

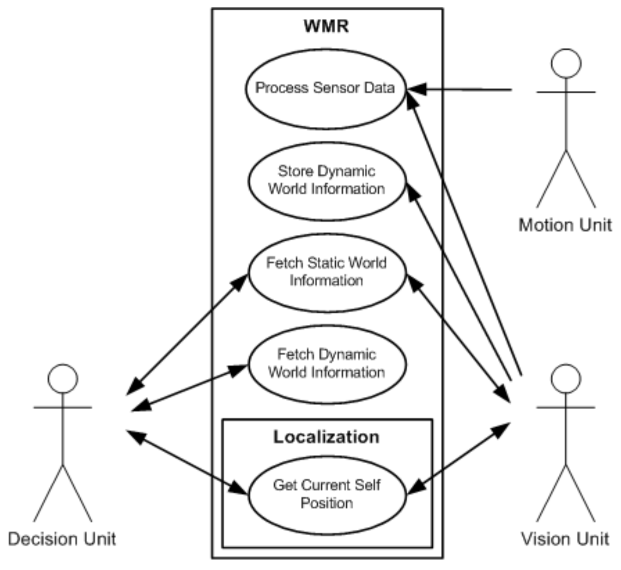

WMR
Process Sensor Data
Store Dynamic World Information
Fetch Static World Information
Fetch Dynamic World Information
Localization
Get Current Self Position
Motion Unit
Decision Unit
Vision Unit

Figure 7.2. Use Case Diagram

Process Sensor Data Raw sensor data information has to be fed into the WMR. This data can be odometric data, compass readings, infrared sensor distance readings, or landmark sighting vectors. The actors Motion Unit and Vision Unit participate in this use case.

Store Dynamic World Information The vision system is able to detect dynamic objects like robots or the ball directly. This data has only to be stored by the WMR. No further processing is needed. The Vision Unit participates in this use case.

Fetch Static World Information Static information about the world (e.g. shape and position of landmarks, number of robots per team) is needed for two tasks: detection of landmarks through the vision system and for decision making. Thus, Vision Unit and Decision Unit participate in this use case.

Fetch Dynamic World Information For strategic assessments of a situation information about the position of all dynamic objects is needed. The Decision Unit participates in this use case.

Get Current Self Position Similarly to the dynamic world information, the current self position is important for situation assessment. Furthermore, this information helps in identifying landmarks more precisely. The two actors Decision Unit and Vision Unit participate in this use case.

The objects of the problem domain in the real world are shown in Figure 7.3. The depicted object-oriented analysis (OOA) diagram represents the structural part of a requirements model. It defines the problem which has to be solved by the software [Kai99]. The domain of robot soccer contains a `Ball-A`, the `Playground-A`, two goals (`Goal-A`), and soccer robots.

The attribute *Color* of the class `Ball-A` denotes of course the color of the ball. According to the rules, an orange ball is used.

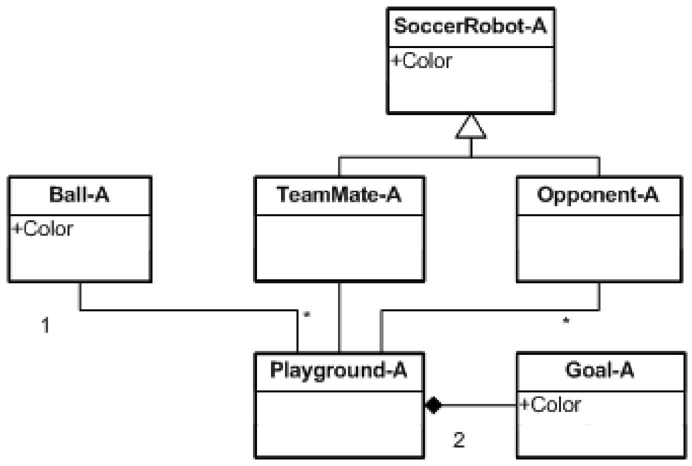


Figure 7.3. OOA Class Diagram of the Objects of the Problem Domain

The soccer robots are grouped into the team mates and the opponents. The object `TeamMate-A` represents every soccer robot of the robots own team — the robot itself is also a member of the group of team mates. Each soccer robot of the opposing team is represented by an `Opponent-A` object. While the number of balls in a game of AMiroSOT is limited to one, the number of soccer robots per team may vary from one to up to eleven soccer robots. During training and testing, one team may be completely absent. This varying number of soccer robots is defined by the '*' for the associations between `Playground-A` and `TeamMate-A` and `Opponent-A` in Figure 7.3. Thus, the design allows zero or more soccer robots per team.

The classes `TeamMate` and `Opponent` can be generalized to a `SoccerRobot` class. Thereafter, common attributes are factored out to `SoccerRobot`. The attribute *Color* defines the team color of a soccer robot. Possible team colors are red and green.

The two goals are areas inside the playground with a special meaning. Each team aims at pushing the ball into the goal of the opponent to score. Further, the goals can be distinguished from each other. The attribute *Color* of the class `Goal` models this distinction. One goal is blue, the other yellow. Thus, they can be used for localization purposes. In AMiroSOT, the borders are tall solid walls which isolate the soccer robots from the outside world. Thus, all objects of interest reside inside the playground. Hence, the ball and the soccer robots are always inside the playground.

As stated above, the simple and the complex implementation share the same high-level system design. Thus, before showing their design in detail, this high-level design is described, see Figure 7.4. According to [Kai99], "an OOD[1] model is a model of the proposed software system's internal construction." It defines the solution to the above given problem by describing the objects of the software system.

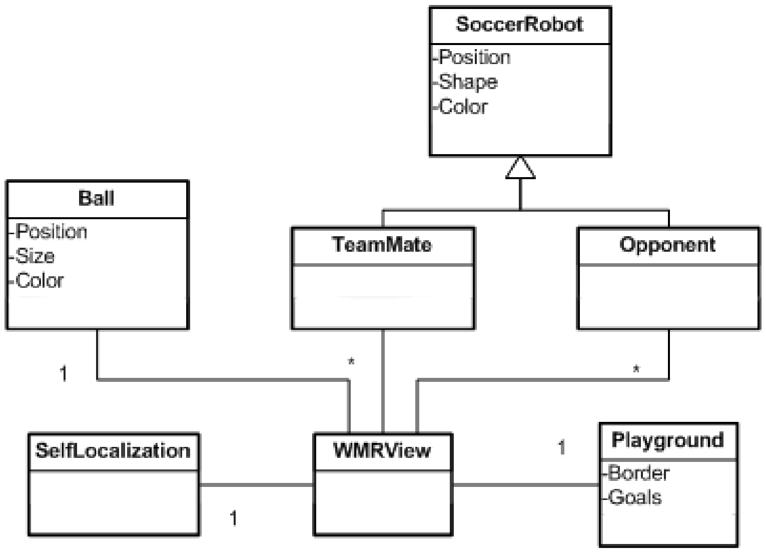


Figure 7.4. High-Level OOD Class Diagram of the WMR

[1]OOD — Object-oriented design

The dynamic entity team mate — `TeamMate-A` in the OOA model — which is represented in the software system by the class `TeamMate`. Similarly, the opponent robot is represented by the class `Opponent` (`Opponent-A` in the OOA model). As in the OOA model, `TeamMate` and `Opponent` are generalized to the class `SoccerRobot` and is the software systems' representation of the class `SoccerRobot-A`. It has the attributes *Position*, *Shape*, and *Color*. The attribute *Color* defines the team color and is the representation of the attribute *Color* in the OOA model. *Position* denotes the location and heading of the robot on the playground. The attribute *Shape* describes of course the shape of the robot.

The class `Ball` is the representation of the ball (`Ball-A` in the OOA model). It differs from `TeamMate` and `Opponent` in the attribute *Size* instead of *Shape*. A ball in robot soccer is always a sphere. Thus, only the radius of this sphere has to be given.

The playground `Playground-A` is in the software system represented by the class `Playground`. The two goals from the OOA model (`Goal-A`) are now attributes of `Playground`. An additional attribute is *Border*. This attribute enables the class to determine if a position is inside or outside the playground. The design decision that the goals and the borders are parts of the playground consequently results in the fact that every static world information is modeled into the class `Playground`. Thus, an additional class has to be added which stores the instances of the dynamic entity classes `Ball`, `TeamMate`, and `Opponent`. This is done by the introduction of the class `WMRView`. As described in Chapter 4, a WMR stores information about the static environment and the dynamic entities. Thereafter, `WMRView` not only stores instances of the dynamic entity classes, furthermore it stores an instance of the class `Playground`. For the realization of the use case "get current self position" a system capable of transforming sensor readings into position estimations has to be added — the class `SelfLocalization` (see Chapter 5). As described above, the localization is a part of the WMR.

The sequence diagram in Figure 7.5 shows the interaction between the three actors and the instantiated objects from the above described OOD. Further, it shows the interaction between external actors and the software systems' internal objects and the time-dependence between all use cases but one. The use case "fetch static world information" occurs during system configuration only. The interaction takes place during one operation cycle — from receiving the odometric data to providing all relevant information to the Decision Unit.

The first five sequences initiated by the Motion Unit and the Vision Unit are in no particular order. The raw sensor readings from the odometry and the sighting vector of the detected landmark are stored into the localization for further processing. The positions of the detected ball and the two robots are passed to their respective object to be stored in the position history. All these sequences are represented in the use case diagram by the two use cases: process sensor data and store dynamic world information.

The sequences initiated by the Decision Unit are the two use cases: fetch dynamic world information and get current self position. First, the Decision Unit requests information about the current self position. This information is provided by the localization which calculates

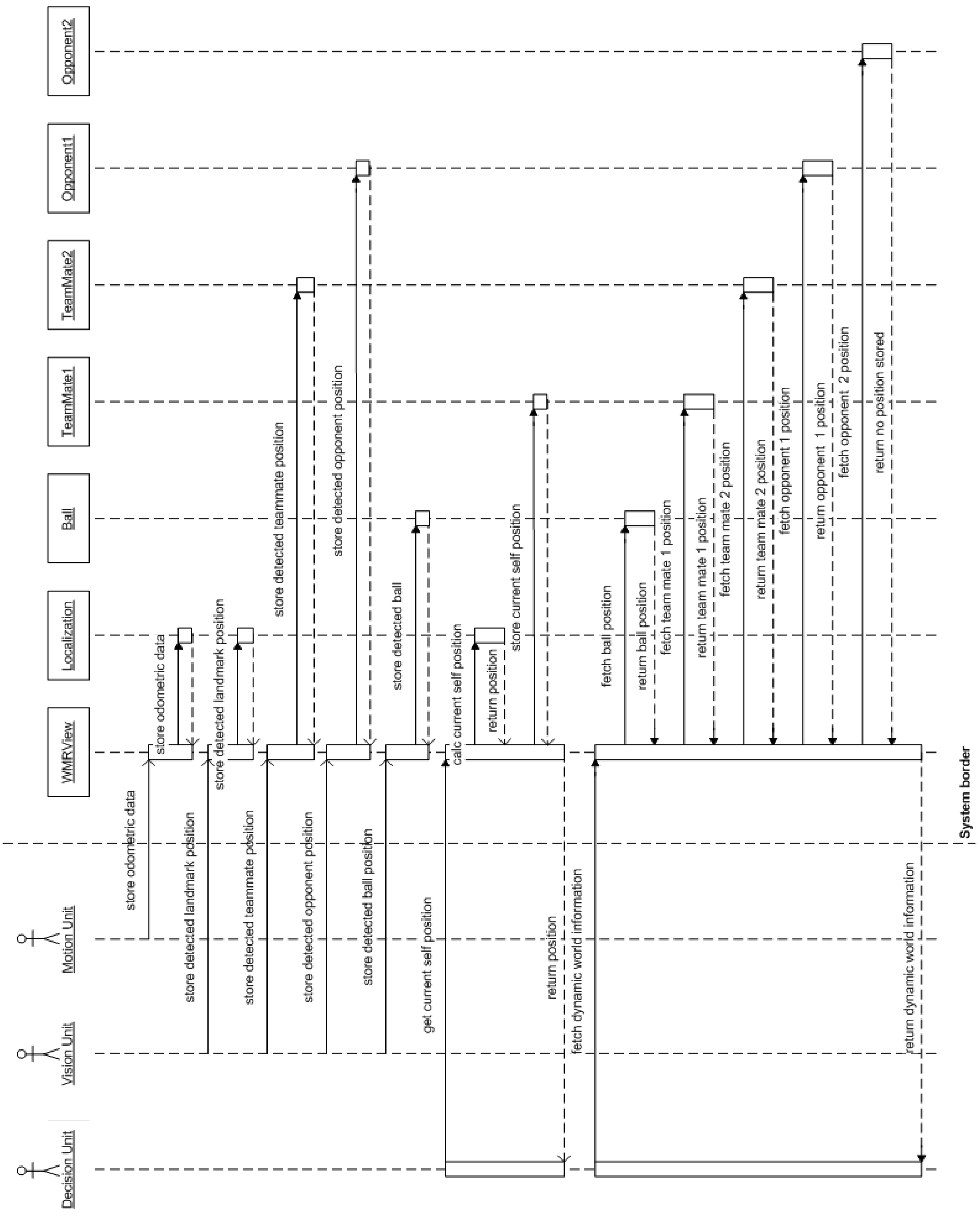

Opponent2
Opponent1
TeamMate2
TeamMate1
Ball
Localization
WMRView
Motion Unit
Vision Unit
Decision Unit
System border
store odometric data
store odometric data
store detected landmark position
store detected landmark position
store detected teammate position
store detected teammate position
store detected opponent position
store detected opponent position
store detected ball position
store detected ball
get current self position
calc current self position
return position
store current self position
return position
fetch dynamic world information
fetch ball position
return ball position
fetch team mate 1 position
return team mate 1 position
fetch team mate 2 position
return team mate 2 position
fetch opponent 1 position
return opponent 1 position
fetch opponent 2 position
return no position stored
return dynamic world information

Figure 7.5. Sequence Diagram

it using the latest sensor information. The self position is stored to the history of the first team mate. Team mate one represents the robot itself. Finally the self position is returned to the Decision Unit. Next, information about all dynamic objects is requested. Thus, the ball position history and the position histories of all team mates and opponents are polled. All but one real world objects represented by their respective position history have already been sighted. For the ones with at least one entry in the history, the last known position is returned. For the second opponent, which has not been observed yet, 'no position stored' is returned.

Concluding, the sequence diagram shows that the WMRView is a unit which decouples the inner system from the outer clients/actors. All information is passed to or received from the WMRView. The inner classes like `Ball` can be accessed only through the WMRView. Such a design is a structural design pattern called façade [GHJV95, Chapter 4].

7.2 Geometric World Model Repository

As mentioned above, two different approaches for a geometric world model repository (WMR) are described here. The simple approach is designed to store only the history of the positions and the playground. The complex approach additionally stores the arbitrary shapes of the robots. In case of the Tinyphoon robot, this is a square. This enables a more precise check if an estimated position is inside the borders of the playground. In the case the robot is close to the border, the heading of the orientation of its shape gives additional information on the plausibility of this position. A shape overlapping the border indicates a weak estimation.

The WMR is used — next to the above described features — as a hardware abstraction layer. Every sensor reading is stored into the WMR for exchange purposes. Values of infrared sensors, odometric sensors, vision readings, etc. are put into the WMR.

7.2.1 Simple WMR

This scaled-down WMR provides only a position history and static environmental information. The position history contains two types of information: the positions based upon observations and positions interpolated from previous entries. The entries are sorted by timestamp. If the maximum queue length has been reached, the oldest entries are deleted first.

All dynamic entities (team mates, opponents, and the ball) are treated as points. Thus, overlapping between dynamic entities cannot be recognized. The information if a position is inside the playground is reduced to the boolean values true and false.

The static environmental information consists of the shape of the playground and the landmarks. A playground has six landmarks (three for each goal). Information about the shape and the color of the dynamic entities is not stored in this simple WMR. This data has to be provided to the actors by an a-priori static configuration. For example, the Vision Unit has to read the geometric information about dynamic entities from a config-file.

7.2.2 Complex WMR

This WMR is designed as a geometric map built out of polygons (Figure 7.6). This allows verification of the possibility of positions. The intersection area between two dynamic entities or between an dynamic entity and the borders of the playground can be calculated and put into proportion of the total area of the dynamic entity. Thus, small overlapping may be accepted due to the knowledge about imprecisions in localization. Larger overlapping indicates that the localized position(s) may be useless. Other than with geometric primitives — where the calculation of the intersection area between two distinct primitive types is always a special case and has to be specified — polygon intersection is always calculated identically. To speed up this process, each polygon has an outer bounding box. These boxes are aligned to the x and y axis. Most of the time, a test against the bounding boxes is sufficient to determine exactly if an overlapping or an intersection between two polygons occurred. Only if the test result is ambiguous, the time-consuming polygon intersection test has to be executed.

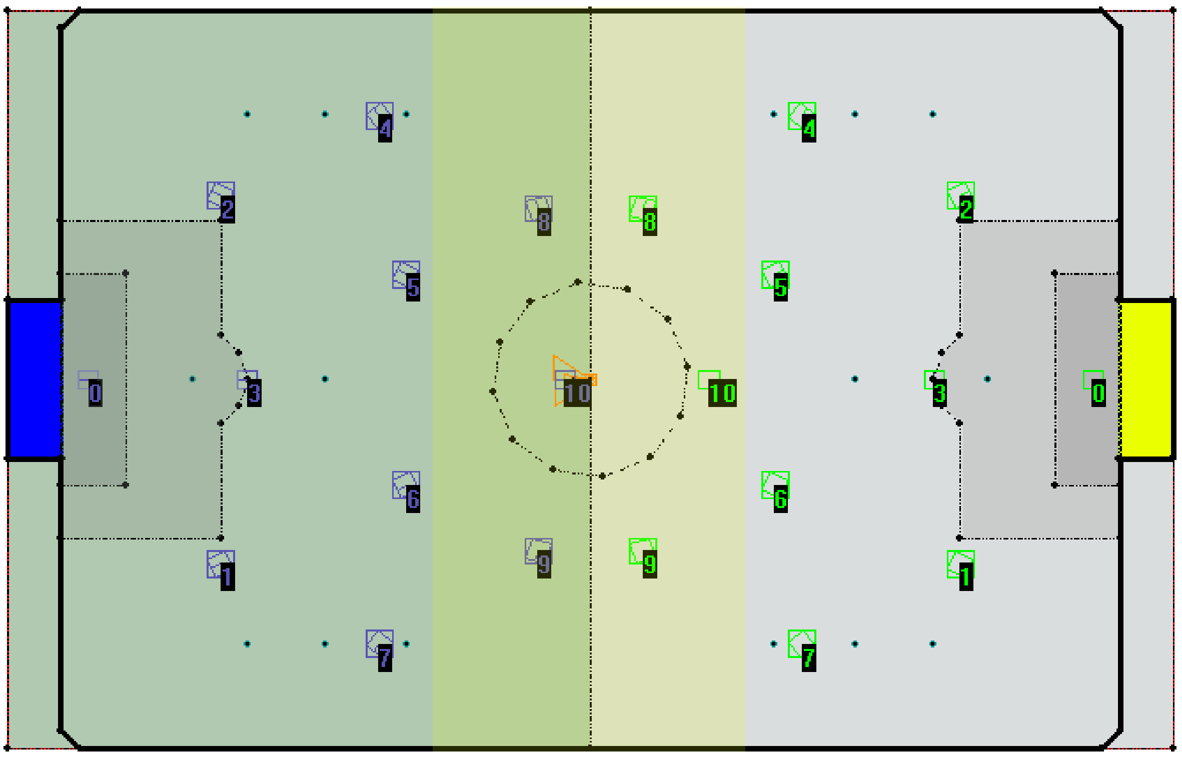


Figure 7.6. Geometric Map

Each dynamic entity — ball, team mate, or opponent — has a position predictor attached to it. Other than localization — which estimates the current position based upon sensor readings — prediction is a look into the future based upon the estimated position history.

There are different types of position predictors: linear, non-linear, and knowledge-based. For the ball, a linear predictor is sufficient — the ball moves straight ahead at a decreasing velocity. Only if an external force affects on the ball, it changes its course. In the case of a border, this impact can be forecasted and the predicted path can be adjusted. The case of an impact with a robot is highly dynamic (when will it happen) and difficult to predict (what is the exact position and orientation of the robot). Thus, in this case, the linear predictor is only informed that an impact happened, and that the history of ball observations has to be cleared. Robots have a non-linear movement. Precise long-term predictions are not possible. The Kalman filter can be used for short term predictions. As described in Section 5.4.4, this filter can be configured to estimate non-linear movement. For long-term predictions, knowledge-based approaches may be used. The future actions of a robot are predicted, based upon a history analysis and potential threats.

The ball has a tail attached to it. The area described by this tail is used to answer if a robot is behind the ball. This knowledge is important when trying to shoot the ball into a certain direction.

Additionally points of interest can be added. These may be positions on the playground (e.g. penalty kick position) or of landmarks detectable by the vision system. The stored landmarks are the same as for the simple WMR. The points of interest are drawn as dots in Figure 7.6.

The concept of areas of interest provides a grouping mechanism for the dynamic entities. One dynamic entity can be in several overlapping areas. Only the center point of a dynamic entity is of interest when matching it against an area. In principle, these areas are strategically relevant sectors like the two goal areas, each half, and the middle of the game field. A robot close to the center but in its own half is in the middle area and the own half area. An area can contain several dynamic entities. This makes querying for all dynamic entities e.g. in the middle area fast and easy. These areas are not necessarily static. They can also be attached to dynamic entities. This enables to create an area which contains all robots close to the ball. In Figure 7.6 these areas are differently colored. The dashed lines are the ground markings.

7.3 Self-Localization

Similarly to the WMR, the self-localization system contains two approaches — a simple one, and a complex one based upon the Particle filter as described in Section 5.6. Both return an estimated position based upon the current sensor readings. The first approach can only process information which leads directly to another absolute position (e.g. odometric data, sightings of landmarks). The particle filter can process any information which can be collected by the Tinyphoon robot.

7.3.1 Simple Self-Localization

The simple self-localization approach is an implementation of the LLSQ (Section 5.3). In each round, a new absolute position is calculated and fed into the LLSQ. The output is a smoothed path.

The basic concept of this approach is similar to the localization used for Tinyphoon up to now. The main enhancement introduced in this section is the post-procession of the data through the LLSQ.

To estimate a position, the following simple sensor models are used:

Odometric Model The relative movement vector returned from the motion unit is directly added to the current position.

$$\begin{pmatrix} x \\ y \\ \varphi \end{pmatrix} = \begin{pmatrix} x \\ y \\ \varphi \end{pmatrix}_{curr} + \begin{pmatrix} \Delta x \\ \Delta y \\ \Delta \varphi \end{pmatrix}_{odo} \tag{7.1}$$

The innovation perceived from the odometry $(\Delta x\ \Delta y\ \Delta\varphi)^T_{curr}$ is added to the current position denoted by the vector $(x\ y\ \varphi)^T_{curr}$ (Equation (7.1)). This results in the new position $(x\ y\ \varphi)^T$.

Compass Model The compass is used to set the heading of the robot. Due to the bad quality of the compass this value cannot be used directly. Two consecutive sensor readings may have a difference of up to 120°. Thus, the difference between the actual sensor reading and the current estimation heading is used only partially. The estimated direction converges towards the real direction over time.

$$\varphi = \varphi_{curr} + \frac{\varphi_{compass} - \varphi_{curr}}{\ell} \tag{7.2}$$

The current heading of the robot φ_{curr} is subtracted from the value returned by the compass $\varphi_{compass}$ (Equation (7.2)). The result is divided by the trust factor ℓ. The influence of the compass readings is reciprocal to this factor. Finally, the new heading φ is determined by addition of this innovation to the current heading φ_{curr}.

Vision Model The vision model provides two types of information:

- The heading of the robot. To estimate the direction of the robot, the angle between the estimated position and the sighted landmark is calculated. If more than one landmark is visible, the average angle is used. This heading information overrides the compass model. The reason why to keep the compass model is the update frequency. New compass readings are available every localization round, whereas landmark sightings occur at random points in time. Thus, the compass model fills the direction information gap between two consecutive landmark sightings.

- The location of the robot (only x and y axis). The current position of the robot can be calculated using triangulation when the center landmark of a goal and at least one of the pole landmarks are visible. In Figure 7.7 the available data when perceiving the center and one pole of a goal is marked red.

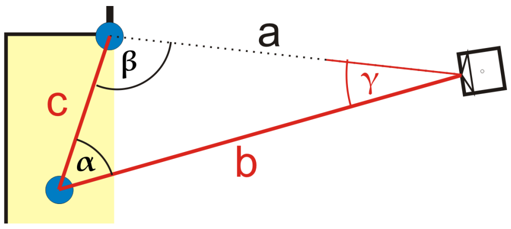


Figure 7.7. Position Triangulation

$$\beta = \arcsin\left(\frac{b \sin\gamma}{c}\right) \tag{7.3}$$

Applying the law of sines in Equation (7.3), the value for the angle β can be calculated. Subsequently, also the third side a and the last angle α can be gained through the law of sines. With a fully defined triangle and two known positions (the landmarks), the position of the robot can be determined.

The availability of only two sides and one not enclosed angle results in an ambiguity [Wik06b]. β can have two values. This ambiguity is bypassed by choosing the β which results in a robot position closer to the previous one. When dealing with all three landmarks (center and both poles), this ambiguity does not appear.

7.3.2 Complex Self-Localization

This self-localization approach is based upon the Particle filter. It uses particle re-injection and re-sampling (see Section 5.6, especially Algorithm 5.5). In the case of a sensor reading with an error outside the acceptable range, the resulting path would have a peak. This peak vanishes in the next localization cycle. To level such peaks, the estimated positions are smoothed using an LLSQ.

Differently to the sensor models used for the simple localization, the following models return the belief for one particle. This belief reflects if the robot is located at the position represented by the particle according to the current sensor readings. The only exception to this is the odometric model. It returns an absolute position.

Distance Measurement Model Using a model of the map, the distance between the particle and the wall in the line of sight of the distance sensor can be calculated. A Gaussian noise is applied to this distance. Using a normalizer, the calculated and the measured distance are put into proportion resulting in the belief. If the values are below the minimum range or above the maximum range of the sensor, the belief is set to the maximum.

Additionally, three fault cases are added to this model:

- The presence of an unexpected object (e.g. robot) between the sensor and the wall.
- The sensor returns the maximum distance, although the next wall is within the range.
- A random value is measured due to disturbances like reflections.

A detailed explanation of this model can be found in [FT06, pages 153–158].

Compass Model The belief for the compass readings is determined using a Gaussian distribution. Figure 7.8 shows a set of particles after applying the compass model on them. The darker they are, the more they are directing into the same direction as the robot (denoted by the red line originating in the center of the robot.).

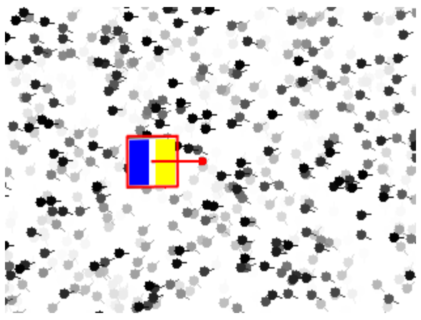


Figure 7.8. Model Compass

$$p(\varphi_{curr}|\varphi_{compass}) = (2\pi\sigma^2)^{-\frac{1}{2}} \exp\left\{-\frac{1}{2}\frac{(\varphi_{compass} - \varphi_{curr})^2}{\sigma^2}\right\} \quad (7.4)$$

The Gaussian function denoted in Equation 7.4 gives the probability that the heading of the current particle of interest φ_{curr} is correct under the condition that the compass sensor returned the sensor reading $\varphi_{compass}$. σ is the standard deviation of the values returned from the compass.

Map Model Every particle is tested for being inside the playground. To avoid binary evaluation results, the particle is tested against three convex hulls of the playground — each one 5% larger than the previous one. Such a hull is rectangularly shaped. The estimated position is a result of all particles. If the robot is located directly at the border and if every particle which is outside the playground would be evaluated to be for sure outside, the position estimated would be systematically biased towards the center of the playground.

To reduce the calculation time needed, this model does not use the precise distance a particle is outside the playground. As shown in Equation (7.5), the particles are divided into five groups: one inside the playground, one outside all convex hulls, one between the playground and the smallest hull, and the other two between the three hulls. The probability that a particle represented by its state vector x is a possible position estimation is determined by assigning it to one of the five groups.

$$p(x) = \begin{cases} 1 & \text{... if particle is inside the playground} \\ 0.95 & \text{... if particle is between the playground and the first hull} \\ 0.9 & \text{... if particle is between the first and the second hull} \\ 0.75 & \text{... if particle is between the second and the third hull} \\ 0.5 & \text{... otherwise} \end{cases} \tag{7.5}$$

As described above, positions close to the border would not be possible if an particle that is outside the playground is marked with a probability of 0. To prevent this, even particles that are far outside are assigned at least a probability of 0.5. The partitioning and values shown in Equation (7.5) are only examples. The optimal values for a given application have to be determined by optimization experiments.

Vision Model The vision system returns two different types of sightings. The first one returns only the direction to the landmark. The second one additionally returns the distance towards it. Thus, they have to be treated differently.

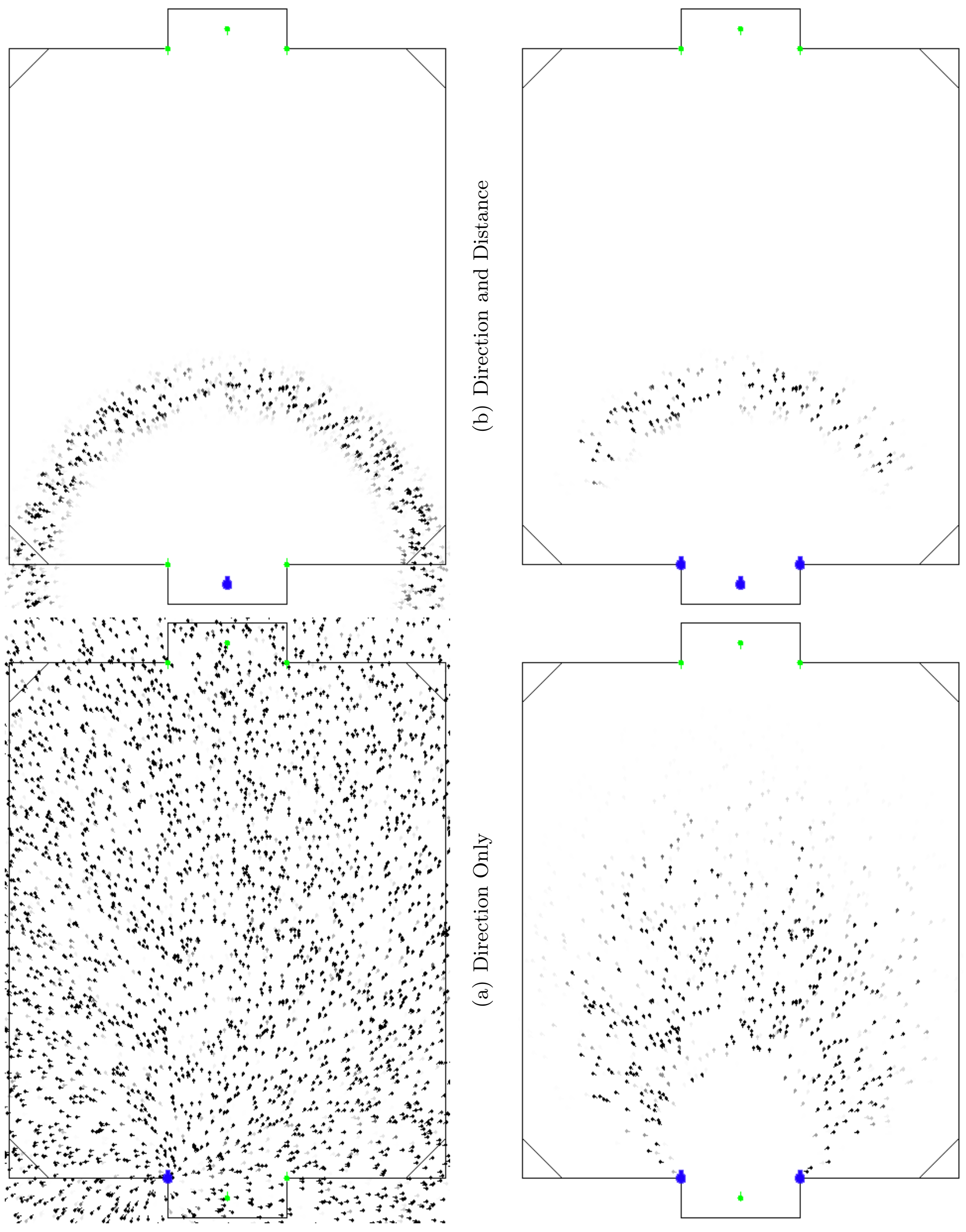
Figure 7.9(a) shows the outcome of the direction-only-model. All particles which are targeting towards the upper landmark of the left goal (marked blue), are assigned with a high belief. Similarly to the compass model, a Gaussian distribution is used to add system noise. Due to the fact that no distance information is available, the particles are regarding their x and y axis still distributed uniformly.

If, in addition, the distance is known, the marked particles are reduced to a ring around the middle landmark of the left goal (marked blue) in Figure 7.9(b). The width is dependent on the system noise added to the distance.

When combining several landmark sightings, only the overlapping particles are marked. Figure 7.9(c) shows the case where the upper and the lower landmark of the right goal are visible. Both landmarks are processed by the direction-only-model. The resulting area of possible positions covers a large area due to the lack of a distance information.

In Figure 7.9(d), the vision system perceives information about all three landmarks of the left goal. Two of these sightings are the goals poles. For these, only direction information is returned. Thus, is is a combination of the two cases depicted in the figures 7.9(c) and 7.9(b)

A detailed explanation of this model can be found in [FT06, pages 176–180].



(a) Direction Only

(b) Direction and Distance

(c) Combination of Two Direction Only Landmarks

(d) Combination of Three Landmarks

Figure 7.9. Model Vision

Odometric Model The relative movement as measured by the odometric sensor is decomposed into three values (Figure 7.10):

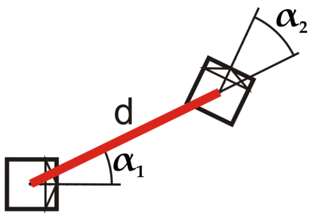


Figure 7.10. Model Odometry

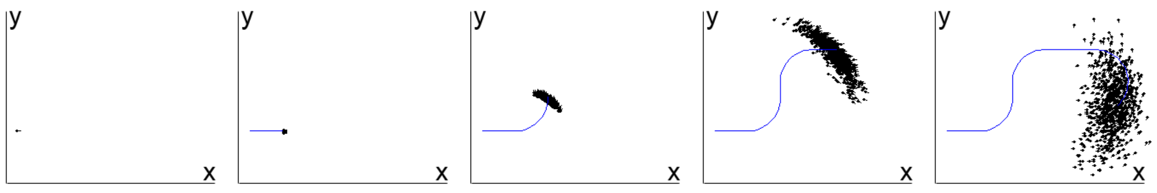
α_1 defines the angle needed to turn the robot towards the target point.

d gives the distance between the starting and the final position.

α_2 defines the angle needed to turn the robot towards the final direction.

The true path that the robot traveled to reach the target point, cannot be measured and is, thus, not of interest. Each of these values is attached a Gaussian noise to model the imprecisions of the odometric sensor (e.g. drift, slipping). Finally, the new position of the particle is calculated using these blurred measurements [FT06, pages 132–139].

Figure 7.11 shows a localization based solely on this motion model. In the beginning, the exact position is known and all particles are at the same position (Figure 7.11(a)). Movement straight ahead is modeled with only small noise. This can be seen in the second step (Figure 7.11(b)). The particles have only slightly different positions. The situation changes after the left turn (Figure 7.11(c)). Due to the assumption that during a curve drift occurs, all particles have spread to the left and the right. After taking another turn, the average distance of the particles to the real position is large (Figure 7.11(d)). Without the support of another sensor model, the prediction is of too low quality. This estimation has to be enhanced by another sensor model. Finally, at step 5 (Figure 7.11(e)), the particles are distributed widely across the field. A trustworthy position estimation cannot be made upon this sparse particle cloud.



(a) Step 1 (b) Step 2 (c) Step 3 (d) Step 4 (e) Step 5

Figure 7.11. Odometric Model Example

7.4 Implementation

This section shows the OOD diagrams for the approach described above. For the WMR, two different diagrams are shown — one for the simple and one for the complex solution. The two self-localizations are shown in a single diagram.

7.4.1 Simple WMR

As described above, the WMR is designed with the façade design pattern as its template. The class representing the façade in the UML diagram depicted in Figure 7.12 is `SimpleWMRView`. In order not to clutter the diagram, the attributes and operations are not included.

The simple WMR is designed to be fast and small. Thereafter, the high-level design (Figure 7.4) is altered. The classes `Ball`, `TeamMate` and `Opponent` are replaced by a `PositionHistory`. The associations between the WMR and the dynamic and static entities of the environment are strengthened to compositions.

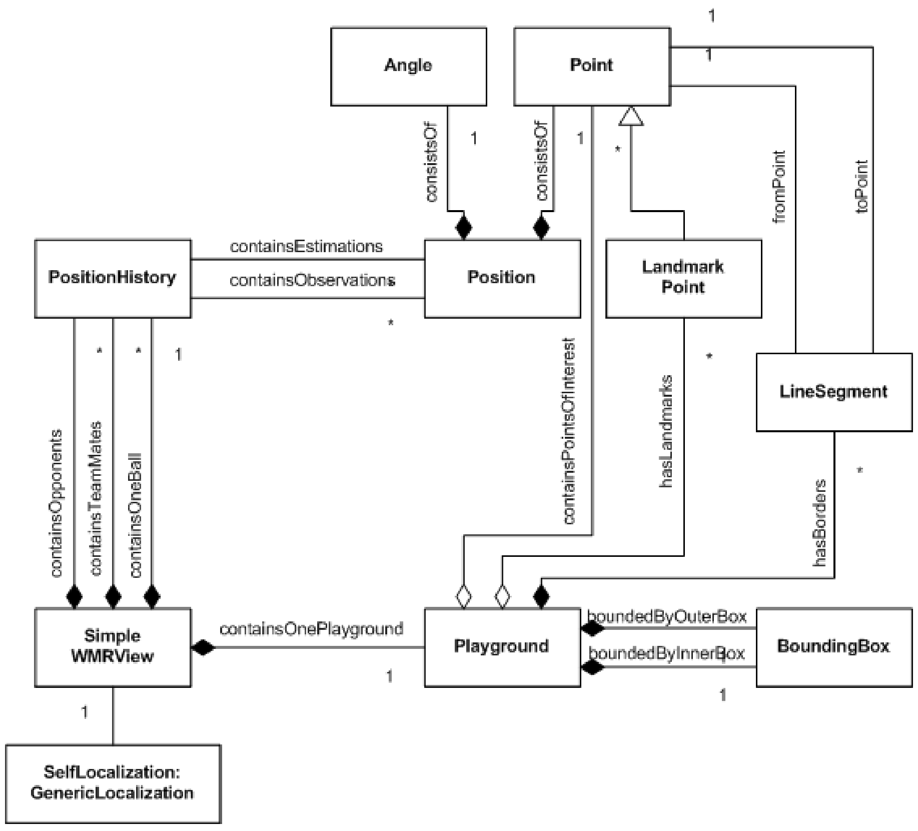


Figure 7.12. OOD Class Diagram for the Simple WMR.

When instantiated, the class `SimpleWMRView` adds for every dynamic entity on the playground an instance of the `PositionHistory` class. To distinguish between the ball, the team mates, and the opponents, the instances of `PositionHistory` for the robots are stored in two different queues and the instance of `PositionHistory` for the ball is stored as a member object of `SimpleWMRView`. The class `SimpleWMRView` expects an instance of the class `GenericLocalization` as parameter for its constructor.

The `PositionHistory` stores positions in two different queues. One queue is responsible for observed positions added to the WMR by e.g. the vision sensor. The other takes care of estimated positions generated by the localization. The positions are of type `Position`. This class is a composition of `Point` and `Angle`.

Next to the dynamic entities, the `SimpleWMRView` class contains an instance of the `Playground` class. This class composes border line segments, an inner bounding box, and an outer bounding box. Further, it aggregates a point for each landmark and points of interest. References to the instances of landmarks and points of interest may be passed to the external software system. Thus, it is unknown during destruction of the instance of `Playground` if these instance can be destroyed or are still needed. Thereafter, the associations between `Playground` and `Point` respectively `LandmarkPoint` are aggregations and not compositions. Additionally to providing static world information, the class `Playground` is able to test if a point is inside or outside the playground.

The borders of the playground are implemented by a set of instances of the class `LineSegment`. The start and the end point of a segment are associated with an instance of `Point` each. The class `BoundingBox` is the implementation of an axis align rectangle. It is optimized to determine, whether a point or another bounding box resides inside or outside the box. One instance of `BoundingBox` is configured to mark the maximum rectangle that fits into the borders. The other instance is the minimum possible rectangle outside the borders. The combination of the line segments and the bounding box speeds up the is-point-inside-playground test. Only if the point is inside the outer, but outside the inner bounding box, further tests with the line segments have to be performed.

The landmarks of the playground are of type `LandmarkPoint` which is a specialization of the `Point` class. It adds an Id to the point. This Id is used for the communication between the vision system and the localization. Thus, a vector pointing from the robot to a landmark can be easily attached to the true position of this landmark. The points of interest are instances of the class `Point`. They mark predefined points on the playground like the kick-off point.

Although the OOD class design depicted in this section (Figure 7.12) differs from the OOD class design in Figure 7.4, the sequence diagram shown above (Figure 7.5) is valid for it. The instance of `Ball` and the two instances of `TeamMate` and `Opponent` are substituted with the regarding instances of `PositionHistory`.

7.4.2 Complex WMR

Figure 7.13 shows the OOD class diagram for a complex WMR. In order not to clutter the diagram, the attributes and operations are not included. Other than for the simple WMR, the OOD model for the complex WMR refines the OOD class structure of the high-level design of the WMR described above.

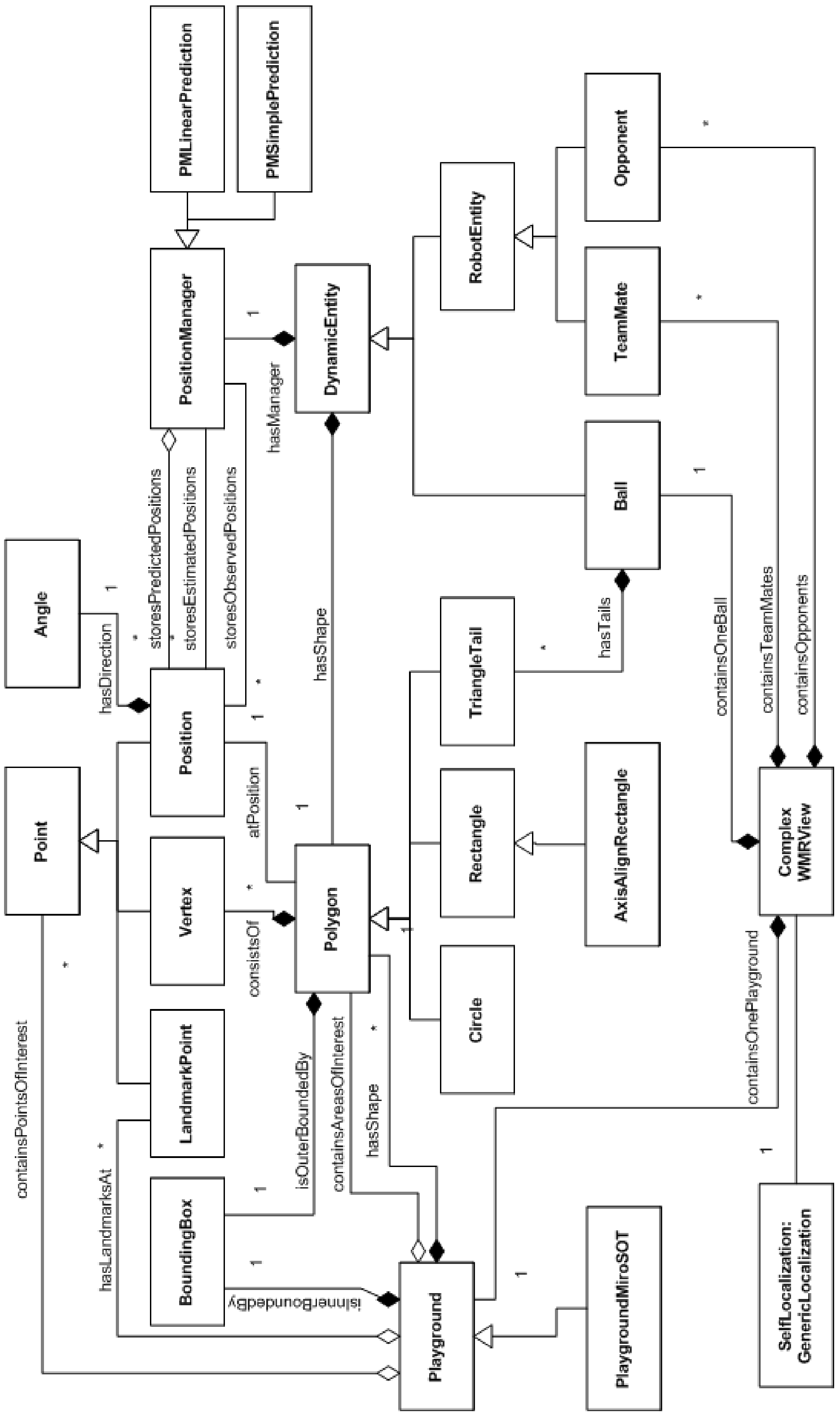


Figure 7.13. OOD Class Diagram for the Complex WMR.

The façade class `ComplexWMRView` is a composition of a playground, a ball, team mates, and opponents. While the simple WMR reduces the dynamic entities to area-less positions, the complex WMR treats them as objects with a two-dimensional shape.

The classes `Point`, `Angle`, `BoundingBox`, `LandmarkPoint`, and `GenericLocalization` are identical to the ones used for the simple WMR. `Position` is now a specialization of `Point` and uses an instance of `Angle` for the direction.

The class `Polygon` is used to formulate the shapes for the dynamic entities and the playground. It is a composition of vertices (the corners of the polygon) and an outer bounding box. The vertices are instances of the class `Vertex` which is a specialization of `Point` for polygon clipping purposes [GH98]. An instance of `BoundingBox` is used to define the minimum outer hull of the polygon. Only if two bounding boxes are overlapping, a time-consuming polygon clipping has to be executed. Using the polygon clipping, precise information about the intersecting area between two shapes can be given. A further element of a polygon is its center position. Initially, the center is set to the average of all vertices, but can be replaced to an arbitrary position. The center is used to move and rotate the shape without necessarily recalculating the new values for each vertex (which includes also the rotation of them). Only if a polygon intersection is performed, the updated vertices are needed.

Finally, three specializations of `Polygon` are implemented: `Circle`, `Rectangle`, and `TriangleTail`. The class `Circle` additionally stores the radius. Thus, the special case of the intersection of two circles can be optimized. `Rectangle` provides width and height for the rectangle. Its further specialization — `AxisAlignRectangle` — suppresses the rotation ability of the `Polygon` class. The four edges of the resulting polygon are always parallel to the axis. Thus, the outer bounding box of the polygon is also its maximum inner bounding box. Thus, before performing a polygon clipping, the other polygon is tested if it is not inside and not outside the bounding box. Finally, the class `TriangleTail` is the realization of an isosceles triangle. The center of the polygon is set to the vertex defining the corner which is embedded between the two equal edges.

Similarly to the simple WMR, the complex WMR stores in its `Playground` points of interest, one point for each landmark, and the maximum inner bounding box. The difference is that the shape of the playground is now defined as a polygon. Additionally to the aggregated points of interest and one point for each landmark, areas of interest are aggregated. These areas are of type `Polygon`. As described above, they are useful for defining strategically important regions and to test which dynamic entity is within this region. The specialization of `Playground` — `PlaygroundMiroSOT` — defines the shape and the points of interest according to the rules of MiroSOT. Similarly to the simple WMR, references to the instances of points of interest, landmark points, and areas of interest are passed to the external software system. During destruction of the instance of the class `Playground` these instances should not be destructed. Thereafter, the associations between `Playground` and `Point` respectively `LandmarkPoint` are aggregations and not compositions.

A `DynamicEntity` is a composition of a `PositionManager` and a `Polygon`. It represents the core of every dynamic entity in the environment. It can be placed to any position, the position manager stores past information (estimated positions) and can provide predictions of future positions.

The class `Ball` is a specialization of `DynamicEntity`. It defines that the polygon has to be of type `Circle`. Further, at least one instance of `TriangleTail` is attached to the ball. Triangle tails are used to test whether a dynamic entity is behind the ball or not. The definition of which direction is behind the ball can be set by the strategy. The default instance is automatically set to the "goal shooting position". For this, the ball has to be between the opponent goal and the robot. Thus, the tail is set to be on the opposite side of the ball regarding the opponent goal.

The classes `TeamMate` and `Opponent` are specializations of the class `RobotEntity`. This class adds robot-specific information like maximum speed, id, color. Currently the classes `TeamMate` and `Opponent` provide only the default shapes for the robots. The `TeamMate` instantiates a rectangle with the precise shape of the Tinyphoon robot. The shape of the opponents is set to the maximum allowed shape as defined by the rules. Additionally, these classes will be used for future extensions. When adding team communication, additional data like strategic decisions, have to be stored for the team mates. Similarly, data resulting from threat analysis of opponents may be added. This data may be the assumed role (striker, defender, etc.) or typical behaviors. Threat analysis and team communication is not in the scope of this work.

Other than the position history of the simple WMR, the class `PositionManager` can predict future positions of a dynamic entity. Thus, next to the associated queues of estimated positions and observed positions, the manager aggregates a queue of predicted positions. Every time a new position estimation or observation is received, the prediction queue is emptied. The class `PositionManager` itself provides only the interface but no implementation for a prediction. Instead it returns the last estimated positions. The specializations `PMLinearPrediction` and `PMSimplePrediction` have prediction algorithms implemented. The simple prediction calculates the average relative movement of the last n steps. The relative movement is changed according to the step period and the time to predict ahead. This value is added to the last position. The linear prediction is based upon the LLSQ (Section 5.3). Other than the simple predictor, this prediction is less sensitive to faulty position estimations but reacts later towards direction changes. References to the instances of the predicted positions are passed to the external software system. Thus, it is not known during destruction of the instance of `PositionManager` if these instances are needed elsewhere. Thereafter, this association between `PositionManager` and `Position` is an aggregation and not a composition.

The class `Polygon` provides several potential pitfalls during implementation. Polygon clipping for arbitrary shapes (e.g. [GH98]) is problematic in the case of overlapping lines or vertices. Also the test for "point inside polygon" (e.g. [Hec99]) may produce false results in special cases. For example, if a scan line algorithm with a vertical scan line is used, two vertices with exactly the same value for the x-axis as the point of interest are producing such a fault. Thus, dependent on the used algorithm, a point may be classified as outside, although it is inside.

The high-level WMR OOD model is refined to the complex WMR OOD model. The core of the new model is similar to the high-level model. Thus, the sequences of the process interactions of the new OOD model can be described by the sequence diagram depicted in Figure 7.5.

7.4.3 Self-Localization

This section describes the OOD class structure of the self-localization of the software system. The UML diagram (Figure 7.14) shows the specialization of the generic localization class (`GenericLocalization`) into the simple solution (`SimpleLocalization`) and the complex solution (`ComplexLocalization`). This abstract class provides the public methods for input and output of the data. The two concrete solutions provide no additional external access to their methods and the data. In order not to clutter the diagram, the attributes and operations are not included.

The `SimpleLocalization` class contains all its sensor models. To smoothen the estimated path, an instance of `LLSQ` is used.

`LLSQ` is implemented according to Algorithm 5.1. The data needed for the matrix is stored in an array of positions. The length of this array can be configured.

The complex solution is realized through a particle filter (or Monte Carlo Localization). The Algorithm 5.5 describes the basic functionality of the class `ComplexLocalization`. This class is a composition of particles and sensor models. Other than in the simple solution, the sensor models for the MCL are placed into their own classes. The base class for the sensor models is the abstract class `GenericSensorModel`. It provides functionality to mark the stored data as processed. The sensor models described in Section 7.3.2 are realized by the following specializations of the generic sensor model: `CompassModel`, `InfraredModel`, `MapModel`, `OdometryModel`, and `VisionBearingsOnlyModel`. The class `VisionFullModel` is a specialization of `VisionBearingsOnlyModel`. The class `VisionBearingsOnlyModel` can only process vectors to landmarks where the distance is unknown. This is extended by the class `VisionFullModel`. `ComplexLocalization` contains one model for the compass, the map, the odometric sensor, and the vision. For each distance sensor one instance of `InfraredModel` is used. This setup is designed to fit the needs of Tinyphoon. The particles generated by the complex self-localization are of type `Particle`. This is a specialization of `Position` and adds a weight attribute. The values for this attribute range from zero to max weight. See Section 5.6 for more detailed information about particles.

The classes `Angle`, `Point`, and `Position` are identical to the ones used in the simple WMR implementation.

The sequence diagram in Figure 7.15 shows interaction of the instantiated main objects of the simple self-localization for the use case “get current self position”. `SimpleLocalization`

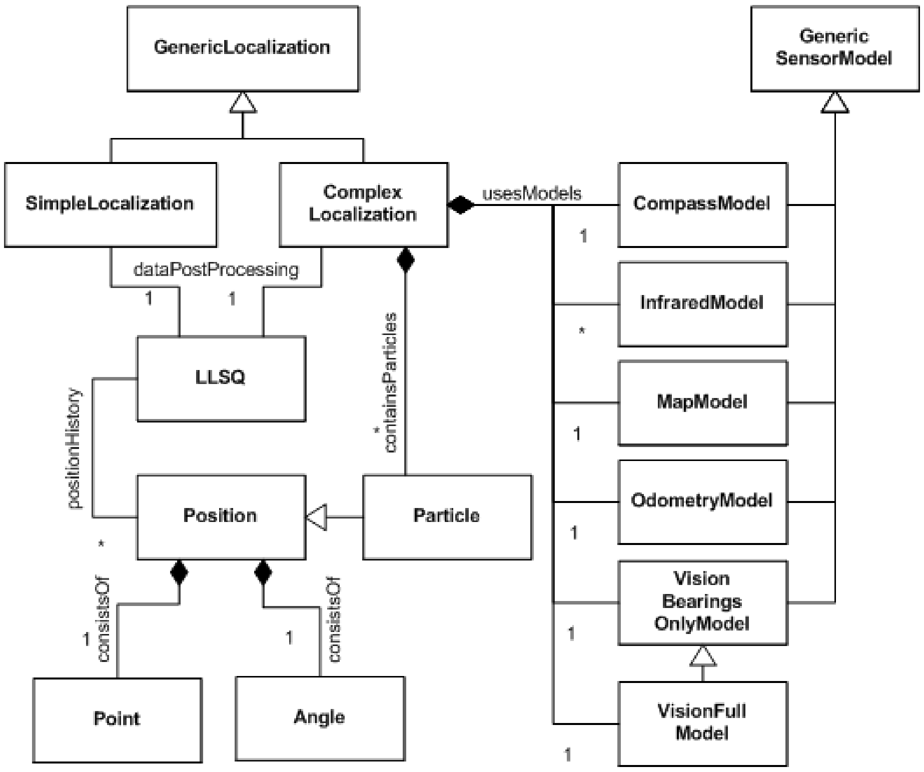


Figure 7.14. OOD Class Diagram for the Self-Localization.

receives the request for the current self position from `WMRView`. The sensor models described in Section 7.3.1 are implemented as operations of the class `SimpleLocalization`. The position estimation resulting from applying the current sensor readings to thee sensor models is for further processing passed to `LLSQ`. The `LLSQ` smoothes the position estimation (see Section 5.3). Finally, the position estimation is returned to `WMRView`. As described above, the localization is a sub-system of WMR. The dashed line in Figure 7.15 marks the border between the WMR and the localization system.

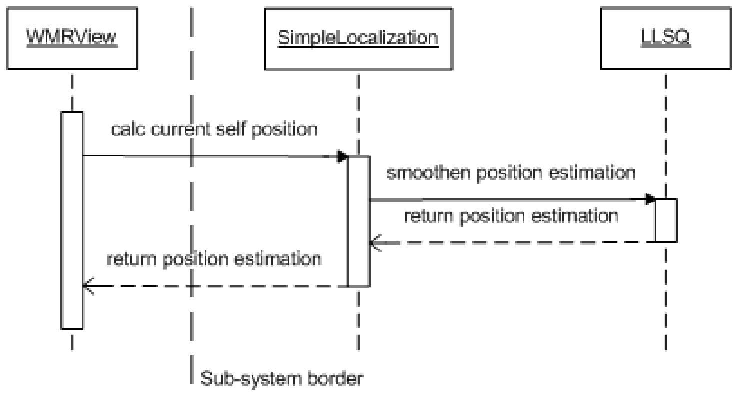

WMRView
SimpleLocalization
LLSQ
calc current self position
smoothen position estimation
return position estimation
return position estimation
Sub-system border

Figure 7.15. Sequence Diagram for the Simple Self-Localization.

Figure 7.16 depicts the sequence diagram for the complex self-localization for the use case "get current self position". Identically to the sequence of the simple self-localization, the sequence is initiated by `WMRView` with the request for the current self position.

Each particle instantiated by `ComplexLocalization` has to be processed by the sensor models described in Section 7.3.2. First a particle has to be reposition according to `OdometryModel`. Next, the probability of the particle is determined by applying the sensor readings to the sensor models `CompassModel`, `InfraredModel`, `MapModel`, `VisionFullModel`, and `VisionBearingsOnlyModel`. By integration of all position hypotheses — the particles weighted by their probability — a single position estimation is generated. This estimation is post-processed by `LLSQ` and returned to `WMRView`.

While no common pitfalls are known for the simple localization approach, the pitfalls known for the complex localization approach are numerous. The most important ones are:

- Due to the low quality of the current estimation, new uniformly distributed particles are injected. Particle injecting reduces the quality of the estimation. Thus, new particles are injected ...
- No particle fits to the vector from the robot to the perceived landmark. The weight of every particle converges against zero. This leads towards numeric problems. After normalization, the weight of each particle is indistinguishable from random values.
- A badly configured particle filter is almost indistinguishable from a faulty implementation. Thus, finding the right parameters for the sensor models is crucial. This has to be done by optimization experiments.
- Calculating the weighted average direction of the particles.
- The delay between the capturing of an image and the actual output of landmark sightings is unknown. The sightings cannot be unbiased by the relative movement made during this delay.

At the time of this writing, the infrared model determines the distance to the next wall by comparison of every wall segment with a line symbolizing the infrared beam. A playground for AMiroSOT consists of 16 wall segments. Thus, for each particle 16 line intersections have to be calculated in each cycle. To reduce the amount of calculations, all results are cached for further use. In [FT06] another optimization is suggested. Initially, an image representing the playground is generated. Using this image, ray-tracing algorithms can be used to calculate the distance.

7.5 Summary

To be able to store information about the world, the robot needs a WMR. For the ability to localize itself in this world, it needs a localization system. This solution shows two approaches for the WMR and the self-localization: a simple, fast one and a complex, powerful one.

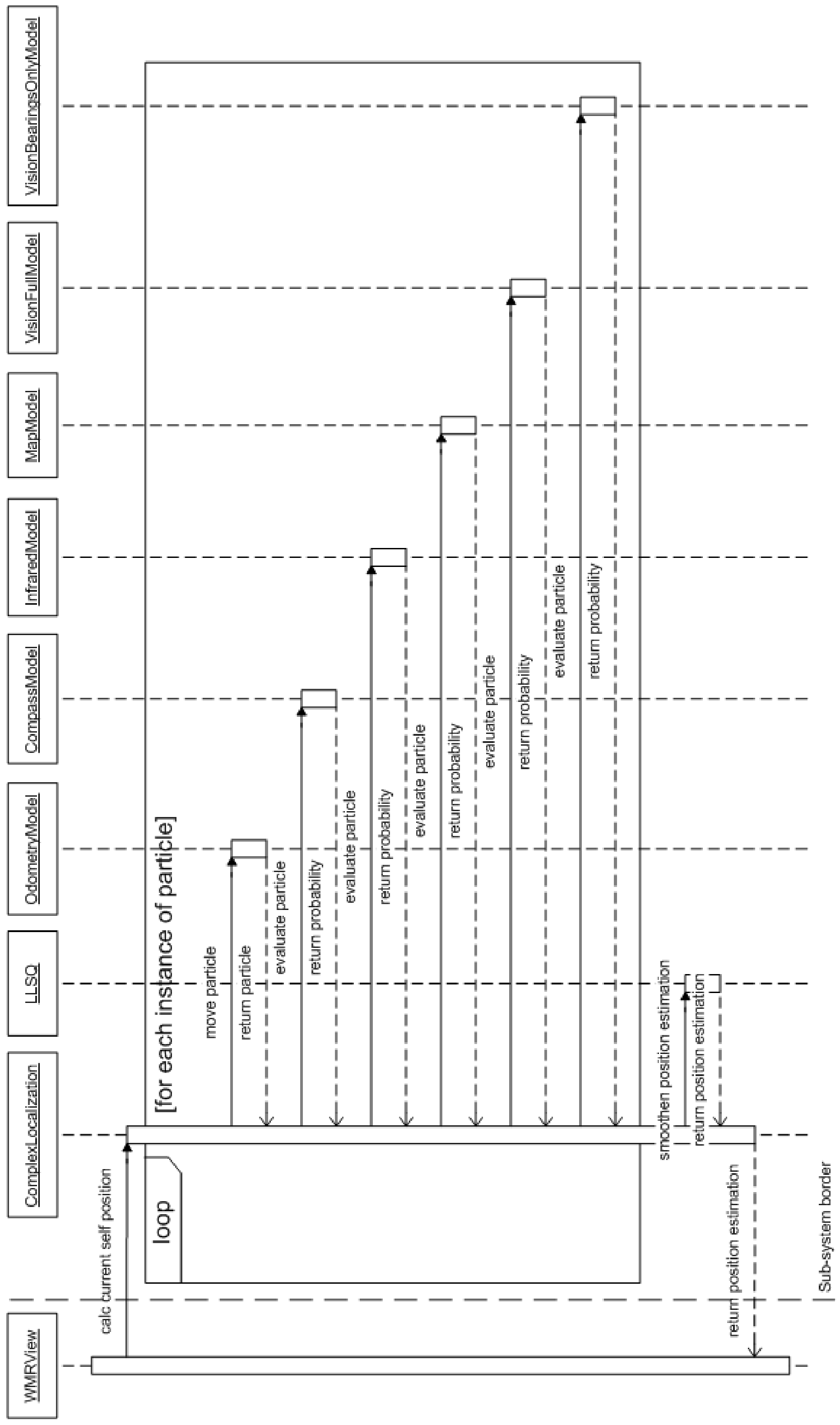


Figure 7.16. Sequence Diagram for the Complex Self-Localization.

The WMR for the simple approach consists of position histories for all dynamic entities (ball and robots), the borders of the playground, and three landmarks for each goal. Three types of sensors are used for the simple self-localization: the odometric sensor, the compass, and the vision system. The sensor models return absolute positions. These positions are combined into one absolute position. To smoothen the resulting estimation, this value is post-processed by using an LLSQ.

The complex WMR is realized by using a geometric map. Every dynamic entity (ball, robots, borders, areas of interest) is stored as a polygon. The advantage of polygons is that arbitrary shapes can be matched for overlapping. Dynamic entities have a position manager attached. This manager stores the history of positions and also provides the functionality of position prediction. The complex self-localization is based upon the particle filter and is able to process all types of sensors Tinyphoon is equipped with. The odometric model moves the particles, while the other sensor models calculate the believe that a particle represents the real robot position. To flatten peaks — which may occur through single faulty sensor readings — the resulting position estimation is also smoothed by using an LLSQ.

8 Implemented Self-Localization Systems in Comparison

Maj. Carter: Sir, the simulations we ran anticipated every conceivable scenario.
Col. O'Neill: You know, Carter, it's the inconceivable ones I'm concerned about.
—Stargate SG-1/Season 6/Redemption Part 1

To show the fitness of the complex self-localization approach, it is compared with the simple self-localization system (discussed in Chapter 7) and then with a self-localization method based on extended Kalman filter (EKF) implemented separately.

8.1 Comparison of the Simple and the Complex Self-Localization

This section compares the simple self-localization system with the complex self-localization system. Both are described in the previous chapter. This comparison was made in a simulated environment which consists of a robot moving along a predefined path, walls as obstacles for the infrared sensors, three landmarks for each goal, and an unknown obstacle.

8.1.1 Simulation Setup

The robot in the simulation was designed to be as similar to the Tinyphoon as possible. It is equipped with an odometric sensor, a compass, three long range infrared sensors (front, left, and right), and a stereo vision sensor. All these sensors are designed to fit the abilities and weaknesses of the real sensors as closely as possible: each sensor experiences system noise and suffers systematical error. The vision sensor takes longer to produce results than the other sensors. The transmission of the sensor data is grouped into two packets. The first one consists of the odometric data, the compass, and the infrared distance information. The second one groups all the landmarks recognized by the vision system. The packet with the odometric data is transmitted more often than the vision data. A jitter is applied to the delay between two consecutive transmissions. All parameters for the sensors and the transmission can be configured to match the real robot best.

Also the real data types are used for transmission. The data may be represented internally by fixed point types or even by floating point types. Thereafter, each value consumes up to four bytes of memory. For transmission purposes the internal precision has to be scaled down to fit smaller types like a two byte integer or even a byte. This down scaling is always a tradeoff between information quality and the number of packets per second. Nevertheless, it results in another decline of the quality of the sensor readings.

The path of the robot is cyclic. At the start, a 360° counter clockwise turn is executed. Then the robot moves towards the yellow goal. After a left-right combination, the robot changes the direction with a 180° right turn. After a 90° left turn, the robot stops close to the unknown obstacle. There is a further rotation of 360° in clockwise direction. Next, the robot turns towards the blue goal in a right curve. The loop is closed after a short track straight ahead with a 180° right turn.

This path covers all different types of movement (rotation on a position, straight and turn movement). From a sensor point of view it covers situations where the distance sensors are far away from the walls but also close to a corner. Further, the robot has to deal with track sections where no vision data is available or where three different landmarks are visible at the same time.

Figure 8.1 shows a snapshot of a typical simulation run. It consists of:

- The black border of the playground (MiroSOT middle league playground, 2.8 by 2.2 meters).
- Six landmarks — three for the yellow goal and three for the blue goal.
- An unmapped obstacle (green box); possibly an opponent robot.
- The sensor readings of the three infrared sensors (red rays).
- A sighted landmark. The upper yellow landmark is inside the vision cone (hinted by the two grey lines) and, thus, marked as visible.
- The dots mark the covered distance of the robot.
- The robot of interest in the center of the red rays.

This figure also shows some examples for the simulated sensor errors. The right infrared sensor is the only distance sensor returning a correct and precise result (maximum range reached). The left sensor returns a slightly too large, but still plausible value. The value

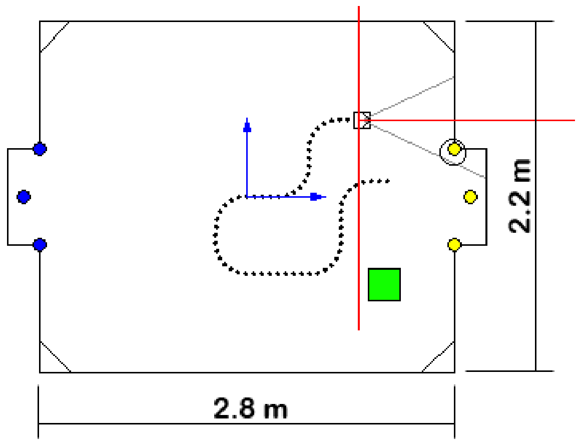


Figure 8.1. Typical Snapshot of a Simulation Run.

reported by the front sensor represents a complete failure. It returns that within the maximum sensor range no obstacle is present instead of the true value. The circle denoting the sighted landmark is placed slightly to the left. This results in a vector pointing from the robot to the landmark which is a little bit to short and points a few degrees too much to the right.

8.1.2 Results

Figure 8.3(a) depicts the real path traveled by the robot. The black dot in the upper left area marks the starting point and the first 360° turn. The second 360° turn is executed at the position marked by the other black dot in the lower right. From the beginning to the closing of the cycle, 237 odometric packets are transmitted. In 109 of these 237 steps, at least one landmark was recognized. These recognitions can be grouped into four distinct sightings of the left goal and six distinct sightings of the right one. During one distinct sighting, the goal was always visible.

The resulting estimated path for the simple localization approach is shown in Figure 8.3(b). In the beginning (green dot), the global position is unknown. Thus, the estimation starts at the position 0/0. After a view steps, vision data is available and the estimated position changes. The path jumps back and forth until it reaches the last estimation (red dot). The low quality of the shape has two sources: the bad odometric sensor readings in combination with a low number of analyzable sightings of a landmark. Only one third of the sightings can be used for the vision sensor model of the simple approach. At least two landmarks have to be visible for the simple approach (see Section 7.3.1). The standard deviation between the real and the estimated position is close to 15 centimeters (Figure 8.2). This precision is sufficient for a rough position estimation.

Figure 8.3(c) shows the estimated path by the complex self-localization which is based on the Monte Carlo localization (MCL) without the post-processing of the LLSQ. Also in this case, the starting position is unknown (green dot). The ability to process all recognized landmarks results in continuous approximation towards the real position during the 360°

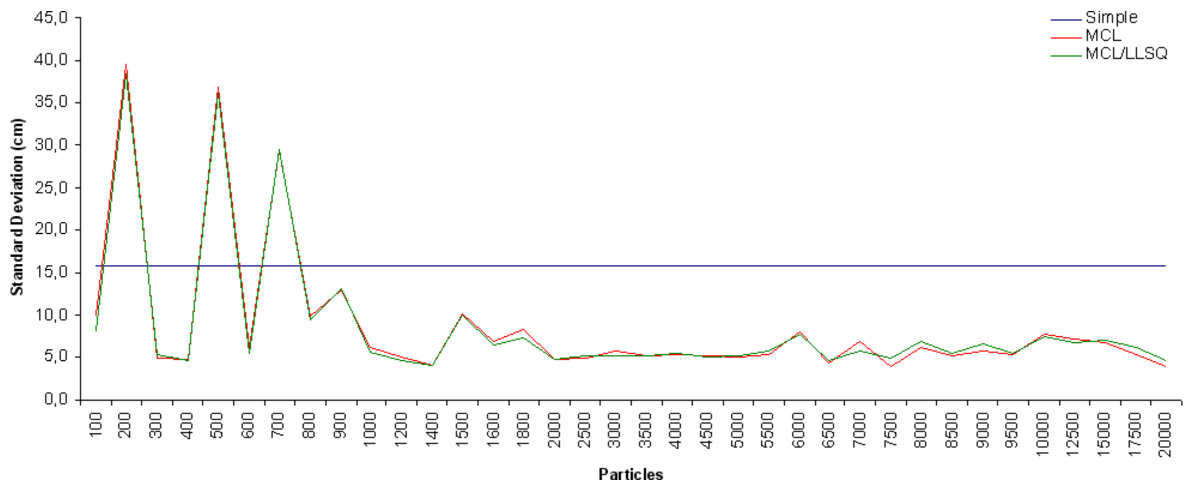


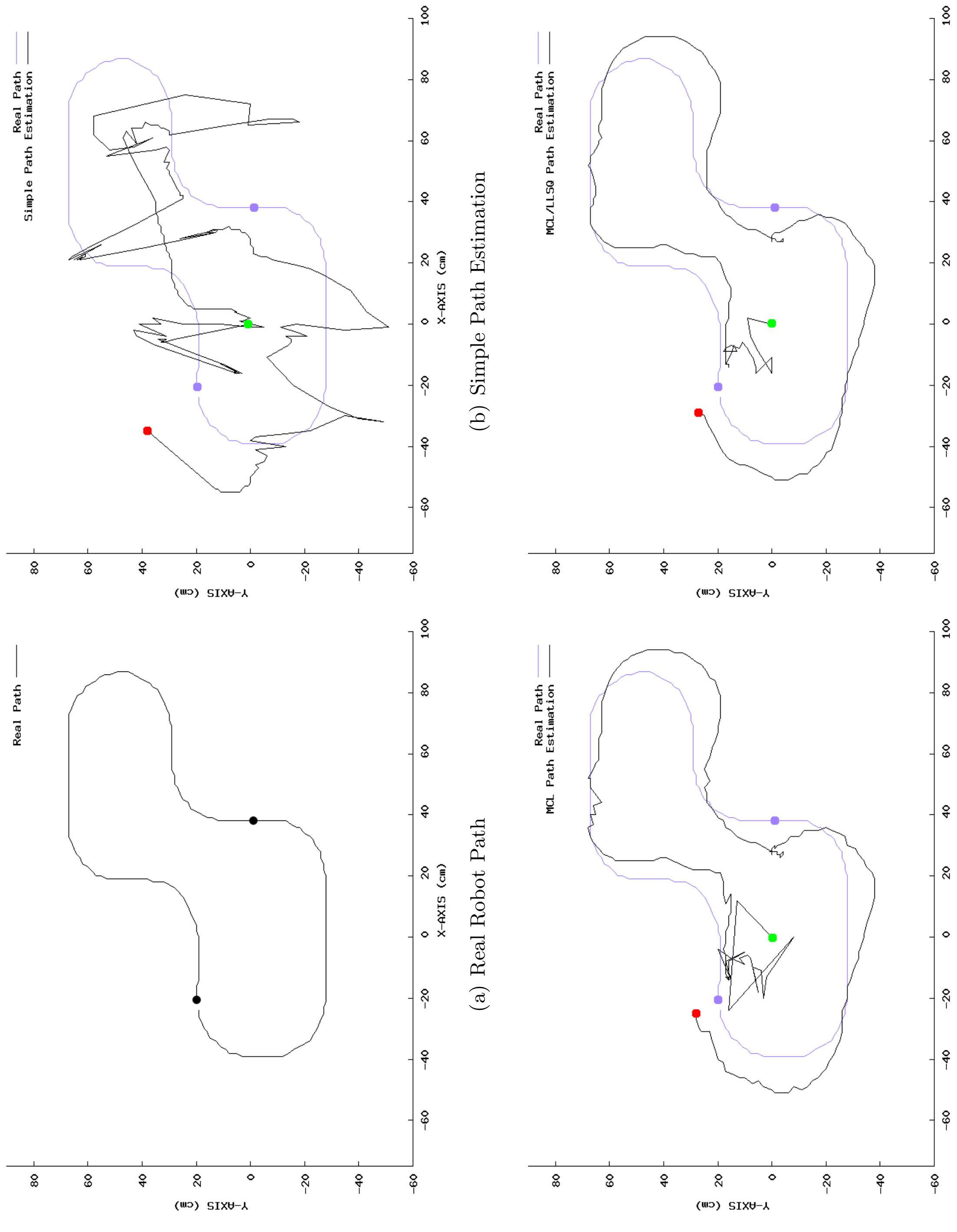
Figure 8.2. Standard Deviation for the Distance Between the Real and the Estimated Position.

turn. The estimation reached a good quality (all particles are located close to one position) after the turn. Additionally, sensor data updates the estimation. The resulting path has only minor peaks. After applying an LLSQ to this path (Figure 8.3(d)), the result is a good approximation of the real path.

This comparison has been executed with different numbers of particles (Figure 8.2). The results show that for particle numbers smaller than 2,000, the quality is unpredictable. It is dependent on the initial distribution of the particles and the values generated by the random number generator. For numbers of 2,000 and higher, the standard deviation ranges from 4.5 to 6.3 centimeters. The graph shows the results up to a maximum number of particles of 20,000. Additionally runs with up to 200,000 particles have been carried out with the same resulting standard deviation. Thus, the quality of the path approximation is bounded by the number of particles and the quality of sensor data. The blue line denotes the quality reached by the simple self-localization for comparison purposes. This algorithm uses no particles and no probabilistic model.

8.1.3 Multi-Hypothesis Resolving

The simulation runs with the MCL have shown that situations are quite common where several — equally plausible — possible position estimates appear. Figure 8.4(a) shows such a distribution of the particles with two centers. The estimated position is determined by a weighted average of all particles. The result is located on the half way between the two centers. Due to the fact that no particle is close to this position, the hypothesis is wrong. To overcome this problem, the particles have to be clustered to groups. Each cluster represents a hypothesis for the position. This results in the presence of multiple hypothesis. Using additional selecting mechanisms, the appropriate hypothesis is selected. Multi-hypothesis



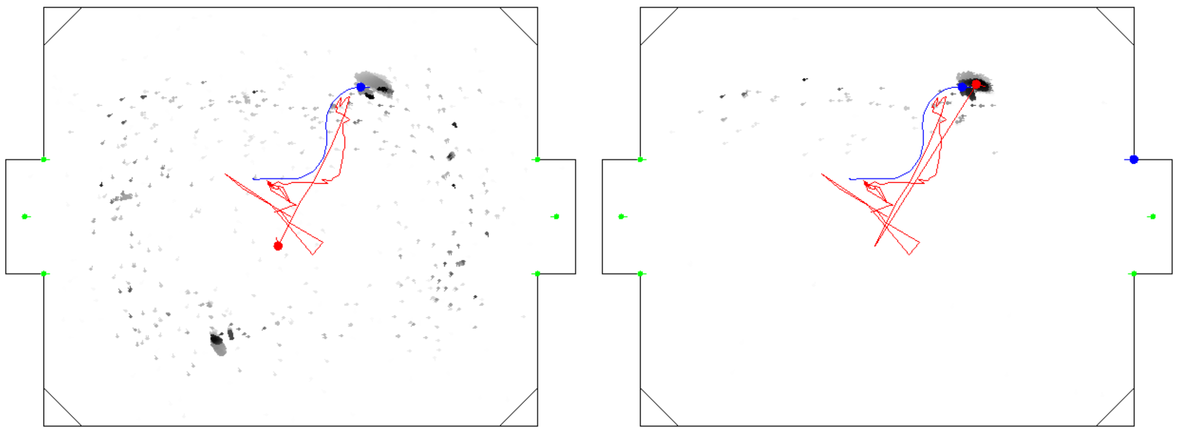
(a) Real Robot Path

(b) Simple Path Estimation

(c) MCL Path Estimation

(d) MCL/LLSQ Path Estimation

Figure 8.3. Simulation Results


(a) Ambiguous Situation

(b) Landmark Recognition Restores Single Hypothesis

Figure 8.4. Multi-Hypothesis

resolving is not in the scope of this work.

In the next step (Figure 8.4(b)), a landmark is sighted. The ambiguity is resolved immediately. The estimated position returns to the first hypothesis. The second hypothesis has disappeared.

8.2 Comparison of the Complex and an EKF Based Self-Localization

A comparison between the complex self-localization and an extended Kalman filter (EKF) is conducted in [BDN07]. The EKF is described in [BSGK07]. The setup of this comparison is slightly different to the one described in Section 8.1. The size of the playground is smaller (1.5 m by 1.2 m compared with 2.8 m by 2.2 m) and there are different landmarks available. In the previous comparison, there are three landmarks for each goal defined. The left and the right post and the center of the goal. While the left and the right landmarks provide only the direction, the center landmark also provides the distance between the robot and the landmark. In this comparison, there are only two landmarks for each goal. Each one provides the direction and the distance between the robot and the landmark.

Only the vision sensor and the odometry are used for localization. The vision system used for this comparison is described in [BSN05, BS06, BSGM06]. It differs from the one used in this thesis (see Section 6.2.3). The external interface is identical — it returns a vector corresponding to landmarks recognized by vision system together with the quality of this recognition.

Figure 8.5 shows the elements of the playground in this comparison. The playground is the one used for the MiroSOT small league. The robot is the Tinyphoon. The path traveled by the robot is a circle around the center with a radius of 50 cm. The starting position is next to the right goal. Four landmarks (two for each goal) are used. The gray circle beneath the robot is the area within which an estimation is acceptable (a circle with a radius of 12.5

cm). If the estimated position is within this range for at least three succeeding steps, it is assumed that the estimation algorithm successfully performed a global self-localization. The localization algorithms (EKF and MCL) are not given the starting position of the robot (bootstrap problem). The robot travels along the circle and reaches the starting position after 100 steps. During each step the robot receives data from the odometry. The vision system returns data only when landmarks are visible.

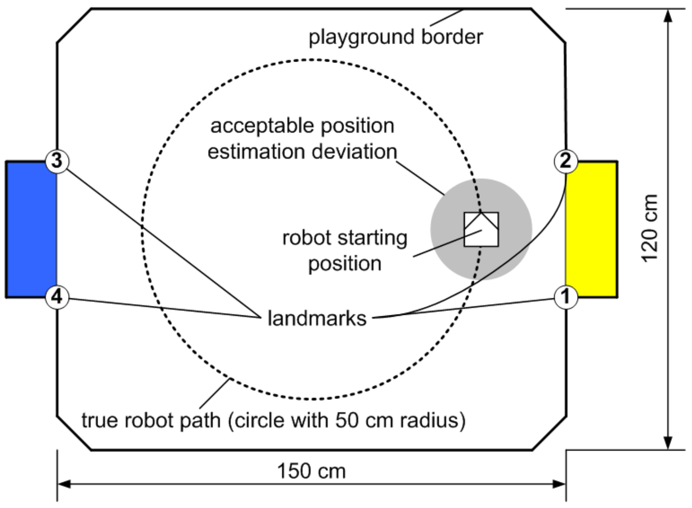

playground border
acceptable position estimation deviation
robot starting position
landmarks
true robot path (circle with 50 cm radius)
1
2
3
4
120 cm
150 cm

Figure 8.5. Environment of the Comparison [BDN07]

Five different sets of sensor data are used. The sets differ in the error added to the sensor data. All of them are collected while traveling the precise circle as described above. For each set, 25 runs for the algorithms are performed. A primary and a secondary quality feature are defined for this comparison. The primary feature is the number of steps with a successful self-localization within the last 20 steps. The secondary feature is the average distance to the true position within these last 20 steps.

The number of particles used for the MCL is 500. Table 8.1 gives the results of the simulation for the MCL with ten different numbers of particles. It shows the average distance between the estimated and the true position and the standard deviation. The row “primary feature” marks how often the primary comparison feature is fulfilled. Thus, a value of 0.9 means that in 18 out of 20 steps the primary feature was matched. The first number of particles that matched this feature in all 20 steps is used for the comparison with the EKF (column written in bold). The average distance settles around 9.0 for 3,000 particles and more.

A single run for one of the data sets is shown in Figure 8.6. The dashed line depicts the true path traveled by the robot. While the blue line marks the estimated path by the EKF, the estimation done by MCL is marked green.

The EKF starts at a random position and uses odometric data until the first landmark sighting is available (marked with ’A’) in step 33. The estimation is updated to this new data. The perceived vector to the landmark does not fit to the current estimate. Due to the low

Table 8.1. Simulation Results for Different Number of Particles

particles	100	150	250	**500**	1000	2000	3000	5000	7500	10000
average	18.4	14.6	11.3	**10.5**	9.6	9.4	9.0	9.1	8.9	9.1
deviation	27.4	16.1	8.5	**7.1**	5.7	4.2	3.3	2.7	2.7	2.0
primary feature	0	0	0.9	**1.0**	1.0	1.0	1.0	1.0	1.0	1.0

quality of the sighting together with the low quality of the previous estimation, the resulting estimated position is still outside the acceptable area defined above. This continues until the first sighting for the opposite goal is done (marked with 'C'). This recognized landmark shifts the estimated position close towards to the true position. Until the final position is reached, the estimation of the EKF is reaching the true position within a range of less than one centimeter.

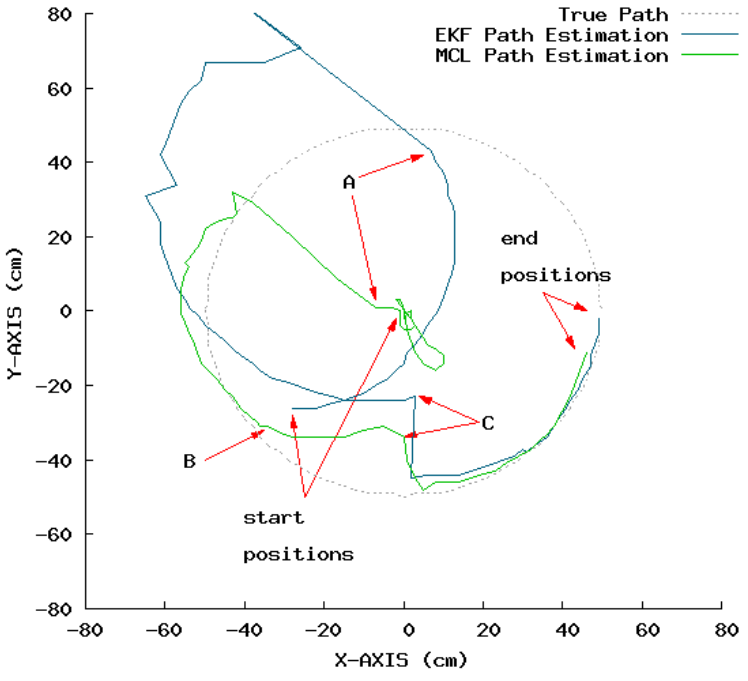


Figure 8.6. Path Estimations for a Single Run [BDN07]

Like the EKF, the MCL is not informed about the start position of the robot. Thus, the particles are distributed uniformly on the playground. This results in an estimated starting position in the center. Although odometric data is used, this estimate stays close to its origin. Due to the nature of the movement of the robot — a circular shaped path — the particles are moving in circles around the center. Thus, they are usually not far outside the borders of the playground. Particles that are outside are marked as unlikely and tend to disappear. As a result, the particles stay uniformly distributed until the availability of the vision data which then allows a global localization. The first sighting results in a position estimate close to the true position. Due to the fact that the particles were still uniformly distributed, enough fitting particles are available. Thus, the resulting estimate is close to the true position. The other particles are replaced with particles close to the estimated position (Sequential Importance Re-injection — SIR). The estimated path stays close to the true path, until the position marked with 'B' is reached. Now, for 15 steps no landmark has been visible to the camera system. Therefore, the cloud of particles has widened (compare with Figure 7.11 in Section 7.3.2). The right-hand side of this cloud is far outside the playground and marked as unlikely. Thus, the estimated path shifts towards left and travels parallel to the border. This is corrected with the first landmark sighting of the opponent goal (marked with 'C'). The final position is within the defined range of acceptance.

Figure 8.7 depicts the distances between the true path and the estimates shown in Figure 8.6 for each of the 100 steps. Additionally, the three steps 'A', 'B', and 'C' are above marked. The dashed gray line marks the maximum acceptable distance to the true position as defined above. Both algorithms — EKF and MCL — fulfil the primary quality feature (within the acceptable area for at least three consecutive steps) for all of the last 20 steps. EKF is better than MCL regarding the secondary quality feature (average distance to the true position) in the last 20 steps.

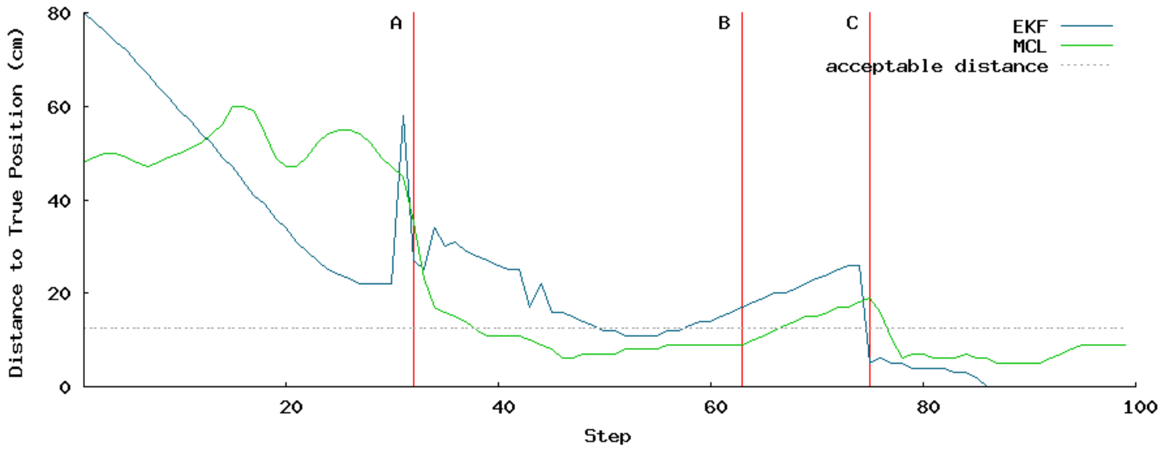


Figure 8.7. Distances between the Estimated Positions and the True Positions for a Single Run [BDN07]

The results for all 125 simulation runs are shown in Figure 8.8 for the EKF and in Figure 8.9 for the MCL. Both figures show the average value, the upper and the lower bound, and average ± the deviation. Further, a border is given below which the estimation performed successfully a global localization. The values before the 33rd step are strongly dependant on the randomly picked start position. Thus, the comparison focuses on values from the 34th to the 100th step. The average values are almost equal during the steps 40 to 80. Both algorithms meet the primary quality feature in the last 20 steps with at least their average values. The EKF meets it also with its worst case estimates (maximum values). Further, the estimates converge towards the true positions. A different result is shown for the MCL.

Although the average is below the threshold, almost half of the estimates are outside the acceptable distance. Hence, in this comparison EKF is superior to MCL.

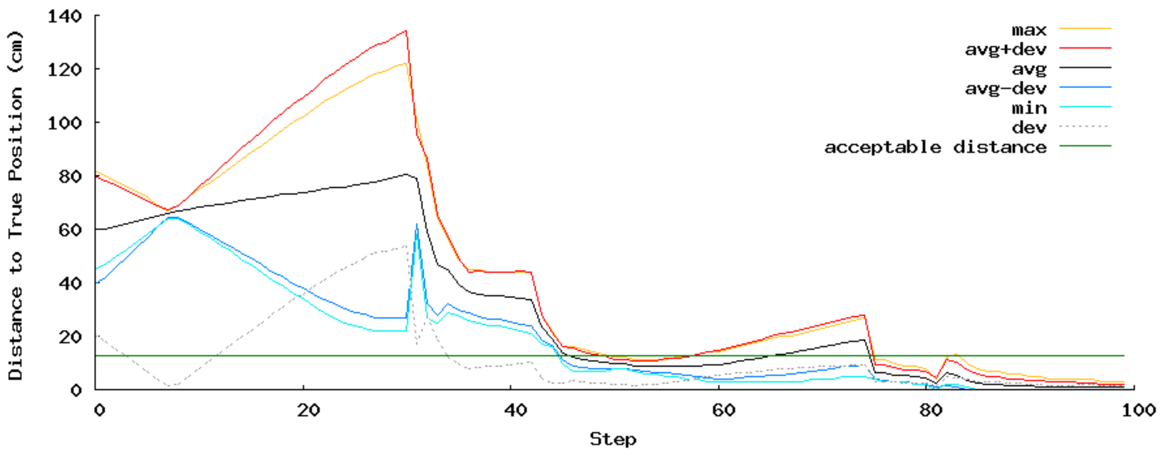


Figure 8.8. Comparison Results for the EKF for all Runs [BDN07]

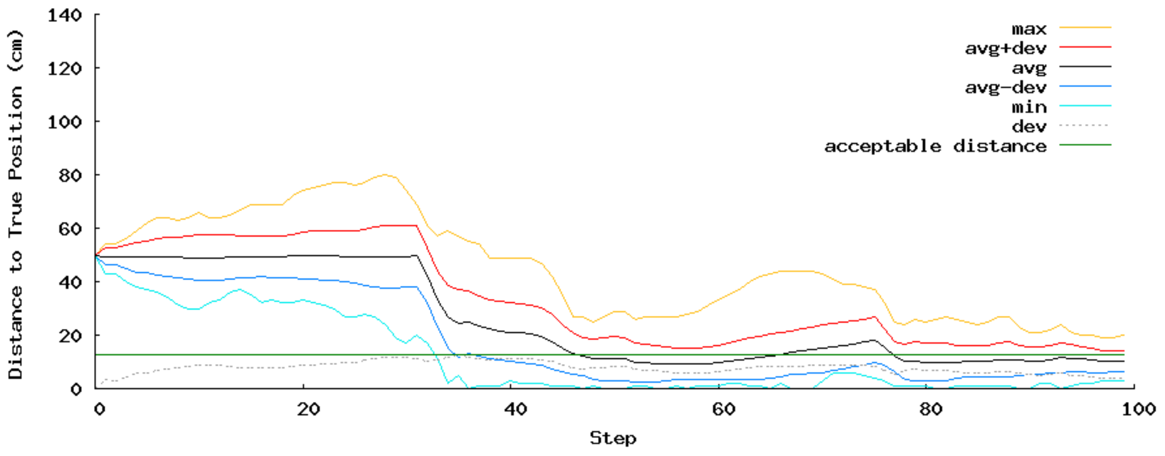


Figure 8.9. Comparison Results for the MCL for all Runs [BDN07]

8.3 Summary

The two self-localization approaches described in Chapter 7 are compared in a simulation. The simple approach should be used when only a rough position estimation is needed. The standard deviation of the estimated position to the real position is about 15 cm. The shape of the path is disturbed by many peaks and the robot appears to move back and forth. Thus, the estimated path can give no further information for threat analysis. The path estimated by the complex localization is of higher quality. The standard deviation of around 5 cm. The shape of the estimated path looks similar to the real one.

Further, tests to determine the optimal number of particles for the complex localization approach have been performed. 2,000 particles mark the point above which no additional quality can be gained and below which the result is varying. These variations are caused by the initial distribution of the particles and values generated by the random number generator.

A comparison between MCL and EKF implemented by another project shows results similar to the localization algorithm comparison of Section 5.7. Both algorithms perform well for their average estimates. If the worst estimates are taken into account, EKF is better than MCL.

For this comparison the determined optimal number of particles for MCL is 500. The reasons for the difference between 2,000 in the previous simulation and 500 in this simulation can be found in a smaller playground, a better vision system, a simpler path, and four instead of two fully featured landmarks.

9 Conclusion and Further Work

Alas!

— Alley Cat

To enable a robot to fulfill tasks autonomously, it has to have knowledge about its location. This knowledge consists of two parts: a map of the environment and the estimation of its position on this map.

In AMiroSOT, the environment consists of a playground and an a priori known number of moving objects. These objects are the ball, the team mates, and the opponents. The shape of all the entities is static during game play. What is changing is the position and the heading of the moving objects. A geometric map (see Section 4.2.1) has been chosen for the software system described in Chapter 7. All entities of this map are represented by either a geometric primitive or by a polygon. Such geometric figures can be replaced without much effort.

The world model repository (WMR) administrates this geometric map and provides a history of the positions of the moving objects (Chapter 4 and Section 7.2). The design of the map, the ball, the robots, and the history depends on the requirements of the applications using the WMR. This work has shown two approaches: a simple one and a complex one. The simple WMR is designed with a special focus on systems with little available resources. During the design of the complex WMR, resource considerations were of no relevance. It provides detailed information about all of the dynamic entities. Not only a position history for each one is available, furthermore, a prediction of future positions is provided.

The position estimation — or localization — is done by combining all available sensor data and the previously estimated position into a new position. This process is repeated each time new data is available. The results from the comparison in Section 5.7 have shown that there is no such thing as the perfect multipurpose localization algorithm. The particle filter has been chosen in this concept due to its good abilities to integrate different types of sensor data (see Section 5.6). This design decision is also supported by the comparison tests carried out in Chapter 8.

Open issues raised by this work are:

- Multi-hypothesis handling. Data from relative sensors like infrared distance sensors lead to particle distributions with more than one center. This results in ambiguity — if not cleared by sensor readings from the vision system — in having more than one valid position estimation. — see Section 8.1.3.
- Enhancement of the complex WMR by using a grid map. The distance information between a given point and the next wall in a certain direction can then be determined using a ray-tracing algorithm — see at the end of Section 7.4.3.
- An automatic calibration for the sensors and the Particle filter's parameters. This can be done by comparing the estimated position with a position determined by an external data source like a camera attached to a PC — see Section 7.4.3.
- More sophisticated prediction modules for the complex WMR. Currently, only steady movement can be predicted (e.g. the ball). Robots may change their direction without any external force. Thus, prediction for future robot positions has to include behavior analysis — see Section 7.4.2.

In robot soccer a team of robots has to cooperate to win the game. This cooperation has not necessarily to be restricted to a cooperation on a strategic level. Each robot perceives the playground only partly. The combination of this information increases the knowledge about the current situation. Thus, a cooperation by exchanging information stored in the WMR is desirable. This cooperative exchange of information accounts to the field of information fusion. A possible future work is to implement this information fusion based upon the results of this thesis.

Bibliography

[Bad07] Markus Bader. Feature based real-time stereo vision on a dual core dsp with an object detection algorithm. Master's thesis, Pattern Recognition and Image Processing Group (PRIP), Institute of Computer Aided Automation, Vienna University of Technology, February 2007. to be published.

[Bai02] Tim Bailey. *Mobile Robot Localisation and Mapping in Extensive Outdoor Environments*. PhD thesis, Australian Centre for Field Robotics, Department of Aerospace, Mechanical and Mechatronic Engineering, The University of Sydney, Australia, 2002.

[BAS$^+$06] Markus Bader, Miguel Albero, Robert Sablatnig, Jose E. Simo, Gines Benet, Gregor Novak, and Francisco Blanes. Embedded real-time ball detection unit for the yabiro biped robot. *Proceedings of the 4th Workshop on Intelligent Solutions in Embedded Systems*, 2006.

[BDN07] Abdul Bais, Tobias Deutsch, and Gregor Novak. Comparison of self-localization methods for soccer robots. Sumitted to 5th International IEEE Conference on Industrial Informatics (INDIN'07), July 2007.

[BFHS96] Wolfram Burgard, Dieter Fox, Daniel Hennig, and Timo Schmidt. Estimating the absolute position of a mobile robot using position probability grids. In *AAAI/IAAI, Vol. 2*, pages 896–901, 1996.

[BM03] Hans-Dieter Burkhard and Hans-Arthur Marsiske. *Endspiel 2050*. Telepolis, HEISE, 2003. ISBN 393693102X.

[Bro91] Rodney A. Brooks. Intelligence without representation. *Artificial Intelligence*, 47:139–159, 1991.

[BS06] A. Bais and R. Sablatnig. Landmark based global self-localization of mobile soccer robots. In *Computer Vision ACCV 2006: 7th Asian Conference on Computer Vision*, 2006.

[BSGK07] Abdul Bais, Robert Sablatnig, Jason Gu, and Yahya M. Khawaja. Location tracker for a mobile robot. Sumitted to 5th International IEEE Conference on Industrial Informatics (INDIN'07), July 2007.

[BSGM06] A. Bais, R. Sablatnig, J. Gu, and S. Mahlknecht. Active single landmark based global localization of autonomous mobile robots. In *Proceedings of 2nd International Conference on Visual Computing (ISVC 2006)*, 2006.

[BSN05] A. Bais, R. Sablatnig, and G. Novak. Line-based landmark recognition for self-localization of soccer robots. In *IEEE International Conference on Emerging Technologies (ICET '05)*, pages 132–137, Islamabad, Pakistan, September 2005.

[CLH$^+$05] Howie Choset, Kevin M. Lynch, Seth Hutchinson, George A Kantor, Wolfram Burgard, Lydia E. Kavraki, and Sebastian Thrun. *Principles of Robot Motion: Theory, Algorithms, and Implementations.* MIT Press, June 2005. ISBN 0-262-03327-5.

[CWEAB00] Howie Choset, Sean Walker, Kunnayut Eiamsa-Ard, and Joel Burdick. Sensor-based exploration: Incremental construction of the hierarchical generalized voronoi graph. *The International Journal of Robotics Research*, 19(2):126–148, February 2000.

[DLP$^+$06] Tobias Deutsch, Roland Lang, Gerhard Pratl, Elisabeth Brainin, and Samy Teicher. Applying psychoanalytic and neuro-scientific models to automation. *The 2nd IET International Conference on Intelligent Environments*, pages 111–118, 2006.

[EI03] Wilfried Elmenreich and Richard Ipp. Introduction to ttp/c and ttp/a. *Proceedings of the Workshop on Time-Triggered and Real-Time Communication Systems*, 2003.

[EM92] S.P. Engelson and D.V. McDermott. Error correction in mobile robot map learning. *Robotics and Automation*, 3:2555–2560, 1992.

[ENW05] Uwe Egly, Gregor Novak, and Daniel Weber. Decision making for mirosot soccer playing robots. *Decision Making for MiroSOT Soccer Playing Robots*, pages 69–72, 2005.

[FBT99] Dieter Fox, Wolfram Burgard, and Sebastian Thrun. Markov localization for mobile robots in dynamic environments. *Journal of Artificial Intelligence Research*, 11:391–427, 1999.

[FG97] Stan Franklin and Art Graesser. Is it an agent, or just a program?: A taxonomy for autonomous agents. In *ECAI '96: Proceedings of the Workshop on Intelligent Agents III, Agent Theories, Architectures, and Languages*, pages 21–35, London, UK, 1997. Springer-Verlag.

[FIR06] FIRA. Fira small league mirosot game rules, 2006. [Online `http://www.fira.net/soccer/mirosot/rules_slm.html`; accessed 2-December-2006].

[Fox98] Dieter Fox. *Markov Localization: A Probabilistic Framework for Mobile Robot Localization and Navigation.* PhD thesis, Institute of Computer Science III, University of Bonn, Germany, 1998.

[FT06] Dieter Fox and Sebastian Thrun. *Probabilistic Robotics.* MIT Press, 2006. ISBN 0-262-20162-3.

[Gat97] Erann Gat. On three-layer architectures. *Artificial Intelligence and Mobile Robot*, 1997.

[GBFK98] Jochen S. Gutmann, Wolfram Burgard, Dieter Fox, and Kurt Konolige. An experimental comparison of localization methods. In *Proc. of the IEEE/RSJ InternationalConference on Intelligent Robots and Systems*, 1998.

[GF02] Jochen S. Gutmann and Dieter Fox. An experimental comparison of localization methods continued. In *Proc. of the IEEE/RSJ International Conference on Intelligent Robots and Systems*, 2002.

[GH98] Guenther Greiner and Kai Hormann. Efficient clipping of arbitrary polygons. *ACM Transactions on Graphics*, 17(2):71–83, 1998.

[GHJV95] Erich Gamma, Richard Helm, Ralph Johnson, and John Vlissides. *Design Patterns: Elements of Reusable Object-Oriented Software.* Addison-Wesley Professional, 1995. ISBN 0201633612.

[Hec99] Paul Heckbert, editor. *Graphics Gems IV.* Academic Press Inc.,U.S., 1999. ISBN 0123361559.

[Hsu06] Feng-Hsiung Hsu. Chess hardware in deep blue. *Computing in Science & Engineering*, 8(1):50–60, 2006.

[Kai99] Hermann Kaindl. Difficulties in the transition from OO analysis to design. *IEEE Software*, 16(5):94–102, 1999.

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

[KDB$^+$06] Stefan Krywult, Tobias Deutsch, Markus Bader, Gregor Novak, and Abel Gonzales Onrubia. Autonomous mirosot - the autonomous way of playing mirosot. *Proceedings of the FIRA RoboWorld Congress 2006*, pages 163–167, 2006.

[KFM02] Cody Kwok, Dieter Fox, and Marina Meil. Real-time particle filters. *Advances in Neural Information Processing Systems 15*, 2002.

[Kry06] Stefan Krywult. Real-time communication systems for small autonomous robots. Master's thesis, Technische Universität Wien, Institut für Computertechnik, 2006.

[Mac93] Alan K. Mackworth. On seeing robots. Technical Report TR-93-05, Department of Computer Science, University of British Columbia, 1993.

[May79] Peter S. Maybeck. *Stochastic models, estimation, and control*, volume 141 of *Mathematics in Science and Engineering.* 1979. ISBN 0124807038.

[MON05] Stefan Mahlknecht, Roland Oberhammer, and Gregor Novak. A real-time image recognition system for tiny autonomous mobile robots. *Real-Time Systems*, 29:247–261, 2005.

[Neg03] Rudy Negenborn. Robot localization and kalman filters. on finding your position in a noisy world. Master's thesis, UTRECHT UNIVERSITY, Institute of Information and Computing Sciences, 2003.

[NM04] Gregor Novak and Stefan Mahlknecht. Tinyphoon - a tiny autonomous mobile robot. *Proceedings of the IEEE International Symposium on Industrial Electronics 2005*, 2004.

[Nov02] Gregor Novak. *Multi Agent Systems Robot Soccer.* PhD thesis, Technische Universität Wien, Institut für Handhabungsgeräte und Robotertechnik, 2002.

[NRB$^+$06] Gregor Novak, Charlotte Roesener, Markus Bader, Tobias Deutsch, Stefan Jakubek, Stefan Krywult, and Martin Seyr. The tinyphoons control concept. *Proceedings of ICM 2006, IEEE 3rd International Conference on Mechatronics, 3.-5. Juli 2006, Budapest, Ungarn, 2006.*, pages 625–630, 2006.

[NS04] Gregor Novak and Martin Seyr. Simple path planning algorithm for two-wheeled differentially driven (2wdd) soccer robots. *Proceedings of the second workshop on intelligent solutions in embedded systems*, 2004.

[NWB$^+$03] Issa Nesnas, Anne Wright, Max Bajracharya, Reid Simmons, Tara Estlin, and Won Soo Kim. Claraty: An architecture for reusable robotic software. *SPIE Aerosense Conference, Orlando, Florida*, 2003.

[Pat05] David A. Patterson. Robots in the desert: a research parable for our times. *Commun. ACM*, 48(12):31–33, 2005.

[RBD$^+$05] Thomas Röfer, Ronnie Brunn, Ingo Dahm, Matthias Hebbel, Jan Hoffmann, Matthias Jüngel, Tim Laue, Martin Lötzsch, Walter Nistico, and Michael Spranger. Germanteam 2004: The german national robocup team. In *RoboCup*

2004: Robot Soccer World Cup VIII, Lecture Notes in Artificial Intelligence, Lisbon, Portugal, 2005. Springer. more detailed in GermanTeam RoboCup 2004. Technical Report (299 pages, `http://www.germanteam.org/GT2004.pdf`).

[SHP+03] Paul S. Schenker, Terrance L. Huntsberger, Paolo Pirjanian, Eric T. Baumgartner, and Edward Tunstel. Planetary rover developments supporting mars exploration, sample return and future human-robotic colonization. *Auton. Robots*, 14(2-3):103–126, 2003.

[SJ05] Martin Seyr and Stefan Jakubek. Mobile robot predictive trajectory tracking. In *ICINCO*, pages 112–119, 2005.

[Thr02a] Sebastian Thrun. Particle filters in robotics. *Proceedings of the 17th Annual Conference on Uncertainty in AI (UAI)*, 2002.

[Thr02b] Sebastian Thrun. Robotic mapping: A survey. *Exploring Artificial Intelligence in the New Millenium*, 2002.

[VNE+01] Richard Volpe, Issa Nesnas, Tara Estlin, Darren Mutz, Richard Petras, and Hari Das. The claraty architecture for robotic autonomy. *Proceedings of the IEEE Aerospace Conference, Montana, March 2001*, 2001.

[WB95] Greg Welch and Gary Bishop. An introduction to the kalman filter. Technical report, University of North Carolina at Chapel Hill, Chapel Hill, NC, USA, 1995.

[Web05] Daniel Weber. Decision making in the robot soccer domain. Master's thesis, Technische Universität Wien, Institut für Informationssysteme, 2005.

[Wen00a] Lothar Wenzel. Kalman-filter, teil 1. *Elektronik*, 6:64–75, 2000.

[Wen00b] Lothar Wenzel. Kalman-filter, teil 2. *Elektronik*, 8:50–55, 2000.

[Wen00c] Lothar Wenzel. Kalman-filter, teil 3. *Elektronik*, 11:52–58, 2000.

[Wen00d] Lothar Wenzel. Kalman-filter, teil 4. *Elektronik*, 13:74–78, 2000.

[Wik06a] Wikipedia. Kalman filter — wikipedia, the free encyclopedia, 2006. [Online `http://en.wikipedia.org/w/index.php?title=Kalman_filter&oldid=36157405`; accessed 25-January-2006].

[Wik06b] Wikipedia. Law of sines — wikipedia, the free encyclopedia, 2006. [Online `http://en.wikipedia.org/w/index.php?title=Law_of_sines&oldid=91597800`; accessed 2-December-2006].

[Wik06c] Wikipedia. Odometry — wikipedia, the free encyclopedia, 2006. [Online `http://en.wikipedia.org/w/index.php?title=Odometry&oldid=64505378`; accessed 28-August-2006].

[Wik06d] Wikipedia. Voronoi diagram — wikipedia, the free encyclopedia, 2006. [Online `http://en.wikipedia.org/w/index.php?title=Voronoi_diagram&oldid=38445806`; accessed 12-February-2006].

[WPF98] W. Vetterling W. Press, S. Teukolsky and B. Flannery. *Numerical Recipes in C.* Cambridge University Press, New York, 1998. ISBN 0-521-43108-5.

[YJ99] Wai K. Yeap and Margaret E. Jefferies. Computing a representation of the local environment. *Artif. Intell.*, 107(2):265–301, 1999.

A

Acronyms

AI	Artificial intelligence
AMiroSOT	Autonomous Micro Robot World Cup Soccer Tournament
DARPA	Department of Defense Advanced Research Project Agency
EKF	Extended Kalman filter
FIRA	Federation of International Robot-soccer Association
GVD	Generalized Voronoi graph
IR	Infared
KF	Kalman filter
LLSQ	Linear least squares filter
MCL	Monte Carlo Localization / Particle filter
MiroSOT	Micro Robot World Cup Soccer Tournament
ML	Markov localization
OOA	Object-oriented analysis
OOD	Object-oriented design
SIR	Sequential Importance Resampling
SIS	Sequential Importance Sampling
UML	Unified modeling language
WMR	World model repository

“To be is to do” — Socrates
“To do is to be” — Jean-Paul Sartre
“Do be do be do” — Frank Sinatra

— Kurt Vonnegut, Jr.