The Memory-Based Paradigm for Vision-Based Robot Localization

D I S S E R T A T I O N

zur Erlangung des akademischen Grades

Dr. rer. nat.
im Fach Informatik

eingereicht an der
Mathematisch-Naturwissenschaftlichen Fakultät II
Humboldt-Universität zu Berlin

von
Dipl.-Inf. Matthias Jüngel

Präsident der Humboldt-Universität zu Berlin:
Prof. Dr. Jan-Hendrik Olbertz

Dekan der Mathematisch-Naturwissenschaftlichen Fakultät II:
Prof. Dr. Elmar Kulke

Gutachter:
1. Prof. Dr. Hans-Dieter Burkhard
2. Prof. Dr. Raúl Rojas
3. Prof. Dr. Verena Hafner

eingereicht am: 13.07.2011
Tag der mündlichen Prüfung: 09.05.2012

Abstract

For autonomous mobile robots, a solid world model is an important prerequisite for decision making. Current state estimation techniques are based on Hidden Markov Models and Bayesian filtering. These methods estimate the state of the world (belief) in an iterative manner. Data obtained from perceptions and actions is accumulated in the belief which can be represented parametrically (like in Kalman filters) or non-parametrically (like in particle filters). When the sensor's information gain is low, as in the case of bearing-only measurements, the representation of the belief can be challenging. For instance, a Kalman filter's Gaussian models might not be sufficient or a particle filter might need an unreasonable number of particles.

In this thesis, I introduce a new state estimation method which doesn't accumulate information in a belief. Instead, perceptions and actions are stored in a memory. Based on this, the state is calculated when needed. The system has a particular advantage when processing sparse information. This thesis presents how the memory-based technique can be applied to examples from RoboCup (autonomous robots play soccer). In experiments, it is shown how four-legged and humanoid robots can localize themselves very precisely on a soccer field. The localization is based on bearings to objects obtained from digital images. This thesis presents a new technique to recognize field lines which doesn't need any pre-run calibration and also works when the field lines are partly concealed and affected by shadows.

Zusammenfassung

Für mobile autonome Roboter ist ein solides Modell der Umwelt eine wichtige Voraussetzung um die richtigen Entscheidungen zu treffen. Die gängigen existierenden Verfahren zur Weltmodellierung basieren auf dem Bayes-Filter und verarbeiten Informationen mit Hidden Markov Modellen. Dabei wird der geschätzte Zustand der Welt (Belief) iterativ aktualisiert, indem abwechselnd Sensordaten und das Wissen über die ausgeführten Aktionen des Roboters integriert werden; alle Informationen aus der Vergangenheit sind im Belief integriert. Wenn Sensordaten nur einen geringen Informationsgehalt haben, wie zum Beispiel Peilungsmessungen, kommen sowohl parametrische Filter (z.B. Kalman-Filter) als auch nicht-parametrische Filter (z.B. Partikel-Filter) schnell an ihre Grenzen. Das Problem ist dabei die Repräsentation des Beliefs. Es kann zum Beispiel sein, dass die gaußschen Modelle beim Kalman-Filter nicht ausreichen oder Partikel-Filter so viele Partikel benötigen, dass die Rechendauer zu groß wird.

In dieser Dissertation stelle ich ein neues Verfahren zur Weltmodellierung vor, das Informationen nicht sofort integriert, sondern erst bei Bedarf kombiniert. Das Verfahren wird exemplarisch auf verschiedene Anwendungsfälle aus dem RoboCup (autonome Roboter spielen Fußball) angewendet. Es wird gezeigt, wie vierbeinige und humanoide Roboter ihre Position und Ausrichtung auf einem Spielfeld sehr präzise bestimmen können. Grundlage für die Lokalisierung sind bildbasierte Peilungsmessungen zu Objekten. Für die Roboter-Ausrichtung sind dabei Feldlinien eine wichtige Informationsquelle. In dieser Dissertation wird ein Verfahren zur Erkennung von Feldlinien in Kamerabildern vorgestellt, das ohne Kalibrierung auskommt und sehr gute Resultate liefert, auch wenn es starke Schatten und Verdeckungen im Bild gibt.

Contents

1 Introduction

In 1920, the term *robot* was first used by Czech writer Karel Capek in his play R.U.R. (Rossum's Universal Robots). The robot concept was then taken over by authors like Stanislaw Lem and Isaac Asimov in their science-fiction novels. Less than a century later, robots are not fiction any longer: they play an important role in our lives. Robotics has emerged as an amazing field of study with an endless list of robot-related research topics.

Stationary robots, like the ones at assembly lines, are common since many decades. The construction of mobile robots is more challenging. They can be useful in a lot of places: in disaster areas, in space, on the moon or on other planets, in caves or mines, under water, airborne, in hospitals, in homes or gardens, on streets, etc. (cf. [33]). The mobile nature of such robots sets high requirements for the hardware. A lasting power supply and good actuator design are among the biggest hardware related challenges. Autonomous mobile robots are a special form of mobile robots [69, 40]. Such robots act on their own, making decisions based on their sensory input. For autonomous mobile robots, the selection of sensors and the way the sensor data is processed are important design criteria [24].

Robots to perform very simple tasks can be constructed with a direct coupling between sensors and actors, similar to Braitenberg vehicles [9]. However, for more complex behavior some sort of internal representation is needed. A good representation of the robot's environment is a good basis for autonomous decision making [77, 55]. When such a representation can't be created based on the measurements made at a single time point, some form of data integration has to be done. Sparse sensor information and noisy measurements are the biggest challenge when creating a reliable model of the world.

A good example of robots with complex behavior are soccer playing robots. My engagement in *RoboCup* motivated the research described in this thesis. RoboCup is an annual competition where teams of autonomous robots play soccer. The vision of the RoboCup initiative is that in 2050 a team of humanoid man-like robots will play against the world champion team of human soccer players - and will win [10, 14]. In today's RoboCup leagues, there are smaller humanoid robots and wheeled robots. Until 2008, there also were 4-legged dog-like robots (Sony's Aibo). Our research group participated in RoboCup competitions with the *Aibo Team Humboldt* and the *GermanTeam* (with colleagues from Bremen, Darmstadt, and Dortmund) [12, 21, 63, 62, 61, 59, 60, 6].

While improving the skills of our team's 4-legged robots, I found the creation of a solid world model as a basis for decision making to be the biggest challenge. All aspects of the world around a robot that are important for decision making can be denoted as the robot state. Robotic literature provides many state estimation techniques; there are a lot of approaches for different purposes [77, 32, 3, 54, 81]. However, I had the desire to improve state estimation quality for robotic soccer and problems with similar specifics. Many of the existing techniques were developed for different robot types (wheel-based instead of legged robots) or different sensors

(laser-range finders instead of cameras). Additionally, soccer robots have limited processing power due to weight constraints, excluding a lot of the existing methods. The permanent physical interaction with other robots adds even more challenges.

This thesis contributes a novel method of state estimation that was inspired by my work with the four-legged robots. Based on that, a localization method for soccer robots is introduced. Data extracted from the robot's camera serves as input to this localization method. I describe a new approach to extract soccer field lines from digital images. All these methods proved to be useful for four-legged and for humanoid robots.

1.1 State Estimation Problem

In robotics, *state estimation* denotes techniques that enable a robot to model those aspects of its environment that are relevant for decision making. Examples of qualities-of-interest are the robot's position and relative positions to objects around it. An important characteristic of most environments is that they change over time. Examples are: moving objects, moving persons, changing light, changing temperature, etc. Some of the changes can be caused by the robot itself, others might have different reasons. As autonomous mobile robots have to make consecutive decisions, the state estimation goal is to create a continuous *world model* that describes parts of the real world and reflects all relevant changes.

Sensors An important prerequisite for a robot to obtain a world model are its sensors. Sensors measure certain real world quantities more or less directly. While a digital thermometer is a device to measure temperature, in reality, it measures an electrical component's resistance that reacts to temperature. The resistance measurement itself might, in fact, be a voltage and current measurement.

However, there can be quantities of interest for which no sensor exists or the robot might not be equipped with such a sensor. For example, a robot might be interested in a certain object's speed. If the robot's only sensor is a digital camera, it has to take several images to calculate the object's speed. A robot trying to determine its position using landmark observation is another example where multiple measurements have to be combined to calculate a certain quantity.

Strictly speaking, a sensor is a device that produces measurements, usually numeric values, that somehow correspond to real-world states. A sensor is useful for state estimation when the relationship between its measurements and the world state is known. Such a description of a sensor is often referred to as a *sensor model*, cf. section 2.2.3.

State Changes Apart from sensors, another important means for a robot to update its world model is knowledge about state changes. Imagine a robot that came up with a representation of its position within its environment. When it performs actions, like *move forward*, it can update its world model, based on knowledge about the effect of the actions, without using any sensor. This is an example for a state change caused by a robot's action.

The state can also change on its own. For instance, a moving object changes its position over time depending on its speed. Another example is the temperature of a hot object cooling down over time. In such cases, the world model can be updated using knowledge about physical

properties, like momentum or friction. The moving object example shows that certain qualities, like speed, might be of interest when updating the world model. While a robot might just be interested in the position of a certain object for decision making, calculating its speed can be important to determine the position.

The description of how the state changes, depending on the previous state and possible actions of the robot, is often referred to as *state transition model*, cf. section 2.2.4.

Common State Estimation Techniques There are a lot of robot state estimation methods. They differ in the way the sensor and the state transition model are represented. Another distinction between different state estimation techniques is how information is processed and stored.

A state estimation approach common in robotics is Bayesian filtering. Different Bayes filter types are described in chapter 2. Popular ones are *particle filters* [77, 15, 32, 1] and *Kalman filters* [49, 81, 51]. All Bayesian techniques use probabilistic sensor and state transition models. The state is described by an approximation of a probability density function over the state space. This state description is updated, iteratively, using the current sensor readings and knowledge about state changes. The methods differ in the way this function is approximated.

1.2 Soccer Robot Localization Specifics

Localization of robots on a soccer field is an interesting state estimation application. Many existing self-localization methods were designed for robots equipped with laser range finders. The focus of this thesis is on robots that use cameras as their main sensor, which brings its own problems. More difficulties arise from the dynamic nature of soccer games. These are the main challenges of vision-based soccer robot self-localization:

- *Limited angle of view.* Compared to laser range finders and 360° cameras, regular cameras have a limited angle of view. This holds true even for wide angle cameras and results in less sensory input.
- *Unsuitable direction of view.* The direction a robot looks in is often determined by the most important object in the environment; this means that all other objects are seen less often. An example is a soccer robot staring at the ball and seeing less of the goals and field lines.
- *Sparse or poor information in sensor readings.* A camera provides only bearing information with high accuracy. Vision-based distance measurements are usually inaccurate.
- *Systematic error in odometric data.* The effects of actions performed by legged robots are less predictable than the ones of wheeled robots because of the long kinematic chains. Joint wear-out can lead to systematic error in odometric data. For example, a robot attempting to walk on a straight line might walk along an arc.
- *Physical interaction with other robots.* The more often a robot collides with environmental objects or with other robots, the less accurate its information about the effects of the

executed motions. A robot also might get stuck on the way through its environment while still performing forward motions. Collisions and obstructions might lead to a large deviation between the assumed and the real motions, this is often referred to as kidnapping, cf. 2.4.2.

1.3 Contributions

The statement of this thesis is that state estimation accuracy can be improved, compared to iterative state estimation techniques, when it is not done sequentially but based on a short-term memory of perceptions and performed actions.

The main contribution of this thesis is the *memory-based paradigm*. It is a guide to how to solve state estimation problems. The method can process sparse sensor information, is able to cope with kidnappings and can handle systematic error; these are challenges for iterative state estimation techniques.

Furthermore, two memory-based self-localization methods and a vision system for field line detection are introduced. All techniques presented here were proven to work on real-world data.

While in this thesis examples from RoboCup are used to illustrate and verify the novel methods, they are not limited to the robotic soccer domain.

1.4 Outline

Chapter 2 wraps up existing state estimation techniques. The *probabilistic paradigm* and the concept of sequential state estimation are described. Additionally, the Bayes filter and its applications, like the Kalman filter and particle filters, are discussed. Particular attention is paid to the limitations of Bayesian filtering.

Chapter 3 introduces the *memory-based paradigm*. It is a supplementation and an alternative for the probabilistic paradigm. The chapter gives motivation for this paradigm, describes the mathematic foundations of *memory-based state estimation (MBSE)*, introduces algorithmic variants of *MBSE*, and proves the concept.

In chapter 4, an *MBSE* application is given which uses only horizontal landmark bearings to localize a robot. Experiments that were done on a four-legged robot (Aibo) are used to illustrate the method's effectiveness and accuracy.

Chapter 5 describes a vision-based compass for soccer robots. The chapter contributes a vision system which recognizes field lines on soccer fields, without any pre-run calibration, and an *MBSE* method using recognized field lines to determine the robot's heading.

In chapter 6, an *MBSE* localization method is introduced which uses this rotation estimation and horizontal bearings. An important component of this method is the correction of faulty odometry based on the rotation estimates.

Chapters 5 and 6 contain the description of experiments that were conducted on a humanoid robot.

2 The Probabilistic Paradigm

A *robotic paradigm* defines principles for designing certain aspects of a robot architecture. Past robotic research has created a lot of paradigms. Popular paradigms were described by R.R. Murphy [53]. The *hierarchical paradigm*, the *reactive paradigm*, and the *hybrid deliberate/reactive paradigm* provide different ways to organize the robotic primitives sense, plan, and act. Other paradigms focus on more specific components, like perception or action selection. Which paradigm fits best, in a given scenario, depends on the problem's specifics. Robotic architectures can follow more than one paradigm when the principles stated by the paradigms don't exclude each other.

A paradigm used for state estimation on many robots is the *probabilistic paradigm* which was introduced and promoted by Thrun, Burgard, and Fox [77, 29, 27, 26, 34]. It pays tribute to the uncertainty in perception and action. With this paradigm, perception problems are seen as state estimation problems where the state is represented using a probability distribution. The knowledge about the probabilistic properties of the robot's sensors and actuators can be used to anticipate uncertainty. Such a prediction can be used for action selection in order to minimize the robot's uncertainty. While many very powerful methods follow the probabilistic paradigm, there are also disadvantages of methods based on this paradigm.

In this chapter, the basic concepts of the *probabilistic paradigm* are introduced. Additionally, some problems that might appear with robots following this paradigm are highlighted. Section 2.1 describes the concept of sequential state estimation, section 2.2 introduces Bayesian filtering, section 2.3 shows the most popular Bayes filter applications, and section 2.4 discusses limitations of methods based on the Bayes filter. These limitations were my main motivation to develop the *memory-based paradigm*, which is introduced in chapter 3.

2.1 Sequential State Estimation

The state estimation goal in robotics is to find out certain parameters about the world around the robot and the relation between the robot and the world. Those parameters are of interest for decision making. Classical examples for parameters describing the state are the robot's position within its world and the relative position of environmental objects. To be useful for decision making, there must be an estimation of the state for each moment in time. So, each state of the sequence of states has to be estimated. The concept of sequential state estimation is described in a couple of books and articles [23, 75, 76].

The evolution of states can be seen as a *Markov process* [7, 57], assuming a future state only depends on the current state, not on the previous states. A state that fulfills this condition is denoted as a *complete state* [77]. In some cases, the complete state has to contain parameters that might not be of interest. For example, the robot's speed and acceleration might needed to

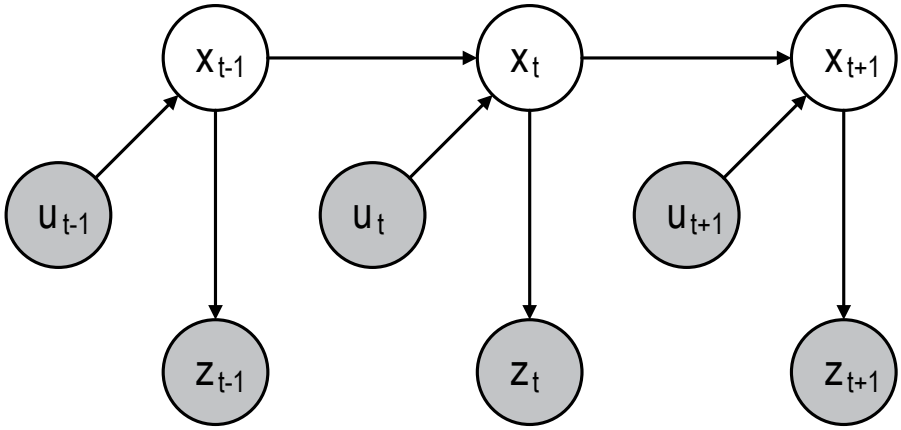


Figure 2.1: *Hidden Markov Model.* The states x_i are not directly visible. A state x_i depends only on state x_{i-1} and the control action u_i. An observation z_i depends only on the state x_i.

be included in the complete state; while for decision making, just the robot's position is needed.

However, when modeling the evolution of states as a Markov process, the states are sets of unknown parameters. The only information sources for changes in the world are the robot's sensor readings and knowledge about the actions the robot performs. So, such systems can be described using *Hidden Markov Models* [58, 31]. The vector describing the (unknown) state at time t is denoted x_t, the vector describing the sensor readings is denoted z_t, and the one describing the performed actions is denoted u_t. Figure 2.1 depicts a Hidden Markov Model that describes the connection between states, sensor readings, and performed actions.

With such a model, state estimation techniques based on Bayesian filtering can be applied. The following sections give more details.

2.2 Bayesian Filtering

In this section, the well known Bayes filter is introduced. For simplicity, the algorithm and all the prerequisites are given in discrete form; while in all cases there is also a corresponding continuous version. Thorough introductions to Bayesian filtering can be found in [77] and [22].

2.2.1 Bayes's Theorem, Law of Total Probability

The Bayes filter uses two theorems from probability theory which are repeated here. The *Law of total probability* [57, 11] gives the probability $p(x)$ for an event x when the conditional probability $p(x|y)$ is given for each event y out of a set of mutually exclusive events whose probabilities sum to unity:

Theorem 1. (Law of total probability)

$$p(x) = \sum_y p(x|y)p(y)$$

Bayes's theorem [57, 11] gives the relation between $p(x|y)$ and $p(y|x)$:

Theorem 2. (Bayes's theorem)

$$p(x|y) = \frac{p(y|x)p(x)}{p(y)}$$

2.2.2 Belief

Let X_t be a random variable describing the state to be estimated (the hidden state) at time t. If x_k is a specific state, out of the set of all possible states, then $p(X_t = x_k)$ gives the probability for the hidden state to be x_k. The set $\{p_{k,t}\}$ denotes the discrete probability distribution that contains the probability for all possible states at time t. This probability distribution is often denoted as *belief* at time t. The discrete Bayes filter is a recursive algorithm operating on such discrete probability distributions. Its inputs are the probability distribution at time $t-1$, denoted as $\{p_{k,t-1}\}$, the sensor readings z_t at time t, and the descriptions u_t of actions performed at time t. The output is the probability distribution $\{p_{k,t}\}$ at time t.

2.2.3 Probabilistic Sensor Model

A sensor model describes the sensor properties. A probabilistic description can be given using the *measurement probability* $p(z|x)$, which is the probability for the sensor to measure z, given that the current state is x. Note that this is a forward model. For a given system, it usually is easier to provide such a forward model (what is the expected measurement, given a specific state) than to provide the opposite (how is the state restricted, given a specific measurement).

2.2.4 Probabilistic State Transition Model

A state transition model (called *motion model*, when the state contains just parameters describing the robot's pose) describes how the state changes based on the robot's controls. A probabilistic description can be given using the *state transition probability* $p(x_i|x_{i-1}, u)$ which is the probability for the system to change from state x_{i-1} to state x_i, given the control u. Note that this is also a forward model.

2.2.5 Bayes Filter Algorithm

The Bayes filter algorithm is based on the notions and theorems introduced above. Algorithm 2.2 shows the discrete variant of the algorithm. The algorithm updates the probability distribution $\{p_{k,t}\}$ (the belief) for each step in time. This is done in two steps. The *control update* or *prediction step* calculates the prediction $\bar{p}(X_t = x_k)$ for all states x_k. The prediction is the probability for the robot to be in a specific state, given the last belief $\{p_{k,t-1}\}$ and the current

Input: $\{p_{k,t-1}\}, u_t, z_t$
foreach k **do**
  $\bar{p}(X_t = x_k) = \sum_i p(X_t = x_k | X_{t-1} = x_i, u_t) \cdot p(X_{t-1} = x_i)$
  $p(X_t = x_k) = \eta \cdot p(z_t | X_t = x_k) \cdot \bar{p}(X_t = x_k)$
end
return $\{p_{k,t}\}$

Algorithm 2.2: *Discrete Bayes filter.*

measurement u_t. For that calculation, the state transition model is used and the law of total probability (Theorem 1) applied.

The *correction step* or *measurement update* includes the current measurement z_t. The new belief is calculated based on the prediction $\bar{p}(X_t = x_k)$ and the current measurement for all states x_k and t_t. For that calculation, the sensor model is used and Bayes's theorem (Theorem 2) applied. Note that in Bayes's theorem, $1/p(z_t)$ is replaced by the normalizer η. This normalizer η has to be chosen such that all probabilities of the new distribution sum to unity.

A classical example to show how the Bayes filter works is a one-dimensional experiment where a robot walks down a hallway. This robot has a sensor that detects whether (or not) it is in front of a door. It localizes using the knowledge about the door positions, the actions it performs, and its sensor readings.

Figure 2.3 shows such an experiment's simulation result. In this experiment, the robot walks from left to right in 10 *cm* steps, performing a measurement every fifth step. For illustration purposes, the robot is simulated such that there is no error in the measurement and no error in the motion execution. However, for the belief calculation using the discrete Bayes filter the following measurement probabilities are assumed:

$$\begin{aligned}
p(Z_t = \text{door} \;\; | X_t = \text{in-front-of-door}) &= 0.95 \\
p(Z_t = \neg\text{door} | X_t = \text{in-front-of-door}) &= 0.05 \\
p(Z_t = \text{door} \;\; | X_t = \neg\text{in-front-of-door}) &= 0.30 \\
p(Z_t = \neg\text{door} | X_t = \neg\text{in-front-of-door}) &= 0.70
\end{aligned}$$

Furthermore, these state transition probabilities are used:

$$\begin{aligned}
p(X_t = x \qquad\quad | U_t = \text{walk-10-cm}, X_{t-1} = x) &= 0.15 \\
p(X_t = x + 10\ cm | U_t = \text{walk-10-cm}, X_{t-1} = x) &= 0.70 \\
p(X_t = x + 20\ cm | U_t = \text{walk-10-cm}, X_{t-1} = x) &= 0.15
\end{aligned}$$

These values lead to the Gaussian-like peaks in the belief. When the robot reaches the first

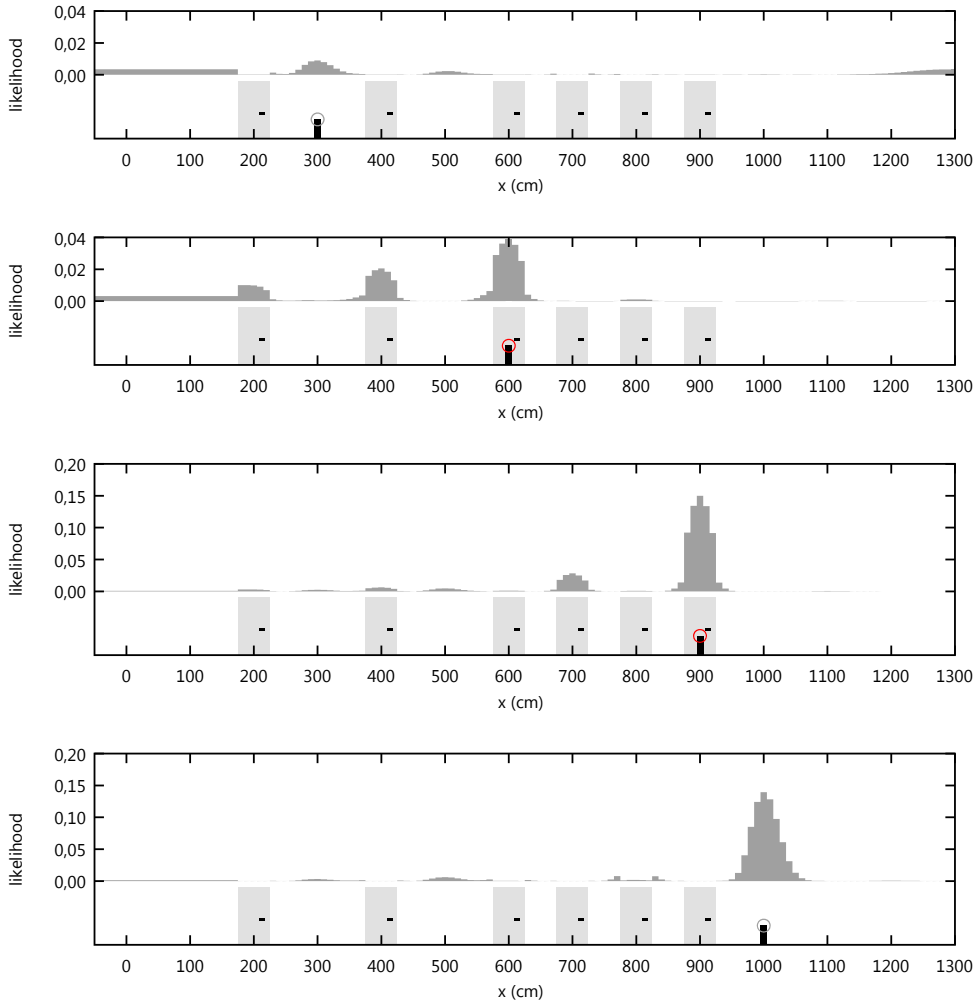


Figure 2.3: *Robot-in-hallway experiment.* The robot is visualized by the black vertical bar at the bottom of the diagrams. A gray circle shows that the door sensor does not detect a door; a red circle symbolizes a door detection. The light gray rectangles are the doors, the dark gray curve shows the belief. The robot starts at $x = 0$ and walks from left to right. The single diagrams show how the robot's belief changes.

door, the sequence of simulated (error-free) door detection measurements is:

$$\begin{aligned}
&\neg\text{in-front-of-door}(\quad 0\ cm)\\
&\neg\text{in-front-of-door}(\ 50\ cm)\\
&\neg\text{in-front-of-door}(100\ cm)\\
&\neg\text{in-front-of-door}(150\ cm)\\
&\quad\text{in-front-of-door}(200\ cm).
\end{aligned}$$

Only the first door has a 200 *cm* free space to the left. So, the belief calculated using the discrete Bayes filter has a single maximum after this sequence. The confidence (maximum likelihood) increases while the robot passes more doors on its way down the hallway.

In section 2.4.2, this experiment is used in a slightly modified version to show how the Bayes filter performs when kidnappings occur.

2.3 Bayes Filter Applications

The discrete Bayes filter variant can be understood as an implementation of the continuous form. However, there are many other filters based on the Bayes filter. They are briefly introduced in this section. A detailed introduction is omitted, as it can be found in many publications. Links to literature are given in the respective subsections.

The difference between the Bayes filter implementations is how they represent the belief. Based on this representation, two filter classes are distinguished: *Parametric filters* approximate the belief based on a fixed functional form (usually a Gaussian) of the distribution; *nonparametric filters* represent the belief using a finite number of values, where these values in some way correspond to regions in state space.

In the following subsections, three parametric filters (Kalman, extended Kalman, unscented Kalman) and two nonparametric filters (discrete Bayes, particle) are introduced.

2.3.1 Kalman Filter

The *Kalman filter (KF)* is a parametric filter. Informative introductions can be found in [49, 81, 51]. The Kalman filter uses a Gaussian, specified by its mean and its covariance, to describe the belief at a certain time. As the Kalman filter is a Bayes filter variant, it consists of a prediction step and a measurement update step. The Kalman filter is a valid Bayes filter implementation when the following three conditions hold:

- The state transition probability can be described using a linear function and a Gaussian error: $x_t = A_t x_{t-1} + B_t u_t + \varepsilon_t$
- The measurement probability can be described using a linear function and a Gaussian error: $z_t = C_t x_t + \delta_t$
- The initial belief is normally distributed

Systems that fulfill these conditions are called linear Gaussian systems. The Kalman filter can be extended to nonlinear problems. Two variants, the *extended Kalman filter* and the *unscented Kalman filter* are described in the following subsections. There are also extensions to the Kalman filter that address the unimodality. A technique known as multi-hypothesis Kalman uses a mixture of Gaussians to represent the current state.

2.3.2 Extended Kalman Filter

The *extended Kalman filter (EKF)* is a parametric filter. Instructive descriptions can be found in [77] and [73]. The extended Kalman filter does not require a linear system. So, the state transition and the measurement probabilities can be described with

$$x_t = g(u_t, x_{t_1}) + \varepsilon_t \text{ and}$$
$$z_t = h(x_t) + \delta_t$$

where g and h are nonlinear functions.

The EKF algorithm works like the KF algorithm but adds linearization. This linearization is achieved using first order Taylor expansion. The gradient to the functions is expressed using Jacobians. An excellent, detailed extended Kalman filter description can be found in [73].

2.3.3 Unscented Kalman Filter

The *unscented Kalman filter (UKF)* is a parametric filter. Insightful descriptions can be found in [48] and [79]. The unscented Kalman filter uses a different form of linearization, called the *unscented transform*. The functions g and h are not approximated like in the EKF. Instead, the algorithm calculates *sigma points* which are transformed using g and h. Based on the transformed sigma points, the transformed mean and covariance are calculated. The basic idea of the sigma points is that they probe how a nonlinear function changes the shape of the Gaussian to be transformed.

The mean of the original Gaussian and two more points per dimension are used as sigma points. These points are placed around the mean depending on weight factors. The UKF approximates the Gaussian of the transformed function more accurately than the EKF, especially when there is a high nonlinearity near the mean of the original distribution or when there is high uncertainty in the original distribution. A more detailed comparison of the EKF and the UKF is given in [54].

2.3.4 Discrete Bayes Filter

The discrete Bayes filter algorithm is the straightforward implementation of the continuous version and is given in algorithm 2.2. It is a nonparametric filter. Technical descriptions are given in [77] and [29]. When the state space itself is continuous, this filter is called *histogram filter*. The state space is decomposed into a finite number of *bins* by the algorithm. There are several ways this decomposition can be achieved. The most simple is to decompose the state space in equal-sized grid cells. Both the accuracy and the computational complexity are increased when the cell size is decreased.

Dynamic decomposition techniques take the posterior distribution's shape into account. An example for such a technique are density trees. They decompose the state space recursively, resulting in a decomposition that has a higher resolution at regions with higher probability. Another example is *selective updating*, which only updates the bins exceeding a user-defined threshold. With these dynamic decomposition techniques, the computational complexity of the discrete Bayes filter can be drastically decreased.

2.3.5 Particle Filter

Like histogram filters, particle filters are nonparametric filters. Particle filters have become widely used for robotic localization. There is a lot of literature that gives an overview [77, 15, 32, 1] and there are many publications that describe particle filter applications [26, 20, 78, 28, 64, 65]. A particle filter approximates the probability distribution using a random sample set drawn from the distribution. These samples are called *particles*. An important advantage of this method is that it is nonparametric and so can describe a lot of different distributions without knowing parameters like mean and covariance. Another benefit is that such samples can be passed straightforwardly through nonlinear transformation functions.

A particle can be seen as a hypothesis for the state. The more particles in a certain state space region, the higher the probability for that region in the approximated distribution. The higher the total particle number, the more accurate the approximation.

Like all other Bayes filter implementations, the particle filter consists of a prediction and a correction step to incorporate the knowledge about the actions and the measurements. The current action is incorporated by moving each particle in state space. These moves are based on a sample drawn from the distribution which describes the state transition probability. Measurements are incorporated by assigning each particle a weight which depends on the measurement probability.

Additionally, particle filters have a third step, called *resampling*. In that step, a new particle set is drawn from the particle set created by the prediction step, using the weights from the correction step. That operation's consequence is that some particles are no longer in the set, while some points in state space are represented by more than one particle in the set.

2.4 Bayes Filter Limitations

In the previous sections, the Bayes filter and its most important applications were introduced. Which application is suited best depends on the problem's characteristics. Several filter technique comparisons can be found in the literature [22, 54, 79, 48, 49, 73, 81].

With respect to accuracy, it can be said that the Bayes filter in its continuous form provides the best results (ignoring the computational complexity). Histogram filters can approximate this accuracy arbitrarily well, where the grid cell size determines how good the approximation is. The same holds true for particle filters where a higher particle number leads to a better approximation. However, to represent probability distributions describing high uncertainty, a very high particle number is needed.

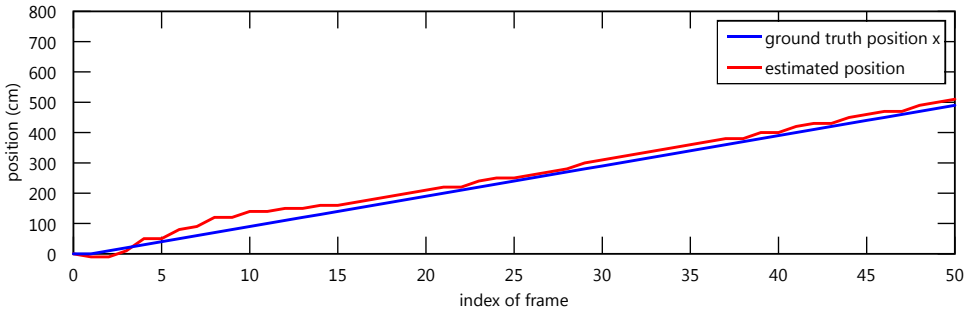


Figure 2.4: *Accurate position estimation.* The blue curve shows the real position of a robot moving along a line with constant speed. The red curve shows the position estimated by a discrete Bayes filter.

This section discusses how these techniques can cope with three particular problems: inaccuracies in the state transition model, kidnappings, and sparse sensor information. The limitations of the Bayes filter applications outlined below are my motivation for introducing the *memory-based paradigm*, described in chapter 3.

2.4.1 Inaccurate State Transition Models

This subsection discusses how a Bayes filter performs when the state transition model is inaccurate, using the following experiment.

Imagine a robot moving along a straight line, equipped with a single distance sensor. This sensor measures the distance to a fixed point along the line on which the robot moves. The robot moves from a position with $x = 0$ *cm* to a position with $x = 500$ *cm*. On each step, the robot measures the distance to the origin and intends to move 10 *cm*. The distance sensor is quite noisy and has a Gaussian error with a 75 *cm* standard deviation. The mean of that Gaussian distribution is equal to the robot's distance to the origin (there is no systematic error) and the standard deviation does not depend on that distance. The motion command execution (*move 10 cm*) is quite accurate: the standard deviation is 1 *cm* and there is no systematic error.

Figure 2.4 shows how good a discrete Bayes filter, using the models given above, can estimate the robot position. The high motion model accuracy leads to a high stability of the position result, as the Bayes filter incorporates past measurements with a relatively high weight.

Figure 2.5 shows what happens, if determining the position is just based on the most current measurement; the estimated position jumps with each measurement.

The Bayes filter requires an accurate state transition model in order to provide an accurate estimate. Figure 2.6 shows how the discrete Bayes filter estimates the position, for the above experiment, when there is a systematic error of 10 *cm* per step in the motion model. This systematic error accumulates and leads to an increasing deviation between the estimated and the real position.

In every case, the best solution for this problem is to avoid systematic error in the motion model. However, this is hard to achieve in some cases. Legged robots are an example where

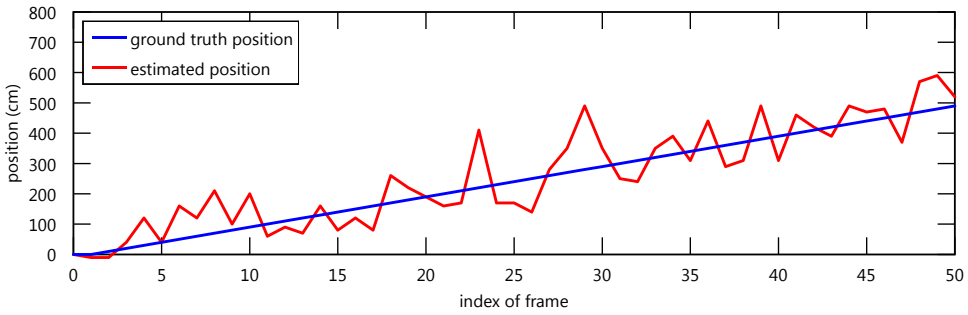


Figure 2.5: *Position estimation based on the most current measurement.* The blue curve shows the real position of a robot moving along a line with constant speed. The red curve shows the position estimation result which incorporates only the most current measurement. The noisy sensor leads to a noisy position estimation.

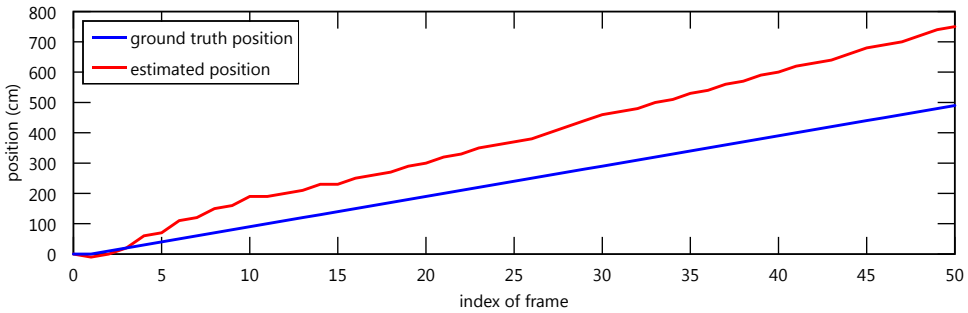


Figure 2.6: *Position estimation using a state transition model with a systematic error.* The blue curve shows the real position of a robot moving along a line with constant speed. The red curve shows the position estimated by a discrete Bayes filter using a motion model with a large systematic error. While the low noise in the state transition model leads to a smoothing of the noisy sensor data, it also leads to an accumulation of the systematic error.

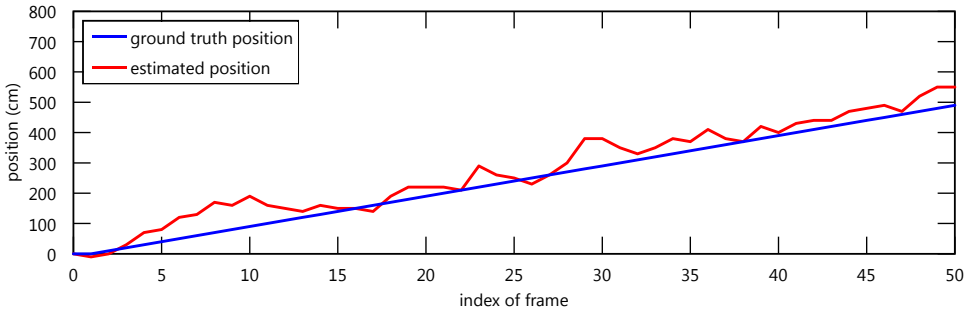


Figure 2.7: *Increased amount of assumed noise reduced the effect of the systematic error.* The blue curve shows the real position of a robot moving along a line with constant speed. The red curve shows the position estimated by a discrete Bayes filter using a motion model with a large systematic error and a large standard deviation. The large standard deviation prevents the systematic error from accumulating. On the other hand, it prevents the incorporation of enough sensor data to smooth the sensor errors, leading to jumps in the estimated position.

state transition models can have large systematic error. The motion model for a legged robot describes how it moves when certain motion commands (e.g. walk forward, make a side step, etc.) are executed. Besides the usual noise, there are reasons for deviations between the command and the result that can not be considered noise but are of systematic nature. Causes for such systematic error can be joint attrition, weak batteries, a different ground structure, etc.

A quick solution, to prevent systematic error from ruining the estimated state, is to increase the assumed noise. Figure 2.7 shows this for the example introduced above. The standard deviation of the motion model used by the discrete Bayes filter was increased to 30 *cm*. The result is, the filter relies more on current measurements, avoiding error accumulation. However, this also leads to significant jumps in the estimated position.

While increasing the assumed noise in the state transition model can help to reduce the effects of unknown systematic error, a better solution is to add a probabilistic model for the systematic error. However, this can only be achieved by adding a dimension to the state space which represents the amount of the systematic error; leading to a higher computational complexity. The *memory-based paradigm*, introduced in chapter 3, provides a means to cope with unknown systematic error without adding computational complexity.

2.4.2 Recovery From Kidnaps

The ability to recover from failure or real sudden state changes is an important filter property. A frequent reason for a sudden robot state change is collision with another robot. While collision usually does not lead to a large position change, the effect on the robot's rotation can be significant. A robot getting stuck while trying to move is another example where its estimated and real state diverge quickly.

What happens when a collision occurs can be illustrated with a modified version of the door

experiment introduced in section 2.2. The difference to the original experiment is that the robot moving to the right, along a hallway, while sensing doors is being teleported to the location with $x = 350\ cm$ as soon as it reaches the location with $x = 250\ cm$. Imagine another robot that accidently crashes into the robot, causing a sudden one-meter long jump. However, the robot affected by that jump has no special sensor to detect it.

Figure 2.8 illustrates that experiment and shows how the discrete Bayes filter responds to that kidnapping. Two noteworthy observations can be made. The first one is that the robot's belief is wrong until it has passed the last door. The second is that the position error (the distance between the maximum of the probability distribution and the real position) is two meters, for most of the time, between the kidnapping and the filter stabilization. So, the error is twice as high as the real position change caused by the collision impact.

This experiment shows that a discrete Bayes filter takes a while to recover from kidnapping. How long a filter needs to recover depends on how much the kidnapping affected the representation quality. One reason for a long recovery time is a high certainty before the kidnapping. Another one is sensors providing only inaccurate information. If kidnapping takes place between two positions that look similar, the recovery also takes longer. Imagine being kidnapped from one floor of an office building to the same position in another floor.

However, a good probabilistic model for such kidnap situations can help to minimize the recovery time. The drawback of describing possible kidnappings, using the state transition model, is that it adds high uncertainty to the posterior distribution. So, for some Bayes filter implementations, the computational complexity increases. For instance, a particle filter needs more particles to represent a more uncertain belief. Note that some Bayes filter implementations can not cope with kidnapping without modifications (Kalman filter, EKF, UKF).

Accurate sensors are a good prerequisite to quickly recovering from kidnapping. They help the filter stabilize as they produce higher certainty in the belief. Section 5.1 shows how accurate percepts can be obtained for a RoboCup scenario.

Another way to cope with kidnapping is to restrict the influence of old measurements on the current belief. Percept selection strategies which pick the best subset of the past observations are part of the *memory-based paradigm*, introduced in chapter 3.

2.4.3 Sparse Information and High Uncertainty

Another important property for state estimators is how they can cope with sparse information and high uncertainty. The state estimation goal is to decrease the uncertainty about the state. Typically, uncertainty is reduced with each sensor measurement and increased with each state transition. The less information provided by a single measurement, the more information has to be accumulated to achieve a specific certainty for the state. Additionally, the uncertainty added by state transitions should not exceed the certainty added by measurements. In this subsection, several classes of sparse measurements are introduced and discussed.

2.4.3.1 Noisy Measurements

The more noisy a measurement, the less information it contains. Imagine a stationary robot. If that robot wanted to find out its distance to a nearby wall using a very noisy distance sensor,

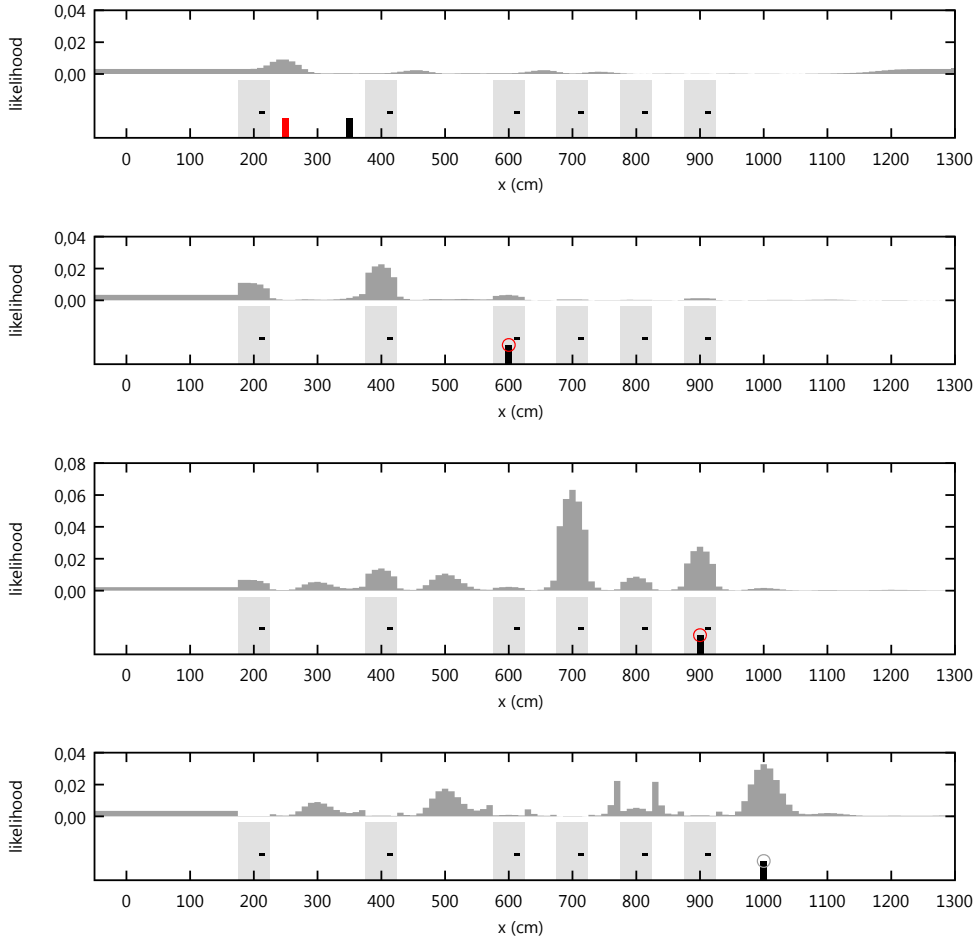


Figure 2.8: *Robot-in-hallway experiment with kidnapping.* The robot is visualized by the black vertical bar at the bottom of the diagrams. A gray circle shows that the door sensor does not detect a door, a red circle symbolizes a door detection. The light gray rectangles are the doors and the dark gray curve shows the belief. The robot starts at $x = 0$ and moves to the right. The diagrams show how the robot's belief changes. The kidnapping is shown in the first diagram where the robot is instantly moved from position $x = 250$ (red bar) to position $x = 350$ (gray bar). The second and third diagram show how the robot's belief does not represent the correct position while it is moving to the right. The last diagram shows that the position of the belief's maximum corresponds to the real position when the robot has passed the last door.

it has to make many measurements. The more noisy the sensor, the more measurements the robot has to make in order to reach a threshold for the probability for the most likely position.

This example illustrates how important it is for a filter to be able to represent sparse information adequately. While parametric filters can represent such sparse information commendably, for example by a Gaussian distribution with a high standard deviation, nonparametric filters usually need many values to represent such distributions.

For the example given above, a Kalman filter provides satisfactory results, independent of the sensor's standard deviation. In contrast, a particle filter needs more particles to be able to process the measurements of a sensor with higher standard deviation. As the robot in the example above is stationary, there is no uncertainty added between two measurements. So, another good way to calculate the robot position is to average all past measurements. This very simple method does not need an internal representation of the current state (like parameters μ and σ for the Kalman filter or a particle set) which is updated in each step. Instead, it simply needs a memory containing all past measurements. The *memory-based paradigm*, introduced in chapter 3, is based on this idea.

2.4.3.2 Measurements Providing Sparse Information

High noise is not the only reason for a sensor to just provide sparse information. The information a given measurement z provides depends on the number of different states in the state space X that lead to the measurement z. The sensor characteristics can be described with a function $f_s(x) = z$ which returns the expected (noise-free) observation z for a given state x. This function's domain is the state space X, the set of all possible measurements Z is its image. As soon as for a given measurement z there is more than one state x with $f_s(x) = z$ (the function f_s is not bijective), this measurement is insufficient to determine the state.

In the remainder of this subsection, different measurement classes are introduced, all of which have the non-bijectivity of the function f_s in common.

Detector Measurements Detector measurements are provided by detection sensors. A sensor which detects whether a robot is in front of a door or not was introduced above. A sensor mounted on an airplane detecting whether it is above land or water is another example. Similarly, a sensor with a defined angle of view is thinkable, that is, a sensor that detects whether a landmark is within a certain angle of view. While for all those measurements the corresponding function f_s is a boolean function, there are also sensors that map from states in the state space to elements of a classification set. Imagine a sensor that detects whether a robot is next to a red, green, or yellow wall.

How much information is contained in such a detector's measurement depends on the world setup. In a long hallway with only a few doors, a door detection provides more information than a no-door detection of a door. That's why measurements that signal the absence of a certain feature are sometimes called *negative information*. However, the example with the sensor which detects whether an airplane is above water or land shows that the amount of information provided by a detector's measurement depends on what the world looks like. The fact worth noting here is that in most cases detector measurements provide only sparse information, as the more information is contained in a measurement, the less likely it is to make it.

The following example shows how the accumulation of sparse information can be used to resolve a multivariate measurement. In this example, a robot moves down a hallway which contains two doors, the distance between the doors is 2 meters. If the robot moves more than 2 meters and then detects a door, it can conclude that it must be in front of the first door. This "conclusion" can be made by a Bayes filter.

The difficulty here is that the sparse information, of not seeing a door, has to be represented by the belief probability function. Many parametric filters are not suitable to represent this special kind of posterior distributions. Parametric filters need a high resolution to represent such sparse information. A particle filter, for example, needs enough particles to represent the sparse information provided by not seeing the doors.

The *memory-based paradigm*, introduced in chapter 3, does not rely on belief representations and thusly evades the problem illustrated above.

Distance Measurements Distance measurements provide the distance to landmarks. There are two main forms of distance sensors. The first form provides the distance to a point in space. A measurement of such a sensor constrains the possible positions to a circle or a sphere. The second form measures the distance to a line in space, for example the distance to a wall or, in RoboCup, the distance to a field line. Such a measurement constrains the robot's position to a single line, or two lines when the landmark can be observed from two different sides.

Imagine a typical localization scenario, where the x-coordinate, the y-coordinate, and the rotation of a robot are to be determined. Then, two (perfect) distance measurements to two different, unique, and distinguishable landmarks restrict the possible locations to two points, the intersecting points of the resulting circles. A special case occurs, when the robot position is somewhere on the line between the two landmarks, then the resulting circles touch at a single point. With a distance measurement to a third landmark, the position can be determined exactly. However, the robot's rotation can not be determined just using distance measurements.

Two (perfect) distance measurements to two different, unique, and distinguishable lines are sufficient to determine the robot's position when these lines are not parallel and observable from only one side, for example: distance measurements to two different walls in a room. However, field lines in RoboCup are indistinguishable and observable from both sides. The observation of two field lines constrains the position to many symmetrically arranged points. Again, the robot's rotation can not be determined using just distance measurement to lines.

How distance measurements can be processed following the *memory-based paradigm* is described in 3.3.3.

Bearing-Only Measurements Bearings are another example for sparse measurements. A bearing sensor renders the bearing to a landmark. For a three-dimensional state space (x-y-position and heading), the information provided by a bearing does not constrain the robot's position. However, for a given position (x, y) it constrains the robot's heading α. The sensor can be described by the equation $z = \arctan(y_l - y, x_l - x) - \alpha$ where z is the resulting measurement for a given state described by the coordinates (x, y) and the heading α. The landmark's position is given by (x_l, y_l).

The difficulties particle and Kalman filters have with integrating bearing measurements are a

big motivation for the *memory-based paradigm*. Chapters 4 and 6 introduce robot localization methods which use horizontal bearings to landmarks.

3 The Memory-Based Paradigm

This chapter introduces the *memory-based paradigm* which is both a supplementation and an alternative to the probabilistic paradigm. While methods based on the probabilistic paradigm are very powerful and have led to a large number of impressive robotic applications, they have to be used cautiously in many situations.

For a given state estimation problem, good reasons to choose a Bayes filter variant, especially a particle filter, are that such filters are easy to implement and produce good results. These properties of probabilistic methods involve a certain danger of abusing the parameters of the methods. Classic examples are: increasing the number of particles when the estimation results differ from the expectations, adding motion or sensor noise when the filter is not reactive enough, or reducing the motion noise when the results are too unsteady. While the use of probabilistic methods is often appropriate, their nature sometimes seduces into treating the symptoms instead of the cause when problems arise.

There are also some cases where the use of probabilistic methods is not advisable. For example, when the probabilistic models that describe a system are not known or require a too high dimensional state space. Some limitations of the Bayes filter and its applications were already discussed in section 2.4.

The *memory-based paradigm* is an approach to state estimation which can better cope with the issues brought up above. My main criticism of the methods based on the Bayes filter is that the sole place for a robot to store and accumulate information is the belief, which follows from the Markov assumption and the notion of the *complete state*. This can lead to problems when the information is sparse, kidnapping occurs or there is systematic error. In this chapter, the *memory-based paradigm* is introduced, which provides new methods for state estimation.

Section 3.1 shows state estimation problems which can be approached using memory-based techniques. In section 3.2, the motivation for the introduction of the *memory-based paradigm* is given. Section 3.3 provides the definition of *memory-based state estimation (MBSE)* and possible algorithmic variants. In section 3.4, experiments and their results are given, which validate the concept. Sections 3.3 and 3.4 are based on [43] and [46]. Parts of [46] were created with Heinrich Mellmann.

3.1 Introductory Examples

In this section, several examples of simple state estimation problems are given that motivate the introduction of memory-based methods.

3.1.1 Navigation On Sea Using Nautical Charts

The art of plotting navigation courses on nautical charts can be seen as one of the oldest state estimation methods. Even in the times of GPS, it is more than useful to examine ancient navigation techniques. In this subsection, the most important means of plotting and piloting on nautical charts are introduced.

The navigational goal at sea is to determine the position and heading of ones ship on a map. For this purpose several navigational instruments can be used. The most important is a compass, which gives the direction to the magnetic north pole. A compass can also be used to measure horizontal bearings to landmarks like lighthouses, steeples, buoys, or windmills. There are a lot of navigational aids, like lighthouses or radio navigation systems, which provide bearings or distances, when appropriate instruments like radio receivers ore binoculars are on board. There is also a large number of instruments that simplify the analysis of natural landmarks like stars or coast lines; the Sextant is the most prominent.

The speed of a vessel, relative to the surrounding water, can be measured using a chip log and a stopwatch. In the early days, usually an hour-glass was used as a stopwatch. Chip logs usually consisted of the log-line wound on a reel and a wooden board weighted with lead. The log-line was knotted with a defined and uniform spacing. Modern instruments for speed measurement usually use impellers or Doppler Sonars. Sonars are active acoustic locators; a depth sounder is one of the most common instruments in this category. It uses the knowledge about the speed of sound in water to determine the depth, using the time between a sent signal and its echo caused by reflection at the sea floor.

The navigational instruments mentioned can be affected by considerable errors. Compasses can be influenced by the ship itself, by other vessels, or by the earth's magnetic field's natural fluctuations. Speed measurements can be disturbed by wrong distance measurements (currents, vegetation, etc.) and by wrong time measurements (inaccurate clocks). Measurements taken by instruments which rely on the sound speed in water can be affected by the water's temperature, pressure, or salinity. However, with additional measurements or correction tables, the influence of such errors can be minimized.

With the instruments described above, it is possible to track a ship's position, when a nautical chart is available. In the remainder of this section, the basic concepts of plotting courses on nautical charts are introduced. This technique is well-documented; good introductions are given in [82], [67], and [56]. Based on this, design goals for the *memory-based paradigm* will be introduced in 3.2.3.

3.1.1.1 Lines of Position

A *line of position (LOP)* constrains the possible position of a ship to a single line. An LOP can be drawn when a compass measurement (ship's angle relative to north) and a bearing to a landmark of known-position (ship's angle relative to a landmark) are available. Figure 3.1 gives an example.

An LOP is drawn as an arrow pointing to the landmark. It is labeled with the time of the observation and the angle to north. An LOP can be obtained without a compass when an observed sector light with known position changes its color while passing it. This also requires

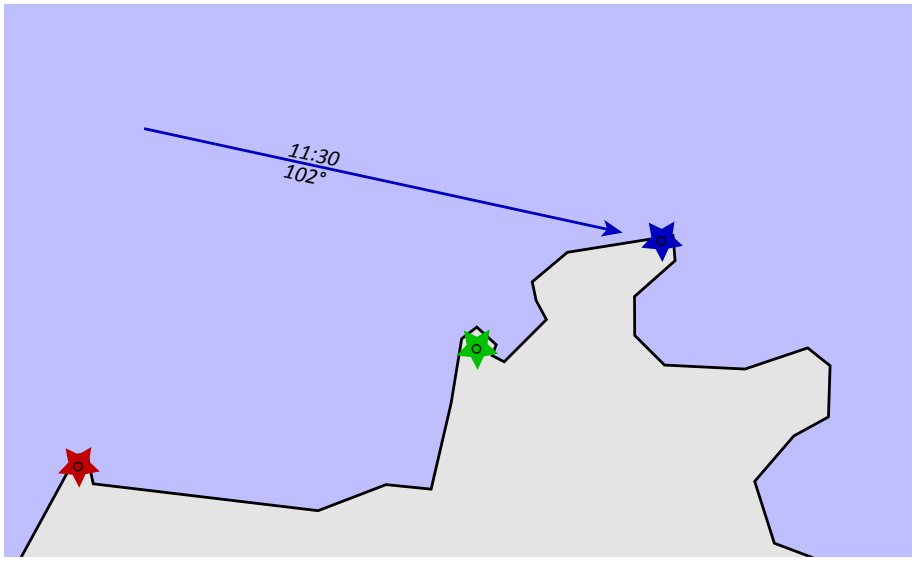


Figure 3.1: *Line of position (LOP).* The shown LOP was obtained by a bearing to the blue lighthouse and a compass measurement which was already corrected (due to the deviation between the magnetic and the true north). An LOP should be labeled with the time when the observation was made and its bearing (the angle to true north).

that the bearings of the sectors are known.

Another way to obtain an LOP without a compass is the usage of a range. A range is an LOP defined by characteristic points with known position. When such two landmarks are observed aligned (both points have the same bearing), the ship is on the line defined by the position of these two points. Such characteristic points can be landmarks, points resulting from alternating convex and concave parts of a coastline, or points defined by tangents to islands. The higher the distance between the two points that define a range, the higher the accuracy of the LOP. Additionally, the distance to the closer landmark should not be too high.

3.1.1.2 Cross Bearings

Cross bearings are the most common way to obtain a *position fix*. A position fix gives the ship position. A cross bearing can be obtained when two different landmarks are visible at the same time or at nearly the same time. The faster the ship moves, the lower the time difference between the two observations should be. The position is given by the point where the two LOPs intersect. Figure 3.2 gives an example.

To get the most accurate results, the two bearings should intersect at an angle close to 90°. When three observations can be made at the same time, the resulting LOPs usually intersect in three different points. In such a case, the triangle's center is used for the position fix. To take possible errors into account, instead of a single point, a position area can be constructed. This is done by replacing the LOPs by corresponding sectors. The area where the sectors, created

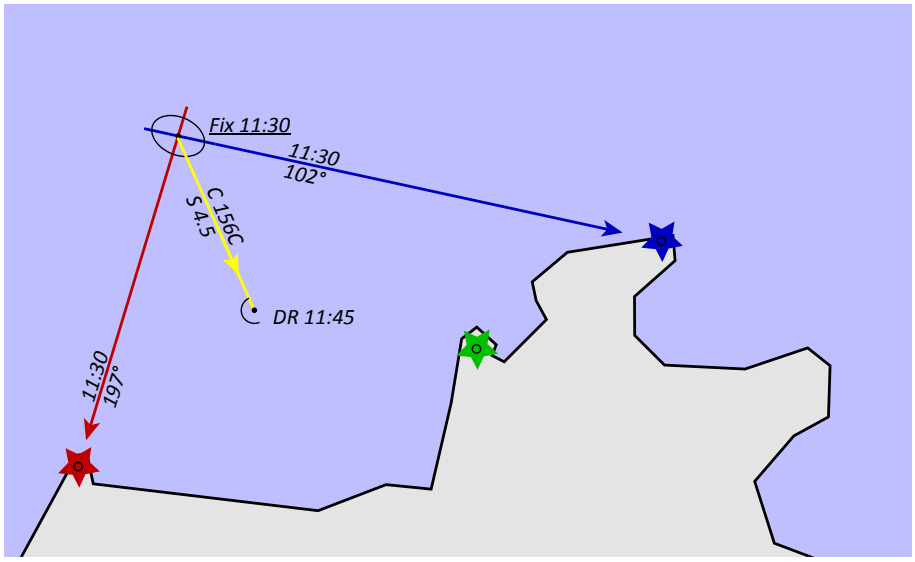


Figure 3.2: *Position fix and dead reckoning.* A position fix can be created when two LOPs can be obtained at the same time. A position fix is marked with an ellipse, the label "Fix" and the relevant time. The label and time are underlined. The closer the angle between the lines to 90°, the more accurate the position fix. Dead reckoning lines are always drawn from the last fix. They are labeled with the ship's speed and heading. When the course or the speed changes or when an observation is made, a new dead reckoning line is drawn. The ends of dead reckoning lines are marked with "DR" and the relevant time.

by different observations, overlap is the position area. Constructing such areas can be helpful when hazardous areas have to be avoided.

3.1.1.3 Dead Reckoning

When no observations can be made, the vessel position can be tracked using emphdead reckoning. This is possible using knowledge about the ship's heading and speed. Each time the ship changes its course or its speed, a new line is drawn on the chart that represents the movement since the last change. Additionally, a new line is drawn when an observation is made. This is important for the construction of running fixes which are introduced below. Such dead reckoning lines are labeled with the relevant speed and heading. The end of such lines is marked with a semi-circle, the label "DR" and the time. Figure 3.2 illustrates this.

A path of dead reckoning lines is always started at the last position fix. It is obvious that this technique is prone to the accumulation of systematic error. However, the more known about possible influences like currents or the wind speed, the more precise the results.

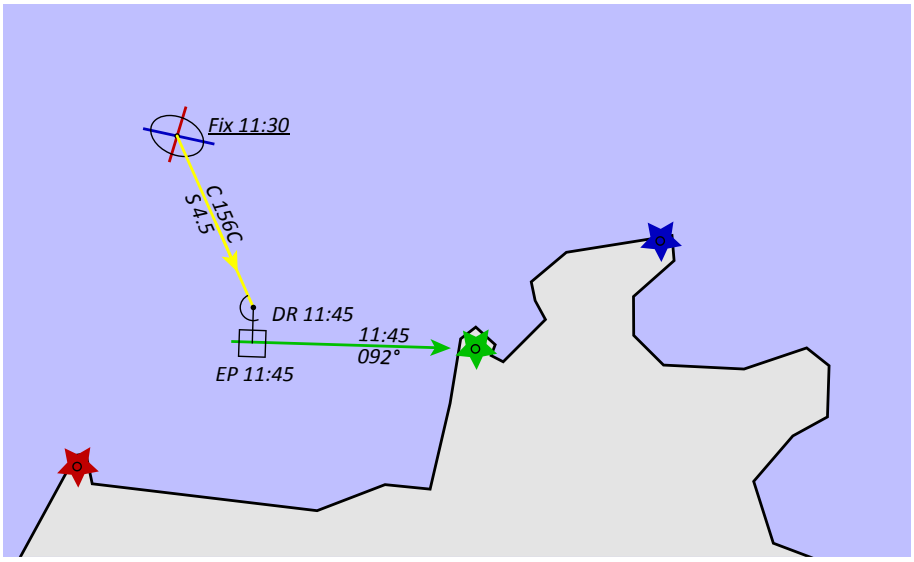


Figure 3.3: *Estimated position.* A single observation allows the construction of an estimated position when the dead reckoning path since the last position fix is known. The estimated position is the position on the LOP obtained by the observation being closest to the according dead reckoning position. The symbol for an estimated position is a square labeled with "EP" and the time.

3.1.1.4 Estimated Positions

When only a single observation is made, leading to just one LOP, this can be used to obtain an *estimated position*. This is done using the dead reckoning position of the observation time. The estimated position is the point on the LOP resulting from the observation closest to the dead reckoning position. That point is constructed by intersecting the LOP with a perpendicular line that goes through the estimated position. An estimated position is marked with a square on the chart, the letters "EP" and the time. Figure 3.3 gives an example.

Note that dead reckoning lines obtained after the construction of an estimated position are drawn starting from the last dead reckoning position; dead reckoning paths always start at the last position fix. Estimated positions are not used as a starting point for new dead reckoning lines. This is important for the construction of running fixes, which are introduced in the next section.

3.1.1.5 Running Fixes

When two observations are made at two different times, a running fix can be created. This can even be done using a single landmark. As with the construction of a position fix, the most accurate results are achieved when the two LOPs intersect at an 90° angle. In order to construct a running fix based on two LOPs constructed from two observations, the respective dead reckoning positions are needed. Figure 3.4 shows a scenario where two LOPs and a dead

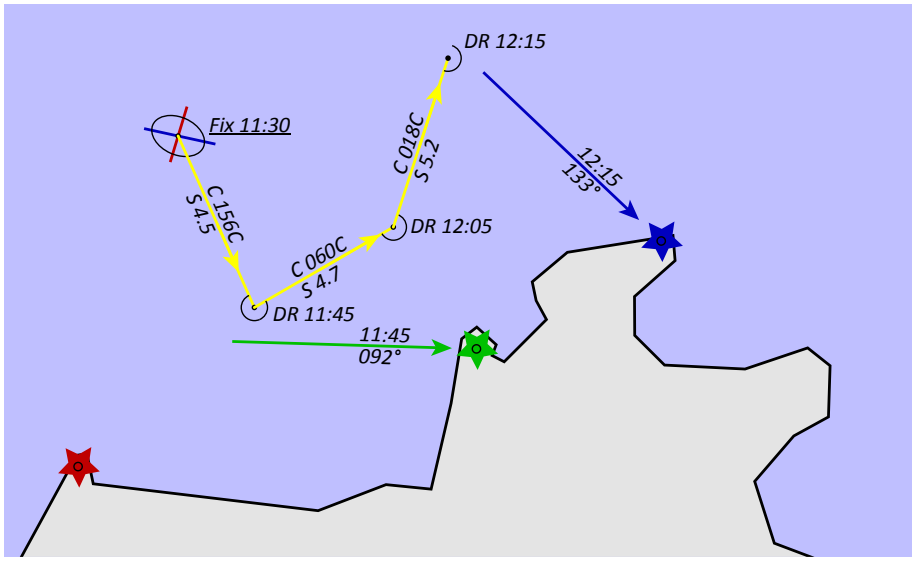


Figure 3.4: *Running fix.* Two observations at two different times can be used to construct a running fix. This can be done when there are two LOPs and a corresponding dead reckoning path. The construction of the running fix for this scenario is shown in figure 3.5.

reckoning path, starting at the last fix, are drawn on the chart.

Based on the dead reckoning positions, the complete distance and direction the ship moved in between the observations can be constructed. This is done by connecting the dead reckoning position which belongs to the first observation with that of the last observation by a thin line. Then the first observation's LOP is advanced to the time of the second observation. To do this, a construction line is drawn which starts somewhere on the first LOP and is parallel to the line which connects the two dead reckoning positions and has the same length. The advanced LOP is drawn parallel to the original one and through that construction line's end. Figure 3.5 illustrates this.

Note that the position of the fix where the dead reckoning path starts has no influence on the position of the running fix. Even the error in the dead reckoning which occurs before the first observation has no effect on the position of the running fix, as just the dead reckoning difference between the two observations is used. This is a very important property of a running fix. The construction of a running fix is a good way to recover from errors in the dead reckoning position or in estimated positions.

3.1.1.6 Scale of Reliability

The different position estimates introduced can be sorted based on their reliability leading to this order, starting with the most reliable:

- Position fix

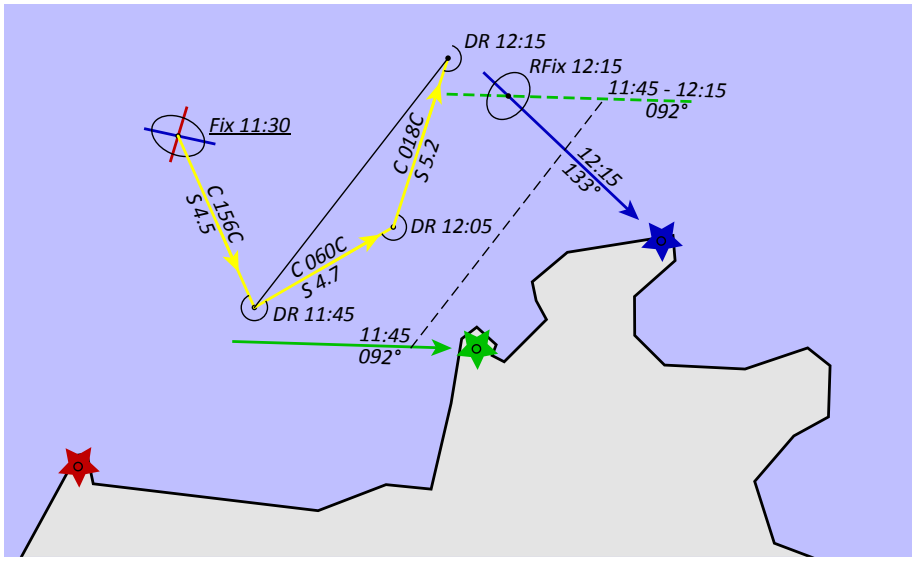


Figure 3.5: *Running fix.* A running fix for two LOPs that where obtained at different times is constructed by advancing the first LOP. The green LOP is moved by the difference between the positions *DR 12:15* and *DR 11:45*. That moved line (the dotted green line) is intersected with the blue LOP to obtain the running fix. A running fix is labeled with "RFix" and the corresponding time.

- Running fix
- Estimated position
- Dead reckoning position

This order is given in [82] and is quite obvious. A position fix uses just current observations and does not include any (unprecise) dead reckoning information. For a running fix, dead reckoning information is used but just starting at the second-to-the-last observation. One observation used for a running fix is current, the other can have a certain age. An estimated position uses one current observation and can be influenced by a large dead reckoning error, as the dead reckoning information is used starting at the last fix, which can be old. However, it is still better than just a dead reckoning position which uses no current observations.

The scale of reliability given above is a strong motivation for the introduction of the *memory-based paradigm*. A basis of the piloting technique introduced above is to rely on current information as much as possible. Another principle is to draw all information available on the chart such that it can be used later, when necessary. Memorizing all actions and observations is a principle of the *memory-based paradigm* as well.

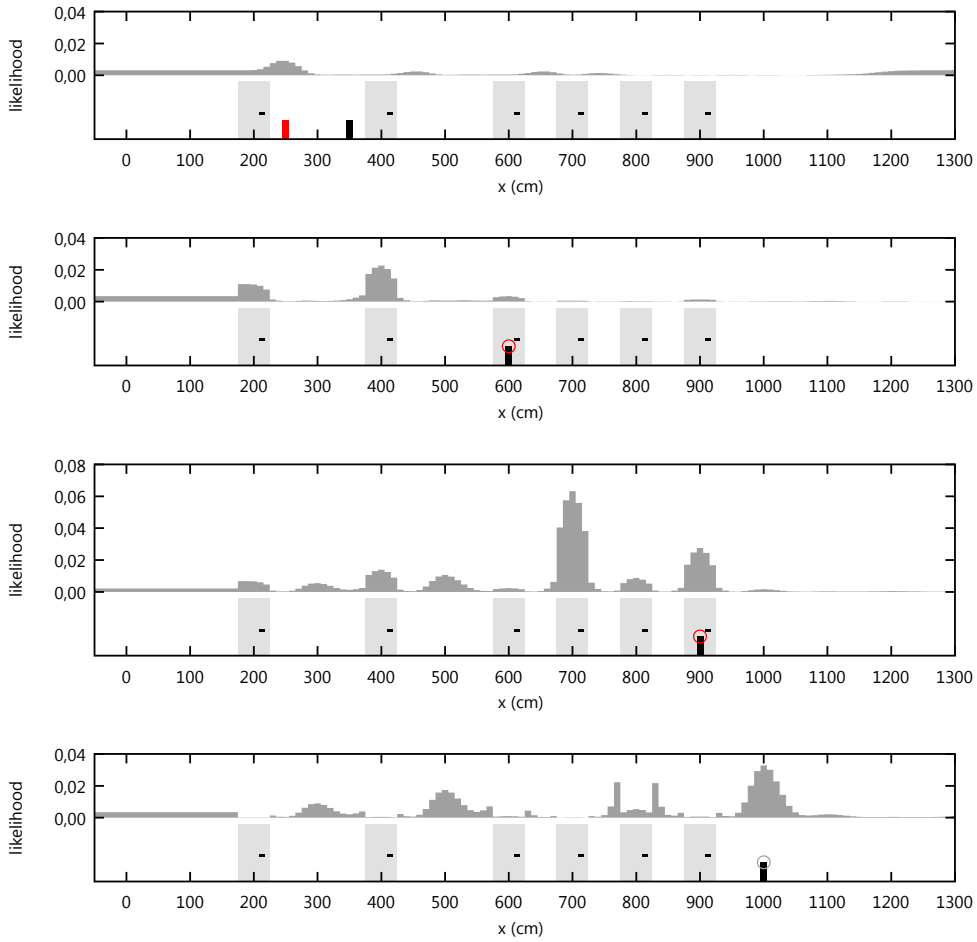


Figure 3.6: *Robot-in-hallway-experiment with kidnapping.* Figure 2.8 is repeated here to simplify comparison to figure 3.7 on the opposite page. The kidnapping is shown in the first diagram, the robot is instantly moved from position $x = 250$ (red bar) to position $x = 350$ (gray bar). The second and third diagram show how the robot's belief does not represent the correct position while it is moving to the right. The last diagram shows that the position of the belief's maximum corresponds to the real position when the robot has passed the last door.

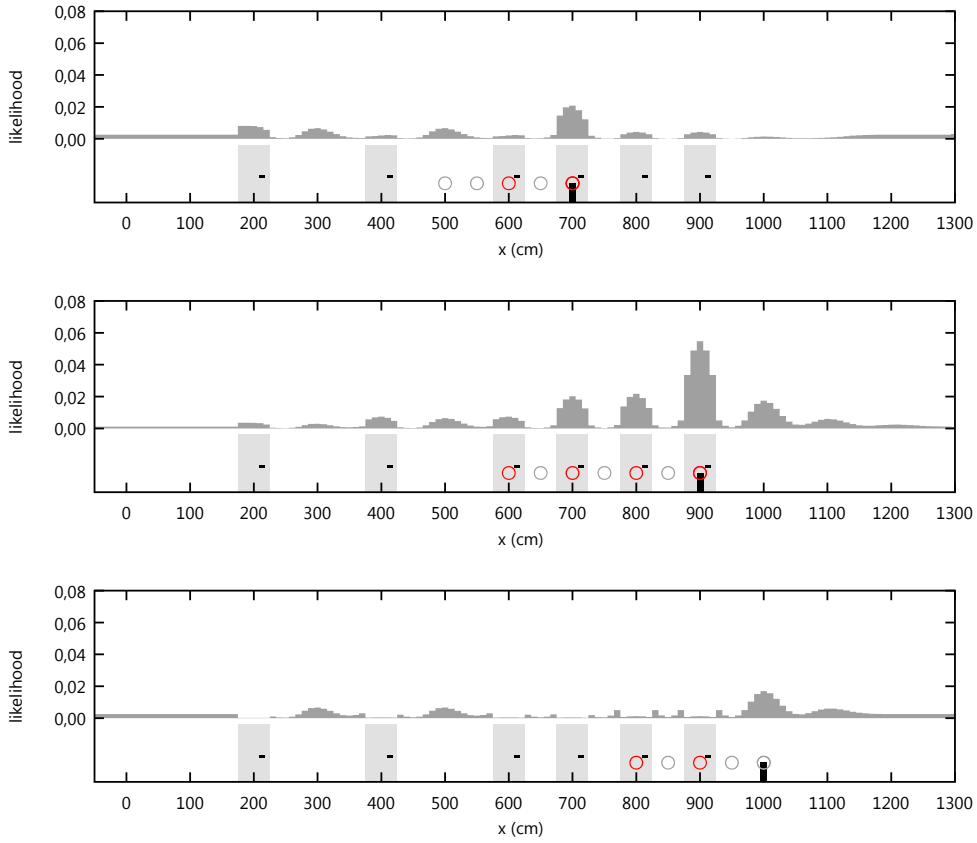


Figure 3.7: *Robot-in-hallway-experiment with kidnapping.* For these diagrams the belief was not calculated recursively from one step to the next. Instead, it was calculated starting with an equal distribution and iterative for the steps highlighted by circles. Note that this increases the calculation time, as instead of one update per step, a number of updates has to be performed. *First diagram:* The position $x = 700$ is the only one where the observations sequence shown (no door, no door, door, no door, door) can end when the robot is moving right. This is represented by the belief. From this position on, the robot's belief has recovered from kidnapping. *Second diagram:* Only at position $x = 900$, a sequence of four doors observed near-by can end. *Third diagram:* An observation sequence (door, no door, door, no door, no door) can only end at position x = 1000.

3.1.2 Kidnapping in a Hallway

In section 2.2, the Bayes filter was described using the classic example of a robot which tries to localize itself in a hallway. Section 2.4.2 used that example to illustrate how a Bayes filter performs after kidnapping. The result of the experiment with the kidnapping was that the robot's belief, after the kidnapping, was wrong and needed some time to recover. Figure 3.6 shows this experiment again. The time to recover can be reduced by adopting the state transition model in a way which represents the probability to be kidnapped. However, it can be difficult to obtain such a model.

The concepts *position fix* and *running fix*, introduced in the last section, provide inspiration for another solution. While there is no counterpart to a position fix for the robot-in-hallway-scenario, as the robot's position in the hallway can not be determined when the robot is not moving, there is an analogy to the running fix. However, to obtain a running fix in a hallway typically more than two observations are needed. It is worth noting again that, for a running fix, only a limited set of current measurements and dead-reckoning information is needed. Previous data can be considered obsolete. Consequently, a running fix is perfectly suited to recover from kidnapping. While in the nautical chart example two bearings are needed for a running fix, the number of measurements needed in the hallway scenario is not fixed. Even if it is assumed that there are no measurement errors, the number of observations needed can vary. How many last observations are needed, depends on the map and the last observations.

Assume all door measurements and actions of a robot moving down a hallway are recorded. Then two questions are crucial to calculate a running fix: How many past measurements are needed and how to calculate the position, based on these measurements. Assume some unspecified algorithm picks the minimum number of measurements needed from the memory. Then, a Bayes-like algorithm can calculate a belief, based on these observations and the actions performed in-between. This method can be understood as a sliding-window approach, with a flexible window size. Figure 3.7 shows how this works, for the data used before. With this method of calculating the position just based on the minimal number of needed observations, the position recovers faster after the kidnapping. However, the calculation time is longer than when a regular Bayes filter is used. In each step, the Bayesian update has to be done once for each of the selected observations, instead of just once.

3.1.3 Propagation of Systematic Error

In section 2.4.1, it was shown how an inaccurate state transition model can lead to the accumulation of systematic error, when a Bayes filter or one of its variants is used. The example used there was a robot equipped with a sensor, measuring the distance to a fixed point on the line it moved along with constant speed. Figure 3.8a) shows, again, how a Bayes filter accumulates the systematic error.

When no information about the amount of the systematic error is available, the accumulation of systematic error can be avoided by increasing the assumed value of noise in the state transition model. This is a way to make the system forget older measurement more quickly. However, as such a modified model does not represent the true effect of motion commands, it leads to sudden jumps in the estimated position. Figure 3.8b) illustrates this.

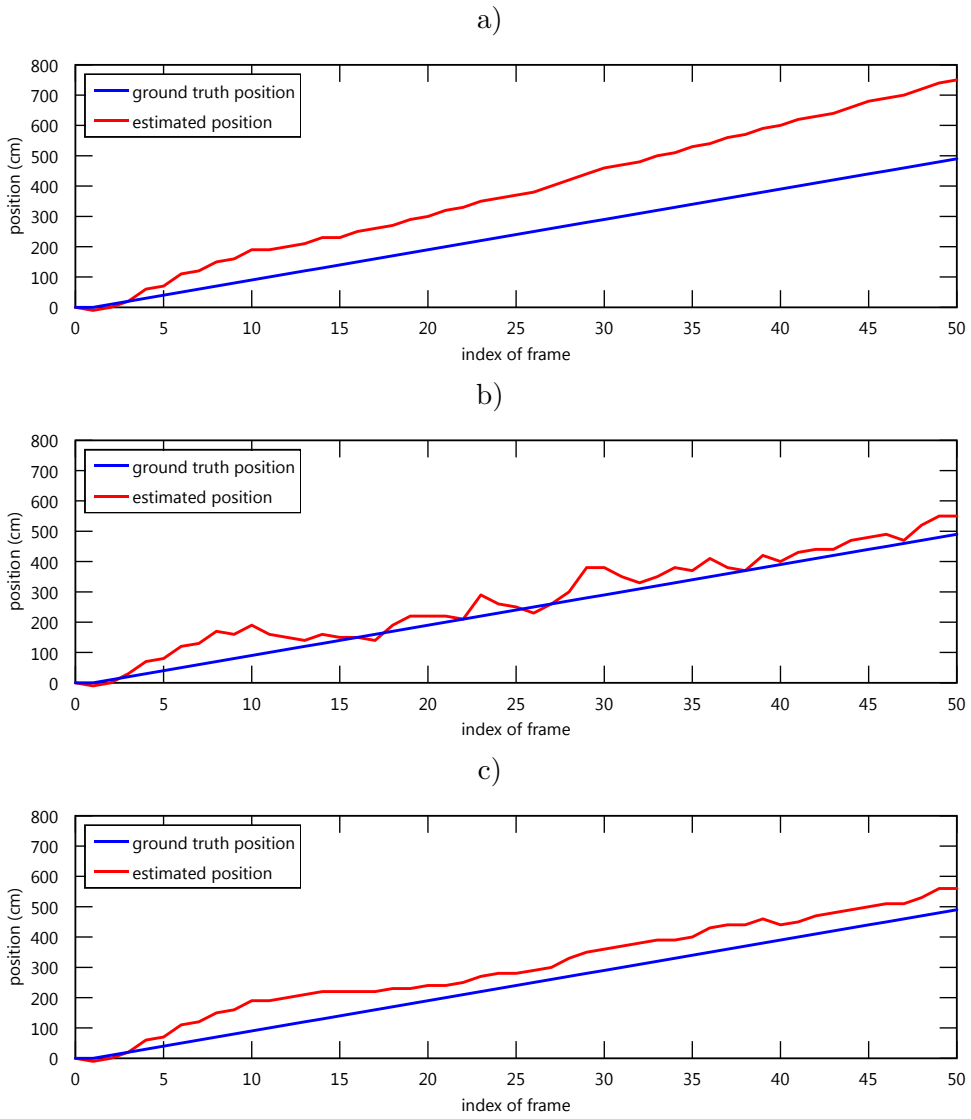


Figure 3.8: *Coping with systematic error.* The blue curves show the real position of a robot moving at constant speed. a) The red curve shows the position estimated by a Bayes filter, using a motion model with a large systematic error. b) The amount of assumed noise was increased to reduce the effect of systematic error. However, this prevents the incorporation of enough sensor data in order to smooth sensor error, leading to the jumps in the estimated position. c) In each step, only the last 10 measurements were used to calculate the position. So, the systematic error is not accumulated, while the curve is still smooth.

Unknown systematic error in the motion model is a widespread problem for autonomous robots. Imagine a couple of the same type robots, all in a slightly different physical condition because of wear and tear. Each of these robots would show a slightly different reaction when executing the same motion command. An appropriate probabilistic modeling would require to measure the systematic error for each robot. Another way would be to add a dimension to the state space which describes the systematic error. These variants require additional calibration efforts or a significant calculation time increase.

Again, introducing a concept similar to a running fix can solve this problem. The basic idea here is to consider only a limited set of measurements for position estimation. This evades the accumulation of systematic error, as older measurements are not considered, while the original models for the state transitions and for the sensor can still be used, leading to a smooth model with minor systematic error. Figure 3.8c) shows this for the distance sensor example.

3.2 Motivation

In section 2.1, the concept of sequential state estimation was introduced. The foundation for this state estimation variant are Hidden Markov Models. In this section, what distinguishes *memory-based state estimation* from the well-known approach based on Hidden Markov Models is described.

For the examples given in the previous section, how different state estimation methods perform is presented. Different ways to accumulate information are given in 3.2.1; while, the discussion is in 3.2.2. Based on this discussion, I state my design goals for *memory-based state estimation* in section 3.2.3.

3.2.1 Different Ways to Accumulate Information

This section presents different ways to accumulate information. Different state estimation forms can be seen as different ways to accumulate information.

3.2.1.1 Direct State Estimation

The purpose of state estimation is to find the best approximation for the state of a system. Commonly, this state can not be observed directly. A classic state estimation example is self-localization. When a perfect position sensor is available, there is nothing left to be done. More generally, there is no need for sophisticated state estimation technique when there are sensors which provide at least as much information as needed to determine the position in every time-step. Formally, such *Direct State Estimation* can be seen as a function f_D which calculates the current state $\vec{x_t}$ at time t based on the vector $\vec{z_t}$ of observations at time t:

$$\vec{x_t} := f_D(\vec{z_t}) \tag{3.1}$$

This direct form of state estimation does not use the knowledge about the actions u_t which led the system from state x_{t-1} to state x_t. *Direct State Estimation* is not applicable when there are less sensor readings available than needed to determine the state directly. Looking at a

thermometer just once to estimate the temperature in a room is an example of *Direct State Estimation*.

3.2.1.2 Hidden Markov Models for State Estimation

As soon as the information obtained at a single moment is not sufficient to determine the state, information has to be accumulated over time. This is done in the Hidden Markov Model approach for state estimation; a detailed description was given in section 2.1. In this approach, the current state is represented as a probability function $bel(x_t)$ which is called the *belief* of x_t. The current state is calculated by a function f_{HMM} usually realized by a Bayes filter variant (cf. 2.2.5), using the last state's belief $bel(\vec{x_{t-1}})$, the current measurement $\vec{z_t}$, and the action $\vec{u_t}$ executed at the transition from $\vec{x_{t-1}}$ to $\vec{x_t}$:

$$\vec{x_t} := f_{HMM}(bel(\vec{x_{t-1}}), \vec{z_t}, \vec{u_t}) \tag{3.2}$$

Bayesian filters can use probabilistic sensor and motion models and accumulate information over time. Following the Markov Assumption, this information aggregation can be done from state to state, as long as the current state contains enough information. So, it is often referred to as recursive state estimation.

3.2.1.3 Memory-Based State Estimation

If the assumption to have a representation of a complete state in each time-step is given up, a function f_M can be defined which calculates the current state x_t based on the current and all past observations $\vec{z_{0:t}}$ and all past control actions $\vec{u_{0:t}}$:

$$\vec{x_t} := f_M(\vec{z_{0:t}}, \vec{u_{0:t}}) \tag{3.3}$$

I call such state estimation variants *memory-based state estimation (MBSE)*. Strictly speaking, *Direct State Estimation* is a variant of *MBSE* which only uses the last observation $\vec{z_t}$. *Hidden Markov Model* based state estimation can also be seen to be a variant of *MBSE*. In this case, the function f_M has to calculate all recursion steps of function f_{HMM} in each time-step (which would break the recursion idea). There are many existing approaches which extend *HMM*-based localization methods using memory-based techniques: [50, 74, 18, 72].

The concept of *MBSE*, as introduced, is a simple prototype; a detailed introduction is given in section 3.3.

3.2.2 Discussion of Introductory Examples

In section 3.1.1.5 it was described, what a *running fix* is: a way to use a past and a current observation (landmark bearing), with accumulated dead reckoning information to obtain a position estimate. How does this relate to the three ways to accumulate information that were introduced in the previous section?

The calculation of a running fix is not possible via *Direct State Estimation* as information has to be accumulated over time. As defined in (3.1), with *Direct State Estimation* the state x_t is calculated based on a function that only uses the last observation: $\vec{x_t} := f_D(\vec{z_t})$. No old state or old observations are available.

The position construction using simple geometry, falls into the class of *memory-based state estimation*. To draw the LOP that is propagated depending on the dead reckoning path, the LOP obtained at the first observation time is needed. In this case, the nautical chart is the memory that stores the vector of all observations $z_{0:t}^{\vec{}}$. The vector of control actions $u_{0:t}^{\vec{}}$ is also stored in the chart, in the form of dead reckoning paths. The propagated and the current LOP intersection can be calculated using a function as introduced in (3.3): $\vec{x_t} = f_M(z_{0:t}^{\vec{}}, u_{0:t}^{\vec{}})$.

However, it is also possible to determine the position based on the observation sequence and the dead reckoning measurements using an approach based on Hidden Markov Models. Three prerequisites are needed: First, a probabilistic sensor model, which gives the expected compass and bearing measurements for each possible position; second, a probabilistic state transition model, which describes the probability of moving from one position to another, given a certain action (which can be derived from the measurement of moved distance); and third, a way to represent the current belief, which depends on the Bayes filter variant chosen. With these prerequisites, the position estimation can be done, starting from an equal distribution for the belief, sequentially adding the bearing observations and the dead reckoning.

While the geometry-based approach is simple and straightforward, the HMM-based approach is more general. For a lot of state estimation problems, it is easier to provide a sensor and a motion model than to pursue geometric considerations, which can get complicated when contradictory information has to be processed. The HMM-approach can cope with that without additional efforts. In the example of calculating a running fix with an HMM-based approach, more measurements can be integrated very easily; while for the geometric approach rules are needed how to proceed when the LOPs don't intersect at a single point.

On the other hand, the HMM-approach needs a complete representation of the current state. This has to be a probability density function over the state space in order to be able to accumulate information over time. While a complete discrete representation leads to high memory usage and high computational costs, there are several solutions to reduce this complexity. Particle filters approximate the function using a particle set; Kalman filters assume the function to be a multivariate normal distribution. However, these approximations have their weaknesses. For example, a very high particle number is needed when the information that has to be integrated is sparse (does not provide a lot of information at once).

The principle of *MBSE* is to combine the advantages from the HMM-approach with those of approaches based on geometric considerations. It should be as generic as the HMM-approach: The sensor and the motion model are the only environmental information needed. It should be as light-weight as the geometry-based approach: no internal representation of the state should be needed to calculate the state. More detailed design goals are given in the next section.

3.2.3 Design Goals

The goal of *memory-based state estimation* is to provide means for state estimation that exceed the ones given by existing approaches. The motivation stems from weaknesses of existing meth-

ods, especially methods based on Bayesian filtering. In section 2.4, the weaknesses of the Bayes filter applications were discussed. In sections 3.1 and 3.2.2, some ideas to cope with these issues were given. In this section, the design goals for the *memory-based paradigm* are formulated which are based on those ideas and on the properties of Bayesian methods.

As *MBSE* should be an improvement, compared to state estimation based on Bayesian filtering, it should have the merits of most of those methods. I regard the main properties of methods based on Bayesian filtering as:

- *Generality:* The formulation of the algorithms is not specific to a domain or a state space. The only domain specific things are the sensor model, the state-transition model, and the state space structure. These models provide the description of the respective system.

- *Usage of forward models:* To estimate a system's state, a description is needed that provides information on the connection between the system state, observations within the system, and possible influences to the system. The nature of Bayesian filters allows this system description to be given using forward models: the sensor and the motion model. A big advantage of this is that forward models are usually easy to obtain.

An important design goal for *MBSE* is that it has the properties listed above. Additionally, *MBSE* should have these properties:

- *Capability to cope with systematic error*: The system should be able to cope with unknown systematic error. In section 2.4.1, it was shown how the results of Bayesian filtering could be affected by systematic error.

- *Capability to quickly recover from kidnapping*: If information is stored recursively in a *belief*, it can take a while until the system can recover from kidnapping. This problem was discussed in section 2.4.2.

- *Capability to process sparse data*: One of the main disadvantages of accumulating sparse information using probability functions over a state space is that a high resolution is needed, which leads to high computational costs (cf. section 2.4.3).

3.3 Memory-Based State Estimation

This section describes the main principles and the basic mathematic foundations of *MBSE*.

Sections 3.3.1 and 3.3.2 introduce the formalities needed to describe the system and justify it, section 3.3.3 describes the method itself. In section 3.3.4, a proof is given showing that the method delivers the same results as a Bayes filter when certain assumptions hold. In section 3.3.5, some algorithmic implementations are presented.

3.3.1 Definitions

This section presents some prerequisite terminology.

Definition 3. (Observation function)
An observation function f_s gives the expected observation z for each state x:

$$f_s : X \longrightarrow Z, x \longmapsto f_s(x),$$

where X is the set of all possible states, Z the set of all possible observations, and $z = f_s(x)$.

If a probabilistic sensor model $p(z|x)$ (cf. 2.2.3) is given, the corresponding *observation function* is

$$f_s(x) = \arg\max_z p(z|x).$$

Definition 4. (Control function)
A control function f_c describes the new state x_i when the previous state was x_{i-1} and the action u_i was performed:

$$f_c : X, U \longrightarrow X, x, u \longmapsto f_c(x, u),$$

where X is the set of all possible states, U the set of all possible actions, and $x_i = f_c(x_{i-1}, u)$.

If a probabilistic state transition model $p(x_i|x_{i-1}, u)$ (cf. 2.2.4) is given, the corresponding *control function* is

$$f_c(x_{i-1}, u) = \arg\max_{x_i} p(x_i|, x_{i-1}, u).$$

Note that the control function does not model the sensor's probabilistic properties. In the same way, only a deterministic effect of actions can be described by the control function.

Definition 5. (Reverse control function)
A control function's reverse $f_c^*(x, u)$ is defined with

$$f_c^* : X, U \longrightarrow X, x, u \longmapsto f_c^*(x, u),$$

such that $f_c^*(f_c(x, u), u) = x$. For a given state x_i and an action u, the reverse function calculates the state x_{i-1} before the action execution: $x_{i-1} = f_c^*(x_i, u)$.

Definition 6. (Concatenated Controls) *The concatenation of two actions $u_i u_j$ is defined such that*

$$f_c(x, u_i u_j) = f_c(f_c(x, u_i), u_j).$$

Note that concatenation usually is not commutative.

Proposition 7. *The reverse of concatenated actions equals the recursive reversion of the single actions: $f_c^*(x, u_i u_j) = f_c^*(f_c^*(x, u_j), u_i)$.*

Definition 8. (Accumulated Controls)
For the sake of simplicity, the symbol v_m is introduced to describe the concatenation of the last m actions:

$$v_m := u_{n-m+1}, ..., u_n.$$

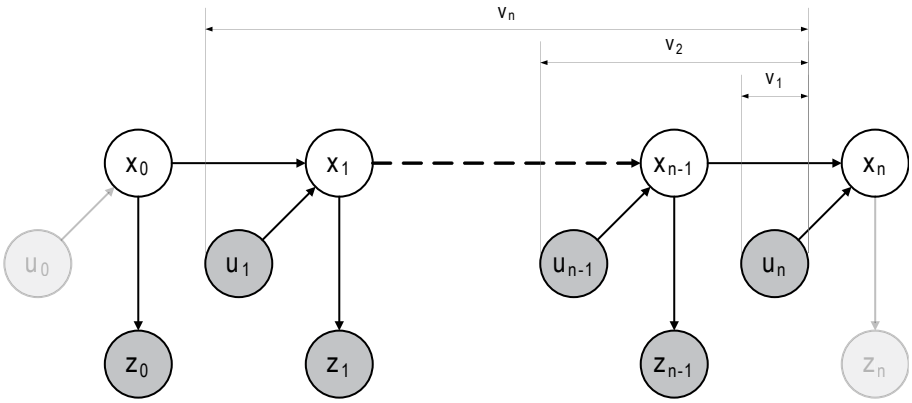


Figure 3.9: *Hidden Markov Model.* The states $x_0, x_1, ..., x_n$ are not directly visible. A state x_i depends only on the state x_{i-1} and the action u_i. An observation z_i depends only on the state x_i. The nodes u_0 and z_n are shaded as the method always starts with an observation (z_0) and ends with an action (u_n). Additionally, the connection between the actions u_1, ..., u_n and the accumulated controls $v_1, ..., v_n$ is illustrated (v_m is the sum of the last m actions, which is the sum of all actions since u_{n-m+1}, cf. Definition 8).

With this definition and proposition 7, it is easier to describe a sequence of previous states, given a current state x_n and a sequence of actions $u_1, ..., u_n$ that led to this state:

$$x_{n-m} = f_c^*(x_n, v_m).$$

Figure 3.9 illustrates the connection between u_1, ..., u_n and v_1, ..., v_n.

3.3.2 Memory Organization

Two information types are available: the sequence of sensor data (observations) and the sequence of control data (actions). These two types of information are organized in a memory which is a matrix M with two columns and n rows. The first column is a vector $\vec{z} = (z_0, z_1, ..., z_n)$ that contains the sequence of observations where the most current observation is z_n. The second column is a vector $\vec{u} = (u_0, u_2, ..., u_n)$ that stores the associated control data.

Each row is a tuple (z_i, u_i) that contains an observation and the corresponding action at the observation time. The number of rows in the matrix increases with each observation. To be able to process more than one observation at the same time, an action is allowed to be of type *do-nothing*.

3.3.3 State Estimation Using Least Squares

MBSE utilizes the memory described above. This section presents the kind of information delivered by a single measurement, what is known from a single observation in the past, and

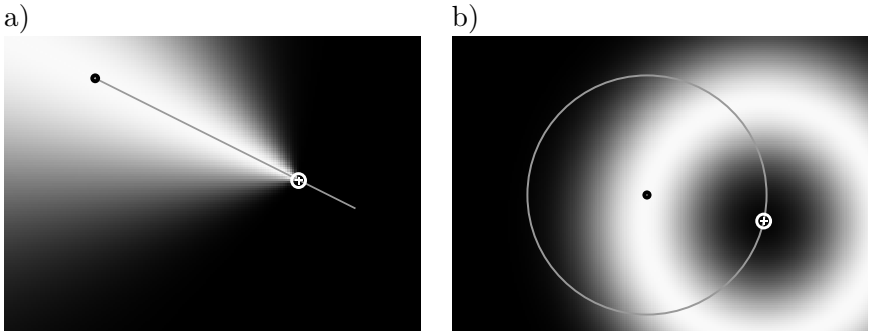


Figure 3.10: *Information gain of a single measurement.* White circle: landmark position. Black dot: simulated robot position. a) Gray line: bearing from the robot to the landmark. The squared difference between the real and the expected landmark bearings is illustrated by the function in the background: $f_1(x, y) = (\arctan(y_l - y, x_l - x) - \alpha)^2$, where α is the angle between north and the bearing from the robot to the landmark. Note that bright areas stand for low function values and dark areas for high. b) Gray circle: distance measurement from the robot to the landmark. The squared difference between the real and the expected distance measurements to the landmark is illustrated by the function in the background: $f_1(x, y) = (\sqrt{(y_l - y)^2 + (x_l - x)^2} - d)^2$, where d is the distance between the robot and the landmark.

what is known from all observations in the past.

3.3.3.1 Information Gain of a Single Measurement

Usually, a single measurement does not provide enough information to determine the state. However, a measurement z constrains the set of possible states according to the observation function defined in 3:

$$X_z := (x \in X : f_s(x) - z = 0).$$

A larger set based on an expected maximum error of t in the measurement can also be defined:

$$X_{(z),(t)} := (x \in X : (f_s(x) - z)^2 < t^2).$$

The likelihood for a state x can be defined as the squared error, resulting from the squared difference between the real and the expected measurement:

$$f_1(x) := (f_s(x) - z)^2 .$$

Figures 3.10 and 3.11 show this function for selected examples.

a) b)

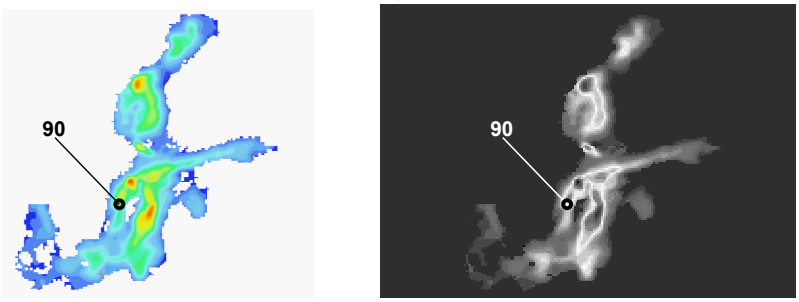


Figure 3.11: *Information gain of a single depth measurement.* a) Map of the Baltic Sea, taken from [5]. The black circle with the small number shows the robot's simulated position and the result of a single depth measurement (90 meters) b) Function $f_1(x, y) = (d(x, y) - s)^2$ shows the squared difference between the real and the expected measurements for each location. The function $d(x, y)$ delivers the depth, for position (x, y), as stored on the map, s is the depth measurement. Places with a small difference are shown bright.

3.3.3.2 Information Gain of a Past Measurement

When constraining the state by past measurement information, the actions since the observation have to be included. With the reverse of the control function defined in 5, the set

$$X_{z,u} := (x \in X : f_s(f_c^*(x, u)) - z = 0)$$

of possible states (assuming perfect measurements and execution of controls) can be defined. With the assumption of a maximum error t_z in the measurement, the larger set

$$X_{(z,u),(t_z)} := (x \in X : (f_s(f_c^*(x, u)) - z)^2 < t_z^2)$$

of possible states can be defined. Finally, a function that describes the likelihood for being in state x after executing control action u and having made the observation z in the beginning can be defined:

$$f_2(x) := (f_s(f_c^*(x, u)) - z)^2 .$$

This likelihood is given by the squared difference between the real measurement and the expected measurement in the state $f_c^*(x, u)$ calculated from x using the reversed motion model.

3.3.3.3 Information Gain of Multiple Past Measurements

To accumulate the information of multiple past measurements, I define the function

$$f_M(\vec{z}, \vec{u}) := \arg\min_k \sum_{t=1}^{n} (f_s(f_c^*(x_k, v_t)) - z_{n-t})^2 \tag{3.4}$$

which calculates the state x with the smallest sum of squared differences between the real measurement z_{n-i} and the expected measurement, for all states $f_c^*(x, v_i)$. Additionally, the value of the minimum is a measure of how good the measurements fit. If the measurements are perfect, the minimum value is 0. The more discrepancies there are between measurements, the higher the minimum value.

3.3.4 Properties of Memory-Based State Estimation

This section shows that the way to calculate the state using $f_M(\vec{z}, \vec{u})$ as defined above produces the same results as Bayesian filtering under the assumption that there is no error in the action execution and Gaussian error in the sensor measurements.

Assumption 9. *There is no error in the motion model:*

$$p(X_t = x_k | X_{t-1} = x_i, u_t) = \begin{cases} 1, & f_c^*(x_k, u_t) = x_i \\ 0, & otherwise \end{cases}$$

Assumption 10. *The sensor model has Gaussian error:*

$$p(z|X_t = x_k) = \mathcal{N}\left[f_s(x_k), \sigma^2\right](z),$$

with $\mathcal{N}\left[\mu, \sigma^2\right]$ being the normal distribution with mean μ and variance σ^2.

Theorem 11. *Let $\vec{u} = u_1, ..., u_n$ be an action sequence and $\vec{z} = z_0, ..., z_{n-1}$ be a sequence of corresponding measurements. Then the state $x_m = f_x(\vec{z}, \vec{u})$ determined by* MBSE *is equal to the state $x_b = \arg\max_k p(X_t = x_k)$ calculated by iterative Bayesian filtering (cf. algorithm 2.2) when the assumptions 9 and 10 hold.*

Proof. Consider line 2 of algorithm 2.2

$$\bar{p}(X_t = x_k) = \sum_i p(X_t = x_k | X_{t-1} = x_i, u_t) \cdot p(X_{t-1} = x_i).$$

Following assumption 9 (no randomness in the motion model), the sum can be eliminated:

$$\bar{p}(X_t = x_k) = p(X_{t-1} = f_c^*(x_k, u_t)).$$

Substitution in line 3 of algorithm 2.2

$$p(X_t = x_k) = \eta \cdot p(z_t | X_t = x_k) \cdot \bar{p}(X_t = x_k)$$

gives

$$p(X_t = x_k) = \eta \cdot p(z_t | X_t = x_k) \cdot p(X_{t-1} = f_c^*(x_k, u_t)).$$

The iteration of n observations and controls results in

$$\begin{aligned} p(X_n = x_k) = \eta_{n-1} \cdot \eta_{n-2} \cdot ... \cdot \eta_1 \cdot \\ p(z_{n-1}|X_{n-1} = f_c^*(x_k, u_n)) \cdot p(z_{n-2}|X_{n-2} = f_c^*(x_k, u_{n-1}u_n)) \cdot ... \cdot \\ p(X_0 = f_c^*(x_k, u_1u_2...u_n)) \\ = p(X_0 = f_c^*(x_k, v_n)) \cdot \prod_{i=1}^{n} \eta_i \cdot p\left(z_{n-i}|X_i = f_c^*(x_k, v_i)\right), \end{aligned}$$

using the definition of concatenated controls (definition 6). The first term[1] and the η_i can be joined to a constant ζ_1 resulting in

$$p(X_n = x_k) = \zeta_1 \prod_{i=1}^{n} p(z_{n-i}|X_i = f_c^*(x_k, v_i)).$$

Since ln is a strictly increasing function, it holds true that

$$\arg\max_k p(X_n = x_k) = \arg\max_k \ln(p(X_n = x_k))$$

therefore that

$$\arg\max_k p(X_n = x_k) = \arg\max_k \left(\ln \zeta_1 + \sum_{i=1}^{n} \ln p(z_{n-i}|X_i = f_c^*(x_k, v_i)) \right)$$

can be deduced. According to assumption 10, it holds true that

$$p(z|x) = \frac{1}{\sigma\sqrt{2\pi}} \exp\left(-\frac{1}{2} \left(\frac{z - f_s(x)}{\sigma} \right)^2 \right)$$

which implies

$$\ln(p(z|x)) = -\frac{1}{2} \left(\frac{z - f_s(x)}{\sigma} \right)^2 + \zeta_2 \ .$$

Applying this, finally results in

$$\arg\max_k p(X_n = x_k) = \arg\min_k \sum_{i=1}^{n} \left(f_s(f_c^*(x_k, v_i)) - z_{n-i} \right)^2 .$$

This equation's left side is the state calculated by iterative Bayesian filtering, while the right side is the state as determined by *MBSE*, cf. (3.4) in section 3.3.3.3. This proves theorem 11. □

[1] The term $p(X_0 = x)$ describes the a-priori probability distribution of the robot state. At this point, it is assumed that the initial robot state is unknown and therefore uniformly distributed, which implies that $p(X_0 = x)$ is a constant for all x.

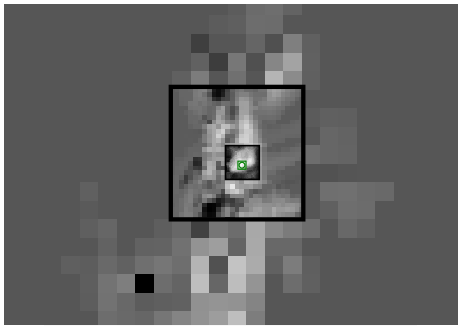


Figure 3.12: *Stepwise grid refining.* The global minimum of the function is found using a grid. The grid spacing is reduced in several steps.

3.3.5 Algorithmic Variants

Calculating

$$f_M(\vec{z}, \vec{u}) := \arg\min_k \sum_{t=1}^{n} (f_s(f_c^*(x_k, v_t)) - z_{n-t})^2$$

can be done in several ways. A few of them are shown in this section. Some of the methods reduce the number of states when looking for the minimum. Other variants reduce the number of measurements that have to be incorporated.

3.3.5.1 Reducing the Number of Locations

Several ways to reduce the number of locations for which the function $f(x, y)$ has to be calculated are discussed in this subsection.

Stepwise Grid Refining This method calculates the function only for the positions (x, y) which are located on a grid with a spacing of d. The value of d has to be chosen, depending on the domain, so that there is always a grid point close to the real minimum of the function. The algorithm finds the grid cell with the function's minimal value. This grid cell is used to define a new grid around this cell with smaller spacing. This process is repeated until the desired position precision is determined. Figure 3.12 shows an example.

Gradient Descent The function's minimum can also be found using the gradient descent method (cf. [71]). Parameters for the algorithm, like number of iterations and step length, have to be chosen depending on the domain. Figure 3.13a) shows an example. The gradient descent algorithm can run into local minima. However, once the true minimum has been found (for example using the stepwise grid refining method described above), choosing this position as the starting position, in the next step, should prevent the algorithm from switching to the wrong local minimum.

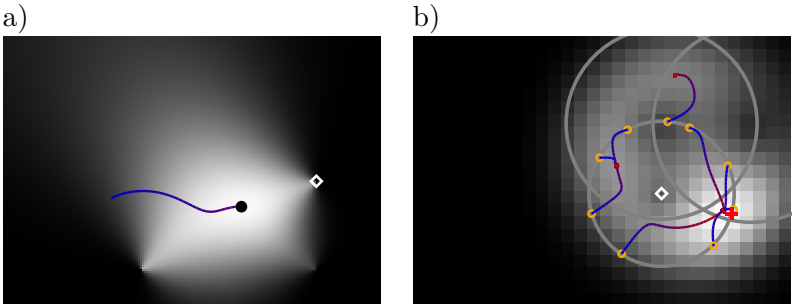


Figure 3.13: a) *Gradient descent.* The function in the background results from combining three bearings to the landmark (white rhombus). As there is only one local and global minimum, the gradient descent finds the robot's position (marked by the black circle). The line illustrates the path of the gradient descent. b) *Gradient descent with multiple starting points.* To be sure not to end in a local minimum, gradient descent is started at several positions. The function in the background results from combining three distance measurements. The starting points are chosen by randomly selecting points out of the set $X_{(z,u),(t_z)}$ which contains all possible positions taking only measurement s into account. The set of gradient descent runs finds all three of the function's local minima.

Gradient Descent with Multiple Starting Points Another way to solve the problem of local minima is to select a set of starting positions and run the gradient descent method for each of these positions. The run that ends up with the lowest minimum can be considered to have found the global minimum. However, some care should be taken when selecting starting positions. I propose a way to choose starting positions based on the knowledge obtained from the last measurement s. The set of starting positions is defined as a random subset of $X_{(z,u),(t_z)}$ (cf. 3.3.3.1) with a fixed number of elements, where z is the last measurement, u the control action performed since that measurement, and t_z the maximum expected measurement error. The set's size and the threshold value t_z have to be chosen depending on the domain. This set should contain at least one position near the global minimum, which is a good starting position for gradient descent. Figure 3.13b) gives an example.

Dimension Reduction This section shows how the calculation can be simplified when the sensor model fulfills special requirements. Let $f_s(\vec{x})$ be the sensor model which predicts a measurement for a state vector $\vec{x} = (x_1, ..., x_n)$ of dimension n. In section 3.3.3.2, the function

$$f_2(\vec{x}) := (f_s(f_c^*(\vec{x}, u_i)) - z_i)^2$$

was defined, which calculates the difference between the measurement and the expectation for a given state $\vec{x}$ for an observation (z_i, u_i). Consequently, the solution of equation

$$z_i = f_s(f_c^*(\vec{x}, u_i))$$

is a set of states $\vec{x}$ for whose elements the expectation is identical to the measurement. If the first $n-1$ components of $\vec{x}$ are fixed and the expectation is required to be identical with the measurement, a constraint for the component x_n of $\vec{x}$ can be obtained that is used to define the function

$$c\left((x_1, ..., x_{n-1}), z_i, u_i\right) := x_n.$$

That function c is used to define a function

$$h(\tilde{x}) := \sum_{i=1}^{k} \left(\bar{c}(\tilde{x}) - c(\tilde{x}, z_i, u_i)\right)^2$$

which shows, for each partial state $\tilde{x} = (x_1, ..., x_{n-1})$, how good several observations (z_i, u_i) fit together. The function $\bar{c}(\tilde{x})$ is defined as

$$\bar{c}(\tilde{x}) = \frac{1}{k} \sum_{i=1}^{k} c(\tilde{x}, z_i, u_i).$$

With these functions, the position can be estimated:

$$\vec{x_{best}} = \begin{pmatrix} \text{argmin}_{\tilde{x}}(h(\tilde{x})) \\ \bar{c}(\text{argmin}_{\tilde{x}}(h(\tilde{x}))) \end{pmatrix}.$$

This increases the calculation speed as the dimension of $\tilde{x}$ is only $n-1$.

In a 2-D world with bearing sensors, this leads to

$$y = c(x, \alpha, \Delta) = y_l - \Delta_y - (x_l - x - \Delta_x) \cdot \tan(\alpha)$$

where Δ_x and Δ_y are the components of $\Delta := f_c(v_i)$. Figure 3.14 illustrates functions c and h, for example. A detailed description of the dimension reduction for a three-dimensional example (position and rotation) can be found in [42].

3.3.5.2 Reducing the Number of Used Measurements

The second method to reduce the computational costs of the algorithm is to reduce the number of measurements. So far, no restrictions on the size of the memory were specified. This means the longer the robot runs, the more information is processed.

Remove Old Measurements Removing older measurements is an obvious solution to this problem. Several strategies for selecting observations are shown in chapter 4.

Remove Measurements with a Low Contribution In this section, an observation selection strategy based on the *contribution* of a measurement is explored. First, a function is defined which calculates the direction of the steepest decent of function f at position x

$$a(\vec{x}, M) = \max_{\vec{y}:|\vec{y}|=\varepsilon} \{\nabla f(\vec{x} + \vec{y}, M)\},$$

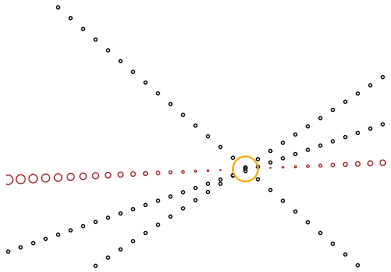


Figure 3.14: *Dimension reduction.* Three measurements were taken. The lines of black circles show the result of the computation of function $c(x, \alpha, \Delta)$ for several discrete values of x for each observation (α, Δ). The brown circles show the function $\bar{c}(x)$. The size of the brown circles illustrates function $h(x)$. The position x_{best}, with the smallest value of h, is marked by the large orange circle. Figure 3.13 shows the function f for the same experiment.

where ∇ is the nabla operator (cf. [11]). Note that this function can also be applied to a local extremum as for the steepest decent is searched for at all locations $\vec{x} + \vec{y}$ with a distance of ε to $\vec{x}$. With that, the *contribution* of an observation set N with respect to all observations M is defined as the angular difference between the directions of the steepest decent:

$$c(N, M) := 1 - \frac{a(x_{best}, M) \cdot a(x_{best}, N)}{|a(x_{best}, M)| \cdot |a(x_{best}, N)|},$$

where $x_{best} = \text{argmin}_x f(x, M)$. For practical reasons, ε should be greater than the difference between the real minimum and the estimated minimum x_{best} of the function f.

The function $c(N, M)$ which calculates the *contribution* can be used to define selection strategies for observations. The simplest option is to keep the number of elements in N fixed; as soon as a new observation is made, the observation with the lowest contribution is deleted. Figure 3.15 shows the result of this strategy for experiments with bearing and distance sensors in a 2-D world.

3.4 Proof-of-Concept Experiments

To test and demonstrate the state estimation method I chose a self-localization scenario from RoboCup. This section describes the experiments.

3.4.1 Experimental Setup

The testing scenario was bearing-only localization [44, 43] on a RoboCup field. In this case, the only measurements are horizontal landmark bearings. There are six unique landmarks (the two goal posts of each goal and center beacons) and fourteen ambiguous landmarks (field line intersections). The landmark setup is shown in figure 3.16a). All tests were done in simulation. Like on a real robot, in simulation the bearing sensors had a limited field of view (60 degrees) and

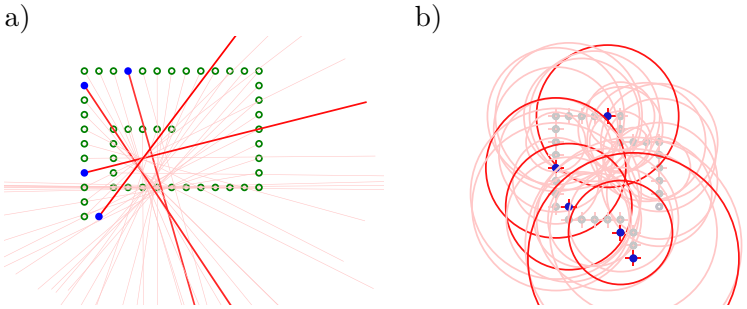


Figure 3.15: *Selecting observations based on their contribution.* The figures show two examples. One from a bearing measurement example and another one from a distance measurement example. The faded measurements were not considered as they were marked to have a low contribution during the past robot run. Only the measurements taken at the highlighted places are used to calculate the position.

a Gaussian error with a 6 degrees standard deviation. The motion model used had an error of 10 percent of the amount of the action in all three dimensions (x, y, rotation). This error causes the dead-reckoning robot position to drift away from *ground truth* (see figure 3.16b). Additionally, small kidnappings were added as they appear regularly in RoboCup games when robots collide. To test self-localization, the robot followed a virtual ball that was moved around on the field, producing the path shown in figure 3.16b). The positions of this path were compared to the corresponding ground truth positions.

3.4.2 Experimental Results

To measure the localization quality, for each time step the difference between the position estimated by the memory-based approach and the ground truth position was determined. This was done for three different scenarios. The first is localization based only on motion data (initialized with the true position in the first step). The second version used the complete history of measurements and executed motions for position calculation. In the last experiment, only the last observations of each landmark type were used. Results are shown in table 3.1 and figure 3.16c,d).

As explained above, using only motion information results in a estimated position drift, leading to a high deviation. Using all information still has this problem because of the kidnappings. Using the landmark selection strategy that selects the last five observations of each landmark type leads to the best results. However, due to the small number of measurements and the noise, the position is subject to small jumps.

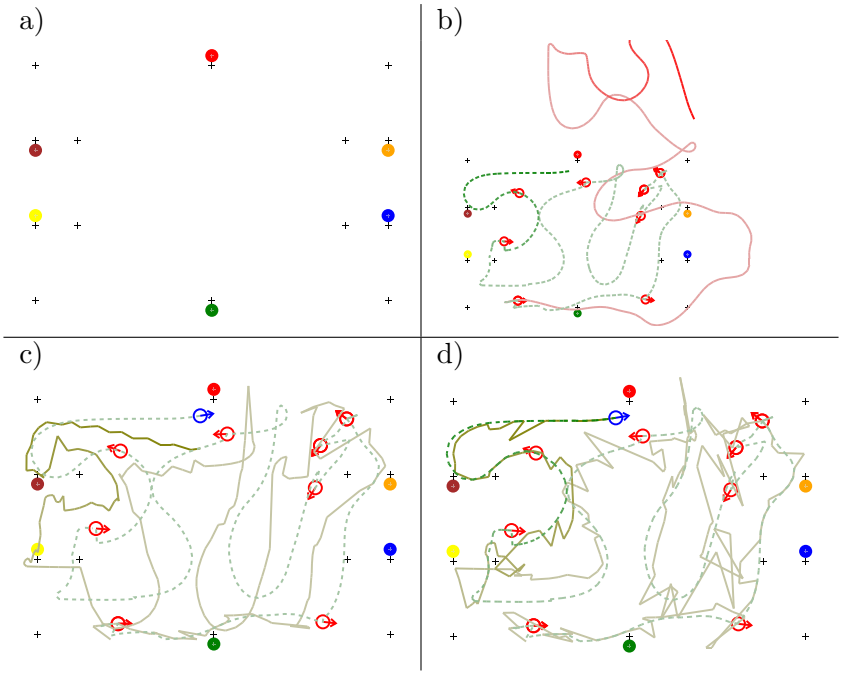


Figure 3.16: *Experimenal results.* a) The artificial landmarks of a RoboCup field. Circles: unique markers for horizontal bearings (center beacons, goal posts). Small crosses: Intersections of field line (not distinguishable). b) Dashed line: The true path the robot took in simulation, starting in the left bottom corner; Solid line: result of localization based only on the knowledge about the action effects. Note that the result gradually drifts away from ground truth because of the noise added to the motion model and the small kidnappings. Circles with arrows along the solid line: Poses where the robot was kidnapped randomly (simulating charging by other robots). c) Solid line: result of memory-based localization using all perceptions. d) Solid line: result of memory-based localization using only the last perceptions of each landmark type.

Selection strategy	Error in *mm*
motion only	$277,4 \pm 183,4$
all landmarks	$52,3 \pm 21,7$
last 5 of each landmark type	$25,7 \pm 19,8$

Table 3.1: *Experimental results.* The selection strategies are denoted in the left column. The right column contains the resulting averaged difference between the calculated position and the ground truth.

4 Bearing-Only Localization

In this chapter, a localization method based on the *memory-based paradigm*, cf. chapter 3, is introduced. Localization is an important component of almost every autonomous mobile robot. The ability to localize, with respect to a map, is crucial for decision making. A big part of the existing work in localization is based on the use of range sensors like sonar, laser range finders, and radar [68] [16] [70] [80].

Digital cameras are an alternative to the sensors mentioned above. Since processors became faster, image processing algorithms which extract features, like landmarks, can be used to create the input for localization methods. However, the information that can be obtained from digital images is of a different nature. The range measuring capabilities are limited compared to sonar sensors, laser range finders, and radar. Another disadvantage is that cameras usually have a limited angle of view. In contrast to the poor range measurement capabilities of vision-based systems, horizontal bearings can be determined very easily. Thusly, there are some bearing-only approaches for localization [18, 72].

However, the existing bearing-only localization approaches are based on sequential estimation techniques. In contrast to existing approaches, the method proposed in this chapter does not need internal representation of the robot's position, updated by alternating motion and sensor updates. Instead, the location is calculated by applying constraints on the robot's position which are derived from the observations and performed actions stored in a short-term memory. The method strictly follows the *memory-based paradigm* and makes use of the concepts *dimension reduction* (introduced in 3.3.5.1) and *measurement selection* (introduced in 3.3.5.2).

While the memory-based method described in this chapter can be used for bearing-only localization, it can also be used as a template generator for localization with particle filters. This is useful when more than just bearing information has to be processed. The distribution of the template positions reflects the position calculation accuracy, which depends on the selected landmark configuration.

The method described in this chapter was developed for and tested on an Aibo robot. However, it is not limited to that platform.

Section 4.1 motivates the memory-based approach. Section 4.2 describes the method in detail. Section 4.3 describes the experiments that were done in simulation and conducted on Sony Aibo robots. A discussion follows in section 4.4.

This chapter is based on [42], [44], and [47]. Log files with ground truth data used for the experiments were recorded by Max Risler.

4.1 Motivation

Compared to distance measurements, high precision is the advantage of bearing measurements. Vision-based distance measurements to landmarks are usually less reliable. There are three main methods to obtain the distance to a landmark: stereo vision, size measurements, and vertical bearings. Common to them all, the reliability decreases with an increased distance to the landmark used. That is not the case for horizontal bearings. Bearings to far-away landmarks are as precise as to near landmarks.

Stereo vision can be affected by poor object matching. Size-based distance measurements are sensitive to obstructions and can be inaccurate with low resolution cameras. A partially obstructed landmark can still be used to obtain a bearing, while distance measurement based on the landmark's size, in the image, is affected by the obstruction. The use of vertical bearings has the disadvantage that the camera's orientation to the ground has to be exactly known to obtain precise distance measurements.

In the last years, methods based on the Bayes filter have been the standard approach to the localization problem. These methods need a probabilistic model of the sensors and the motions, as well as an internal probabilistic state representation (belief). As the representation is a probability density distribution, it is suitable to be used for information integration. The existing methods perform so well on robots equipped with infra-red distance sensors or laser range finders because this type of sensory information can be accumulated by belief probability distributions very easily, even if the models chosen are only a rough approximation of the distribution. However, for a classic localization problem (x, y, θ), where x and y are coordinates and θ is the orientation, a single bearing does not provide enough information to determine the position; at least three measurements are needed. When the robot moves between these measurements, the information has to be accumulated and combined with odometric data. The low information gain of bearings can cause the need for additional efforts in the representational design. Depending on the approximation type (Gaussians, grid, particles), there are different limitations. For example, the resulting distribution of a single bearing measurement might not be Gaussian. This can cause difficulties for Kalman filters as the they approximate the state using Gaussians. The fact that a single bearing is only a weak position constraint causes a high number of particles to be needed with a particle filter. With a grid-based approach, a high-grid resolution is needed to accumulate the bearing information.

The method described in this chapter was motivated by the experiences collected with the localization method our team used in RoboCup for the Aibo robots. A particle filter was used, as described in [25, 19, 83, 39, 4, 2, 3]. Our method is described in [66, 65, 64]. In RoboCup game situations the distance measurements to landmarks are very error-prone, because such size-based measurements are affected by visual obstructions caused by other robots. However, the particle filter was unable to accumulate the information provided by the bearings. The memory-based approach presented in this chapter can cope with sparse information (bearing-only, limited angle of view).

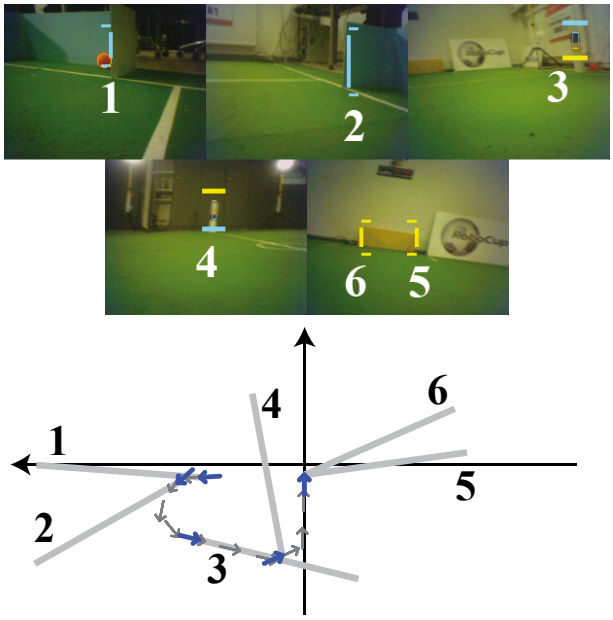


Figure 4.1: *Odometry and horizontal bearings.* Top: five images with six horizontal bearings (1: right goal post, 2: left goal post, 3 and 4: center landmarks, 5 and 6 goal posts) Bottom: gray arrows show the robot's odometry at different times, bold arrows show the odometry associated with the horizontal bearings.

4.2 Localization

As stated above, bearing-only localization works using two kinds of input: bearing measurements and odometry data. The vector

$$\vec{\alpha} = (\alpha_{l_1}, \alpha_{l_2}, ..., \alpha_{l_n})$$

contains the measured bearings to the landmarks $l_1, l_2, ..., l_n$. These angles were measured at different times $t_1, t_2, ..., t_n$.

The second input is the knowledge about the robot's actions. The vector

$$\vec{u} = (u_1, u_2, ..., u_n)$$

contains the robot's odometry at times $t_1, t_2, ..., t_n$.

A robot can obtain these vectors $\vec{\alpha}$ and $\vec{u}$ by storing its observations and the according odometry in a buffer. Figure 4.1 shows a visualization of such a buffer.

In this section, a function $F(x, y, \vec{\alpha}, \vec{u})$ is defined that describes the likelihood of a robot being at position (x, y) on the field. This function can be used to calculate the robot position (the position of the function's maximum) or to generate templates for particle filter localization.

Subsection 4.2.1 discusses a case where the robot is stationary; section 4.2.2 incorporates odometry. In sections 4.2.3 and 4.2.4, it is shown how the robot's position can be calculated and how template poses for particle filters can be generated.

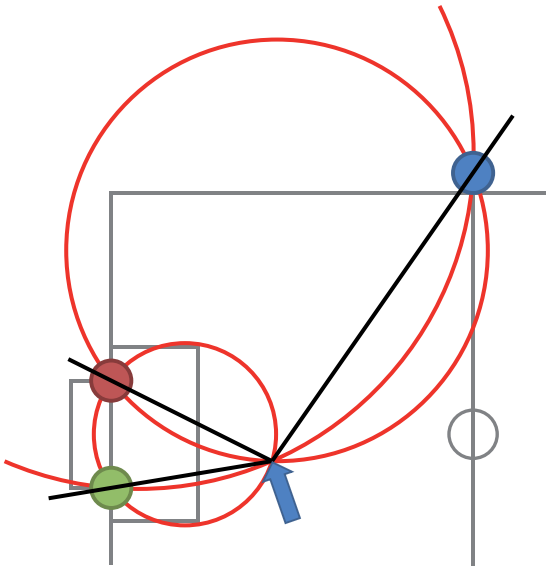


Figure 4.2: *Determining the robot's position by three horizontal bearings to known landmarks.* Black lines: horizontal bearings. Large circles: circles given by two bearings and two landmarks.

4.2.1 Simultaneous Observations

In this subsection, two different methods are shown which can be used to determine the position of a stationary robot. The first uses well-known simple geometry; the second is a constraint-based approach.

Pose Estimation Using Simple Geometry When a robot perceives three landmarks without moving between the observations, the position calculation is straightforward: with the known landmark positions, a single circle can be constructed for each bearing pair. The circle radius is determined by the angle difference and the distance between the landmarks. The intersection point of the circles is the only possible robot position. Figure 4.2 shows an example.

Pose Estimation Using Angular Constraints A single observation of a landmark l at a certain relative angle constrains the angle ϑ_l the robot can have at a certain position (x, y) on the field. This angle is given by

$$\vartheta_l(x, y) = \arctan\left(\frac{y_l - y}{x_l - x}\right) - \alpha_l$$

where (x_l, y_l) is the landmark position, on the field, and α_l is the relative angle to the landmark. Figure 4.3a) shows this function. When two bearings to two landmarks are given, the function

$$D_{l_1,l_2}(x, y) = (\vartheta_{l_1}(x, y) - \vartheta_{l_2}(x, y))^2$$

describes the likelihood of being at position (x, y). Figures 4.3b) and 4.4a,b) show function D for several examples. The function's shape represents how good a certain landmark pair is suited to constrain the position on the field. For example, a plateau in this function (like in figure 4.3b) means a small error in an observation leads to a large error in the resulting position.

The function $D_{l_1,l_2}(x, y)$, introduced above, describes for each position (x, y) how good the angles ϑ_{l_1} and ϑ_{l_2} obtained from two different horizontal bearings match. To use more than two observations $\alpha_{l_1}, \alpha_{l_2}, ..., \alpha_{l_n}$, the average angle of all resulting $\vartheta_{l_1}, \vartheta_{l_2}, ..., \vartheta_{l_n}$ can be calculated for each position (x, y) using this formula

$$\vartheta_{average}(x, y) = \arctan\left(\frac{\sum_{i=1}^{n} \sin(\vartheta_{l_i}(x, y))}{\sum_{i=1}^{n} \cos(\vartheta_{l_i}(x, y))}.\right)$$

Figure 4.5a) shows function $\vartheta_l(x, y)$ for three different landmarks and the resulting average angle. Using $\vartheta_{average}(x, y)$, the function

$$G(x, y) = \sum_{i=1}^{n} (\vartheta_{average}(x, y) - \vartheta_{l_i}(x, y))^2$$

can be defined which describes how similar the angles ϑ_l are. This function has its maximum at the position (x, y) which fits best with all observations $\alpha_{l_1}, \alpha_{l_2}, ..., \alpha_{l_n}$. Furthermore, the function provides an estimation of the position error for known errors in the observation. Figure 4.5b) shows this function for three observations.

4.2.2 Incorporating Odometry

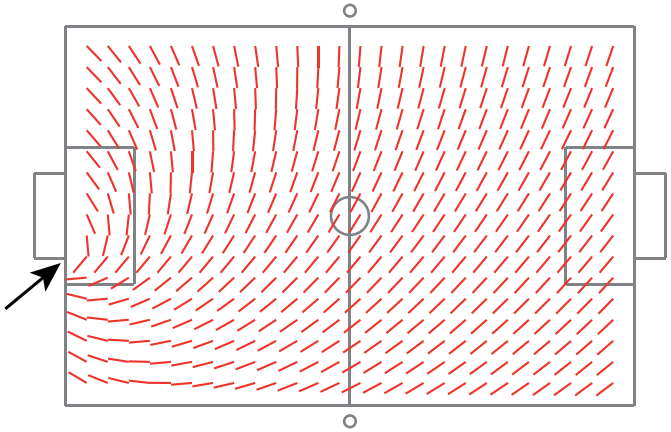
To incorporate odometry, the function $\upsilon_l(x, y, \Delta_{odometry_l})$ is defined which determines the robot's angle at position (x, y) when the landmark l was seen at angle α_l and the robot moved $\Delta_{odometry}(\Delta_x, \Delta_y, \Delta_\phi)$ since the observation. Figure 4.6a) illustrates these parameters and the resulting angle υ_l. To determine υ_l, a triangle is defined which has its corners at position (x_l, y_l) of the landmark l (angle β), at position (x, y) (angle γ), and at position (x_0, y_0) where the observation was taken (angle δ). Figure 4.6b) shows this triangle. Note that in this triangle (x_l, y_l) and (x, y) are fixed. The position of (x_0, y_0) can be calculated using the angle ω from (x, y) to (x_l, y_l) and the distance Δ_d which the robot walked:

$$\begin{aligned} x_0 &= x + \cos(\omega + \gamma) \cdot \Delta_d \\ y_0 &= y + \sin(\omega + \gamma) \cdot \Delta_d \end{aligned}$$

where γ follows using sine rule:

$$\begin{aligned} \gamma &= \pi - \delta - \beta \\ &= \pi - \alpha_l - \arctan\left(\frac{\Delta_y}{\Delta_x}\right) - \arcsin\left(\frac{\Delta_d \cdot \sin(\delta)}{d_l}\right). \end{aligned}$$

a)



b)

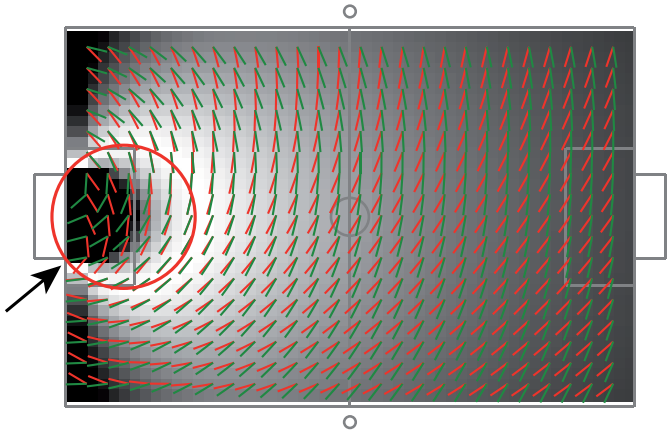


Figure 4.3: *Angular constraints for location estimation.* a) The red lines show the angle ϑ_{l_1} on the field, for each position. This angle results from a given horizontal bearing: the goal post marked with the black arrow. b) The green lines show the angle ϑ_{l_2} on the field that results from a bearing to the right goal post. The gray-scale grid, in the background, shows the square of the difference $(\vartheta_{l_1} - \vartheta_{l_2})^2$ between the angles. This function has a peak near the positions covered by the circle obtained using simple geometry.

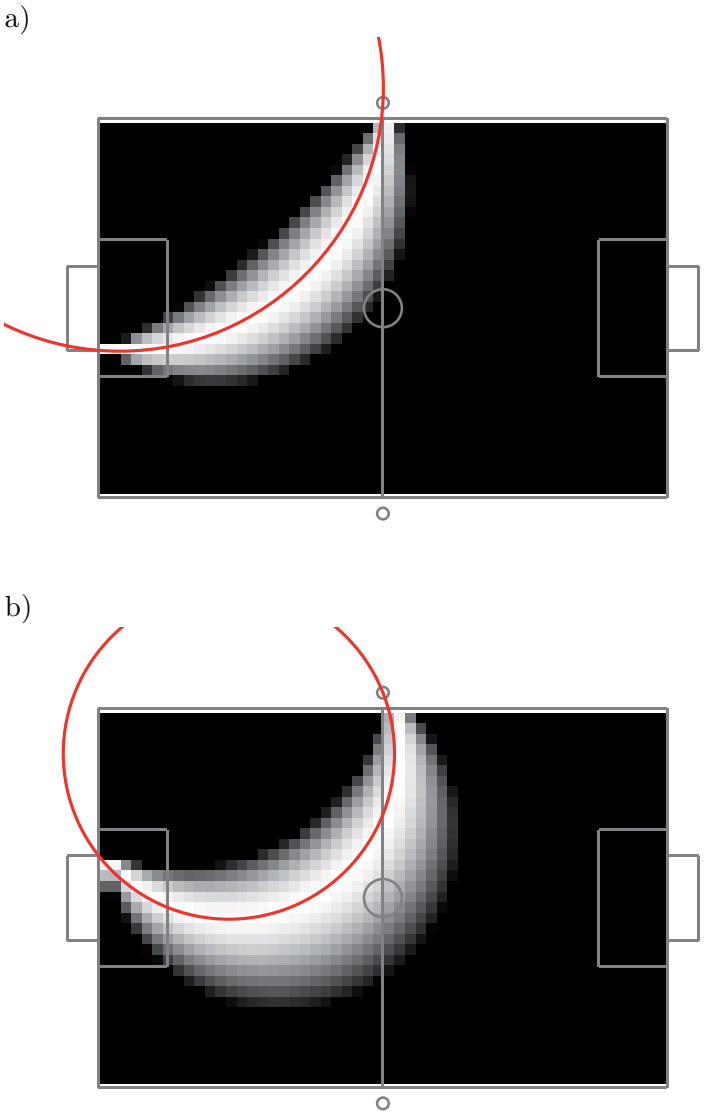


Figure 4.4: *Angular constraints for location estimation.* a) The difference $(\vartheta_{l_1} - \vartheta_{l_3})^2$ between the angles resulting from the bearings to the left goal post and the right center flag. b) The difference $(\vartheta_{l_2} - \vartheta_{l_3})^2$ between the angles resulting from the bearings to the right goal post and the right center flag.

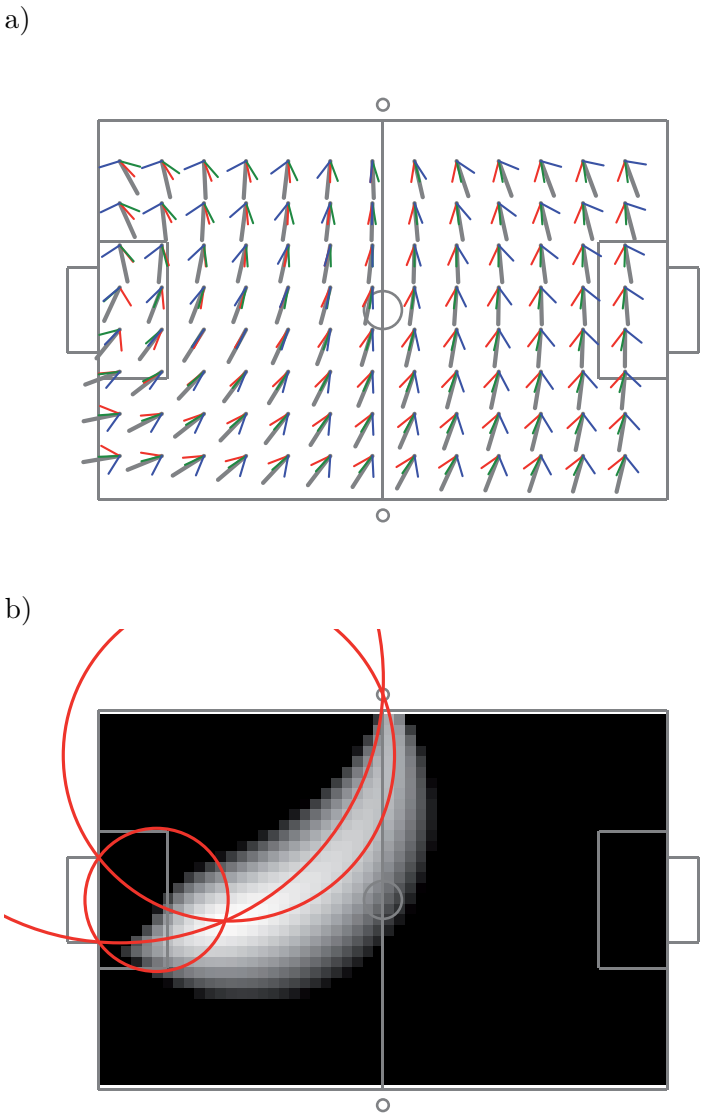


Figure 4.5: *Similarity of angles.* a) The thin lines show the angles $\vartheta_l(x, y)$ for three different observations. The bold line shows the average angle. The robot's position is constrained to the positions where the angles are similar. b) Function $G(x, y)$ is displayed as a height map. White: small difference between the angles, black: large difference between the angles. The red circles are obtained from the method using simple geometry described above.

Note that there can be two possible values for γ when the distance d_l to the landmark is smaller than the distance Δ_d walked. With the known position it (x_0, y_0) follows that

$$\upsilon_l(x, y, \Delta_{odometry_l}) = \vartheta(x_0, y_0) + \Delta_\phi.$$

When the robot is at position (x, y), has seen the landmark l at angle α_l some time ago, and has moved by $\Delta_{odmetry_l}$ since that observation, the function υ_l gives the angle the robot must have. Similar to the function G from section 4.2.1, a function

$$F(x, y) = \sum_{i=1}^{n} \left(\upsilon_{average}(x, y) - \upsilon_{l_i}(x, y)\right)^2$$

is defined which describes the likelihood of the robot being at position (x, y). This function can incorporate an arbitrary number of observations from the past and does not need any internal representation of the position updated by alternating sensor and motion updates. The selected sensor information $\vec{\alpha}$ and the corresponding motion information $\vec{u}$ are processed simultaneously.

4.2.3 Calculating the Robot's Pose

The position of the maximum of function F, given in the last section, is the robot's position. The robot's rotation can immediately be calculated using $\upsilon_{average}$ or the angle υ_{l_0} defined by the last observation. When a fast and rough estimation of the robot pose is wanted, the maximum can be determined by an iteration through the function's domain. When a more accurate estimation is wanted, it can be obtained by means of standard methods, like Gradient Descent, with only a few iterations. Note that such methods usually find only local maxima of the function.

4.2.4 Generating Templates for Particle Filters

Often, there is more information than the horizontal bearings to unique landmarks to determine the robot's pose. In such cases, the function F, described in section 4.2.2, can be used to create template poses for sensor resetting. This is in particular useful when only a small number of particles can be used because of computational limitations. To obtain a fixed number of samples, function F can be normalized such that all values are between 0 and 1:

$$F' := \frac{1}{(1 + F^2)}.$$

Then, a template pose can be created at each position (x, y) with $random() < F'(x, y)^n$ where n is a parameter to adjust how much the sample poses can deviate from the maximum. Figure 4.7 shows templates obtained from function F.

4.3 Experiments

To test the method described in the previous section, several experiments were conducted. In this section, the accuracy of the localization algorithm is shown. Experiments concerning the

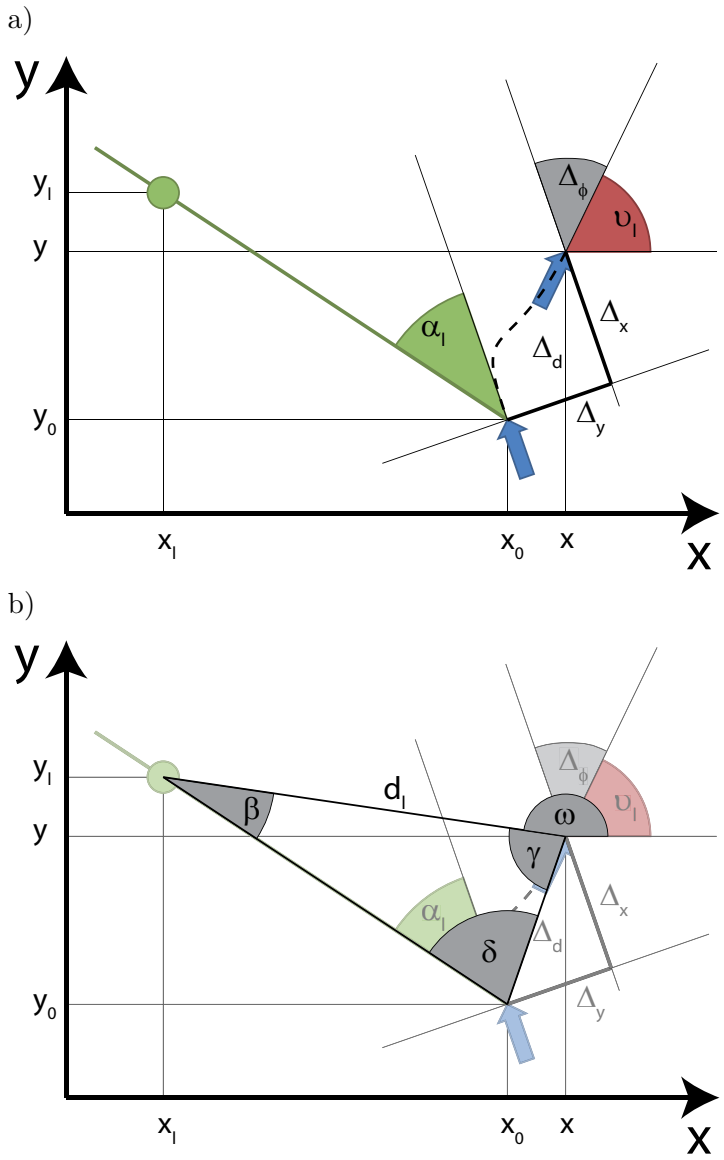


Figure 4.6: *Geometrical considerations to determine the robot's angle* υ_l.

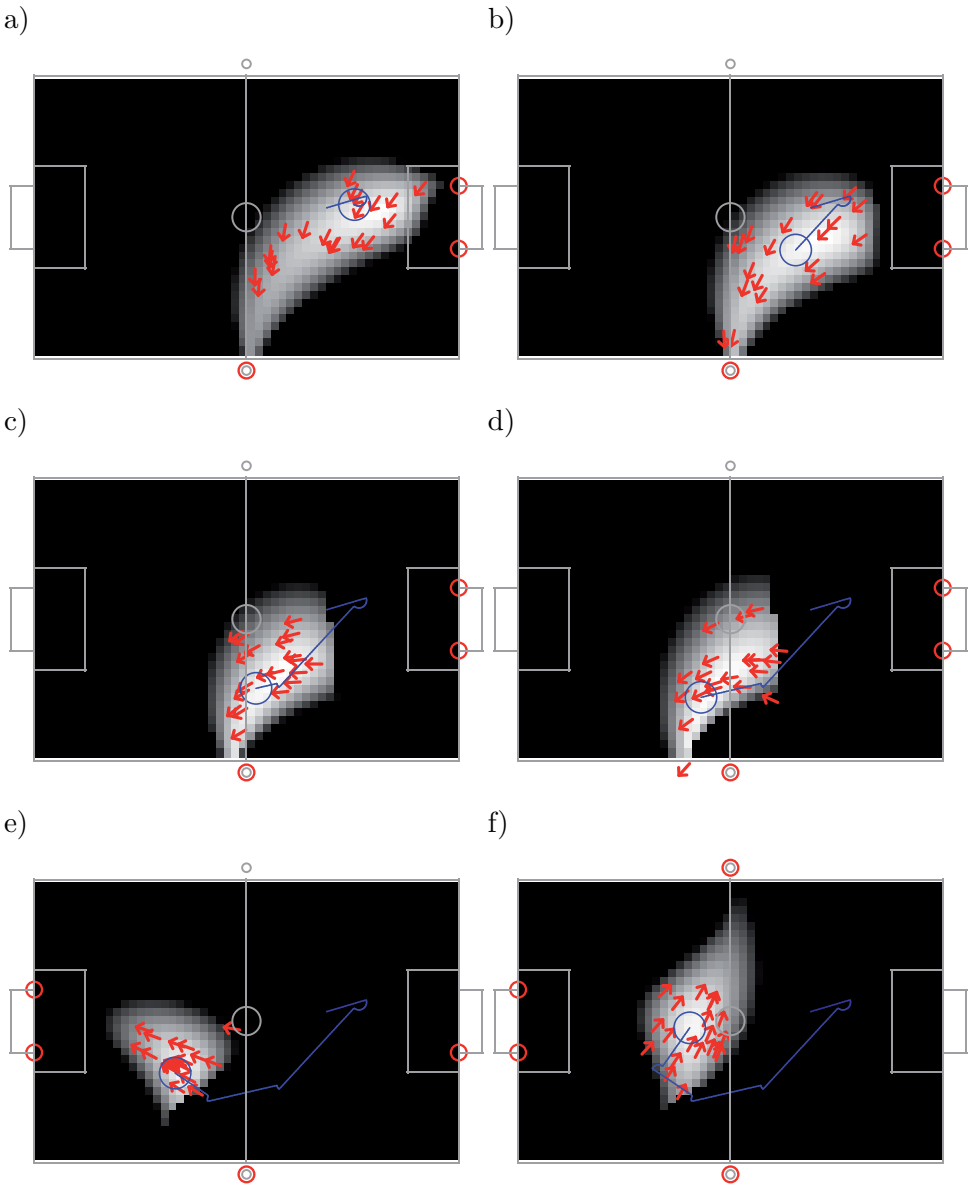


Figure 4.7: *The function* $F(x, y)$. White: high likelihood; black: low likelihood; Arrows: position templates that can be used for sensor resetting in particle filters. Note that usually only a small number of these templates are used. Small circles: the landmarks that were used for position calculation. Large circle: the robot's pose (known from the simulation). Path: the way the robot walked.

landmark selection were done in simulation and described in section 4.3.1. An experiment using only two landmarks to localize as well as a comparison to vision-based ground truth were done using a real Aibo robot and are shown in section 4.3.2.

4.3.1 Experiments in Simulation

The method described in section 4.2 can process an arbitrary number of observations. This experiment's goal was to find out which information from the short-term memory is sufficient for localization. Several strategies to select landmark observations were compared. To be able to compare several selection strategies, recorded data created using a simulation of an Aibo on a soccer field was used. In this simulation, a typical RoboCup *approach-ball*-behavior was used. With this behavior, the robot walks to the ball in a curve while panning the head to scan for landmarks. The ball was moved around randomly on the field. The recorded data consists of images, head joint sensor data, odometry data and perfect ground truth data for comparative purposes. The total path length is 80 meters. The log-file consists of 6600 frames where 30 frames correspond to one second.

Using the camera images and the knowledge about the camera's direction of view (obtained by the sensor readings), the robot can determine the horizontal bearings to six unique landmarks: the left and right post of each goal and two artificial beacons at the center line.

The algorithm can be fed with any combination of observations from the perception history. These four selection strategies were implemented:

- select all observations within the last 200 frames
- select the last observations of each landmark type
- select the last two observations of each landmark type
- select the first and the last observation of each landmark type within the last 200 frames.

Note that choosing 200 frames as a limit is arbitrary but motivated by the fact that within these 200 frames the robot scans a 180° range at least two times. Figure 4.8 shows which landmarks are observed in which frame and what the different strategies select. Figures 4.9 and 4.10 show an illustration of perceptions and odometry data. The selected landmarks are highlighted.

To compare the accuracy of the method, depending on the different inputs, the average position and angle error were measured for the full 80 meter run, for each selection strategy. The results are shown in table 4.11. Figure 4.12 shows the deviation between the ground truth position and that determined by the method.

The experiment showed that the method localizes a robot very precisely. The selection strategy has an influence on the localization accuracy. Best results are obtained by the strategies which use information from the entire range of the 200 frames slot. This is because information about the position can be obtained when landmarks are seen from different perspectives, caused by the robot's motion. However, it does not matter how many observations are combined. Even the simplest strategy that selects observations from the whole 200 frames slot (the first and the

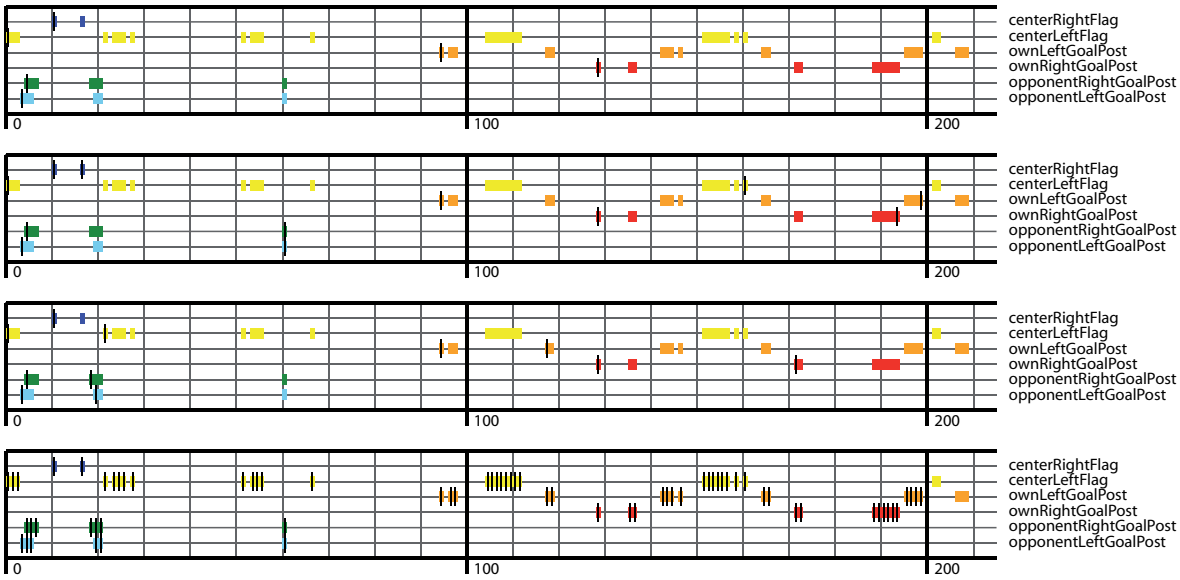


Figure 4.8: *Landmark selection strategies.* The horizontal axis gives the index in the observation memory. The latest observations have index zero. The oldest data is 210 frames old. Colored bars indicate frames when certain types of landmarks were observed. Small black vertical lines indicate the selected frames. Strategy 1) *select last occurrence of each landmark type.* Strategy 2) *select first and last frame of each landmark type within the last 200 frames.* Strategy 3) *select last two occurrences of each landmark type.* Strategy 4) *select all landmarks within the last 200 frames.*

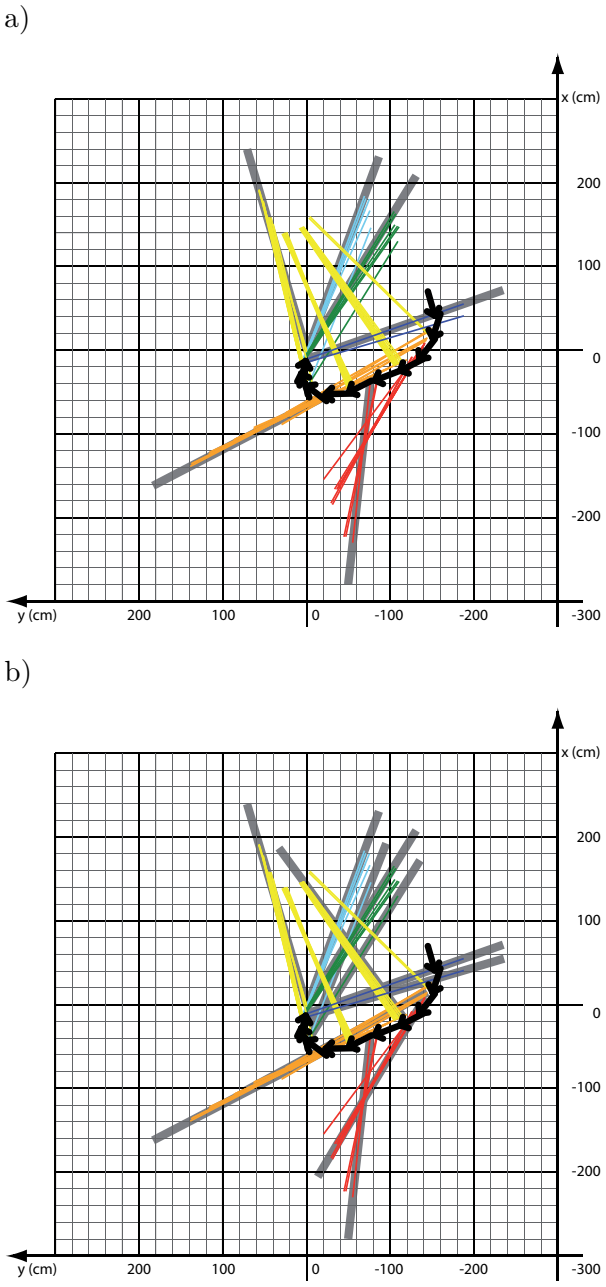


Figure 4.9: *Landmark selection strategies.* Black arrows show the relative path the robot walked during the 200 frames. Colored lines indicate the measured bearings at different times. Bold gray lines indicate the selected observations. Strategy a) *select last occurrence of each landmark type.* Strategy b) *select first and last frame of each landmark type within the last 200 frames.*

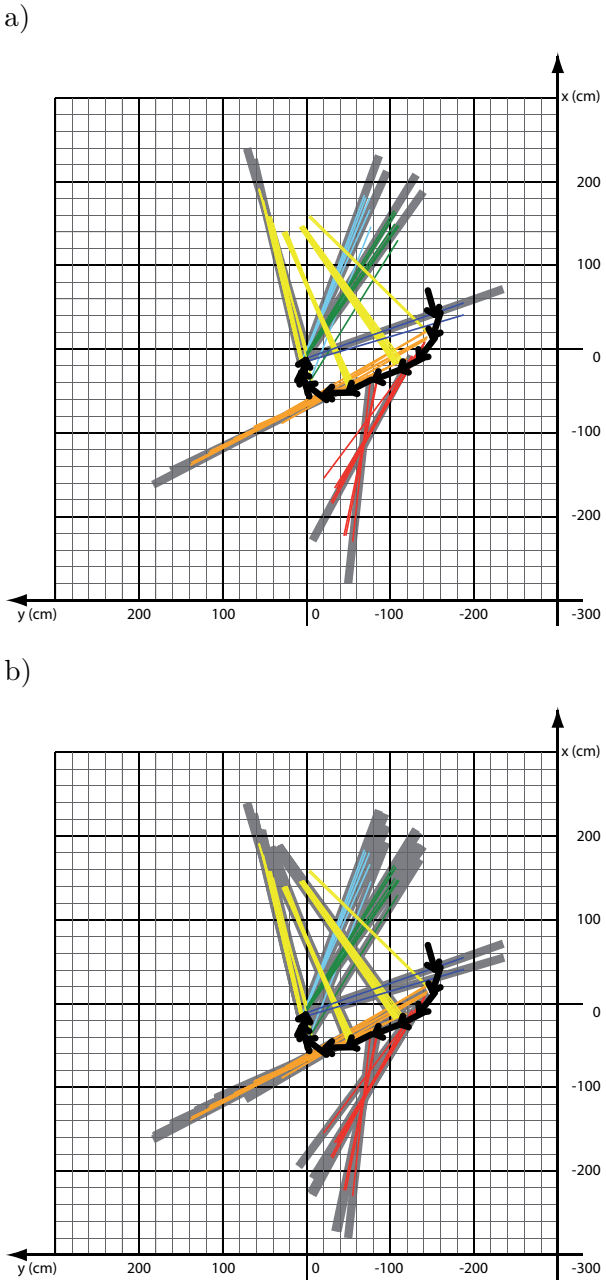


Figure 4.10: *Landmark selection strategies.* Black arrows show the relative path the robot walked during the 200 frames. Colored lines indicate the measured bearings at different times. Bold gray lines indicate the selected observations. Strategy a) *select last two occurrences of each landmark type.* Strategy b) *select all landmarks within the last 200 frames.*

selection strategy	pos. error (cm)	angle error (deg)
last of each type	9.0±16.1	0.9±0.06
last two	8.3±12.2	0.3±0.06
first and last	7.3±8.0	0.6±0.06
last 200 frames	7.3±9.6	0.4±0.06

Table 4.11: *Results of localization tests.* Position and angle error of the method for different landmark selection strategies. All results were obtained using data recorded from a simulation (80 total meters of movement, 6600 frames).

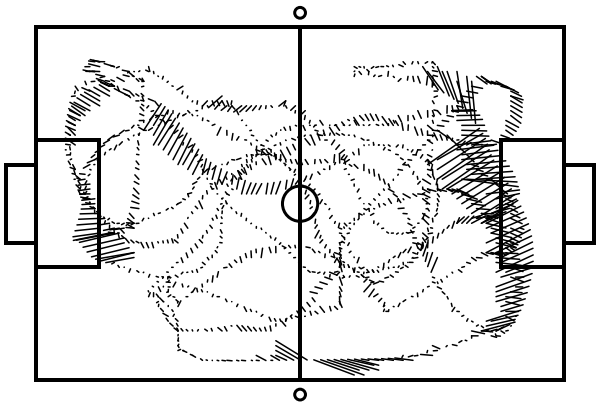


Figure 4.12: *Position deviation.* The black lines connect the ground truth position and the position obtained by the method described in this section for each fifth frame. The landmark selection strategy was *all landmarks in last 200 frames*.

last observation of each landmark type) performs very well. A minimal example illustrating this strategy's power is given in the next section.

4.3.2 Experiments on a Four-Legged Robot

In this section, the setup and the results of the experiments conducted on a real robot are described. An Aibo ERS-7 robot, built by Sony, was used for the experiments. This robot has a pan-tilt camera with a resolution of 208x160 pixels. All tests were done on a RoboCup soccer field (size: 6 m x 4 m) where the posts of the two goals and two beacons, at the half way line, can be used as landmarks. The horizontal bearings to the landmarks needed for the location approach were extracted from images and joint sensor data by the method given in [41]. One experiment shows that the method equips the robot with something like stereo vision. In another experiment, the performance of the system was analyzed.

Pseudo Stereo Vision In this experiment, the robot uses bearings to only two landmarks. This is not enough information to localize when odometry is not incorporated. The robot had

to walk over a RoboCup field from one side line to the other, observing one of the goals. During this walk, it first observes just the right goal post, then both posts, and, in the end, only the left post. As soon as the left goal post is seen, there is enough information to localize, using the first and the last observation of both goal posts (landmark selection strategy *first and last of each type* (cf. section 4.3.1)).

Figure 4.13a) shows an illustration of the robot's motion and its observations. Figure 4.13b) shows a plot of function $F(x, y)$ at the first time the left goal post is seen and the resulting path for the rest of the run. With this experiment, it was shown that incorporating odometry is a benefit for bearing-only localization. Just two beacons are sufficient to determine the position when the robot's moves change its observing position.

Generating Particle Filter Template Poses The bearing-only localization approach was developed as a replacement for the distance-based sample template generation, used by our team's particle filter self-localization [66, 65, 64]. The old method was no longer usable after a rule change that reduced the number of beacons around the field.

The method described in section 4.2.2, as a sample template generator, was added in a way described in section 4.2.4. The particle filter uses 200 particles.

To measure the quality of the improvements, an Aibo robot was steered via remote control over the soccer field in our lab, performing an s-like shape on the field. The robot's head performed the typical Aibo scan motion, looking around and searching for the ball and the landmarks. During this process, log data was recorded, containing camera images, head joint values, odometry data, and ground truth robot positions, obtained by a ceiling mounted camera. Such log-files can be played back to feed algorithms with data. The recorded log data was used to compare different parameterizations of the approach.

For this experiment, again, the landmark selection strategy *first and last of each type* (cf. section 4.3.1) was choosen.

Three localization methods were compared:

1. The particle filter implementation used by the GermanTeam in RoboCup (not using sample templates)
2. The method described in section 4.2.3 using the maximum of $F(x, y)$
3. Particle filter localization using sample templates as described in section 4.2.4.

To compare the ground truth and the determined robot position, the distance between these positions was averaged for the entire run, for each of the methods. The result was that the relative error (compared to field size) of the plain particle filter was 9.13%, the error of the maximum-method 5.13%, and the error of the particle filter using sample templates 3,18% to 3,74% (depending on the number of samples).

Figures 4.14 and 4.15 show a visualization of the path the robot walked and the paths obtained by the localization methods. The influence of the number of samples used for reseeding is given in table 4.16.

The overall result of the experiments is that without template generation there were random jumps and a large deviation from the ground truth robot pose. With sample template generation,

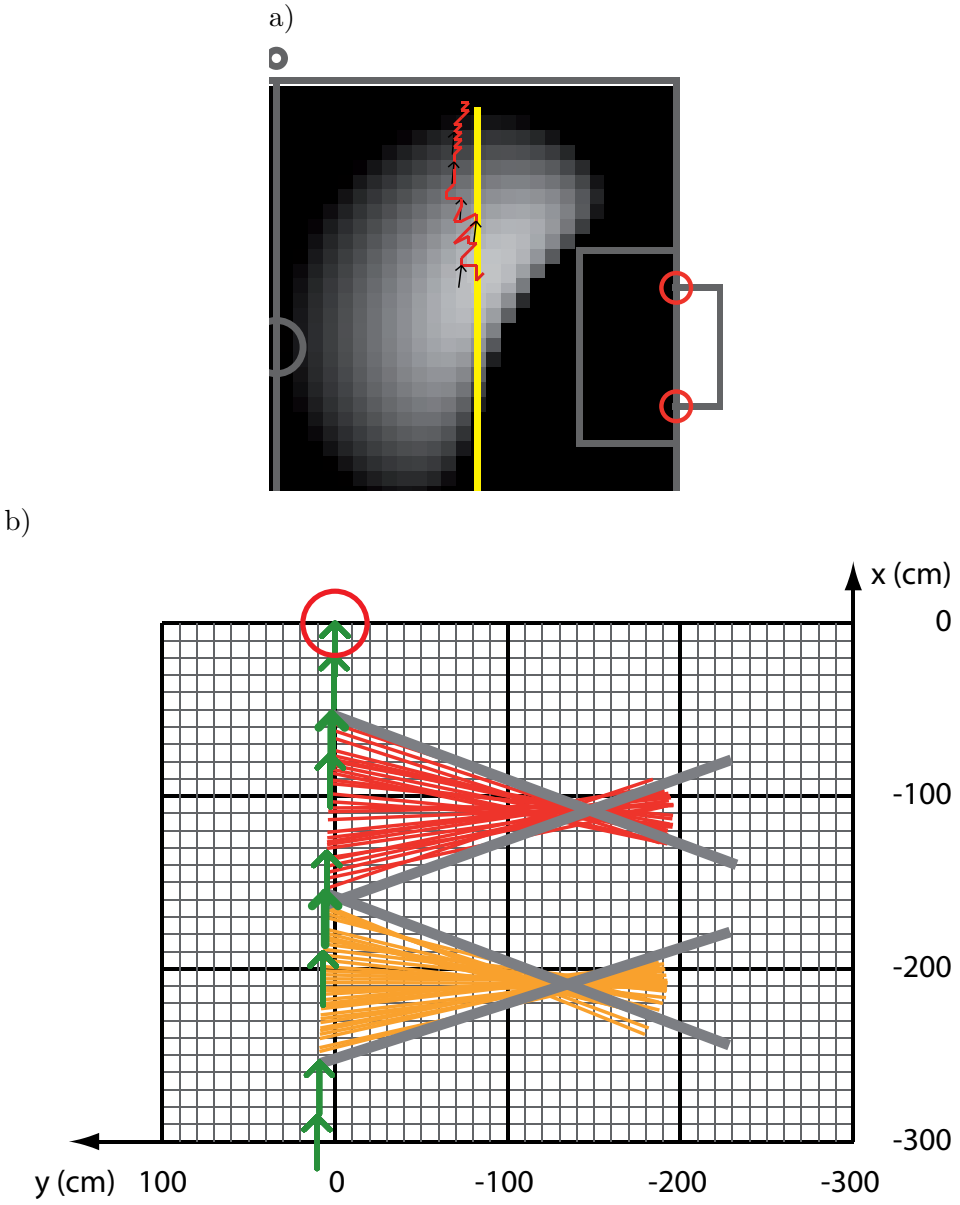
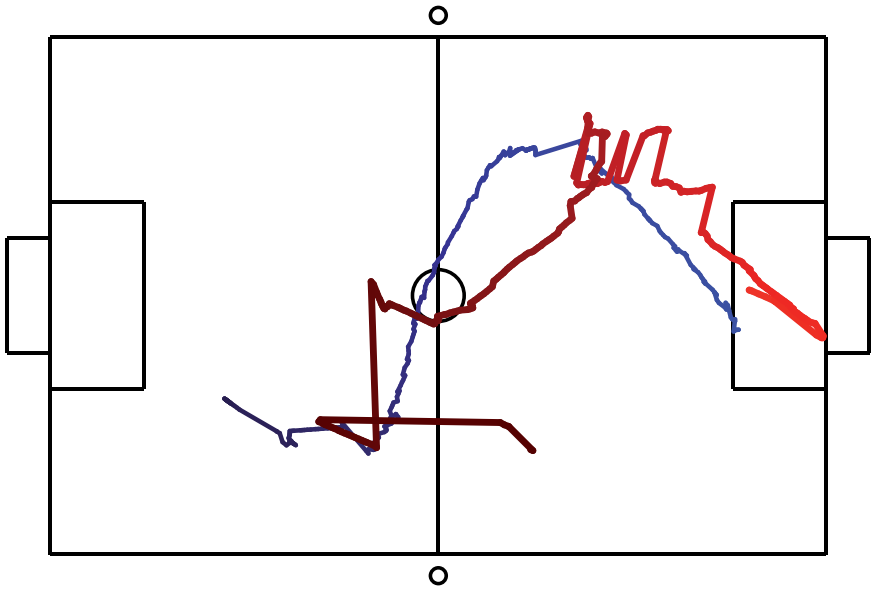


Figure 4.13: *A moving robot localizes using bearings to two landmarks.* a) Circle: current relative robot position. Arrows: odometry history. Thin lines: bearings to right and left goal post. Bold lines: bearings used for localization. b) Background: function $F(x, y)$ when the left goal post was seen. Bold line: the path the robot walked. Thin line: the localization result (since possible). Circles: the goal posts used for localization.

a)



b)

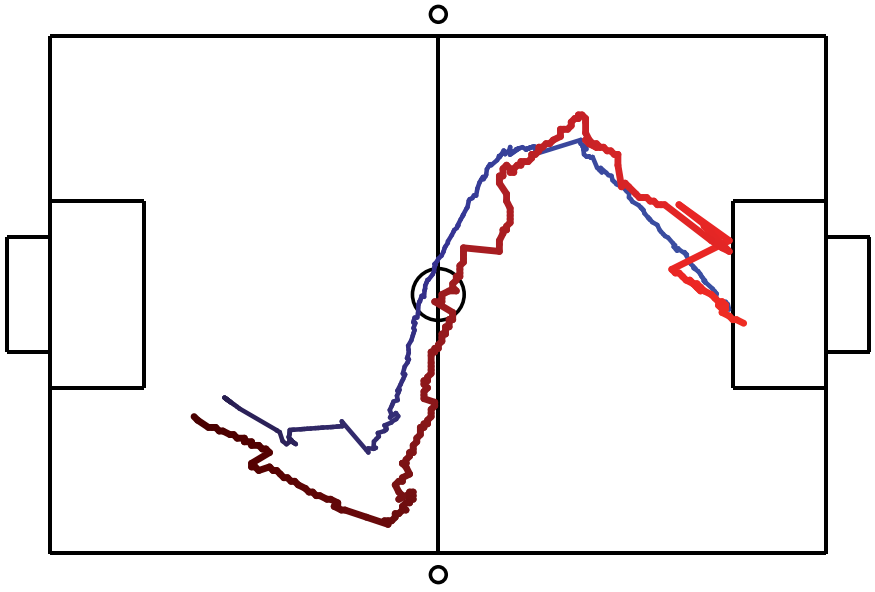
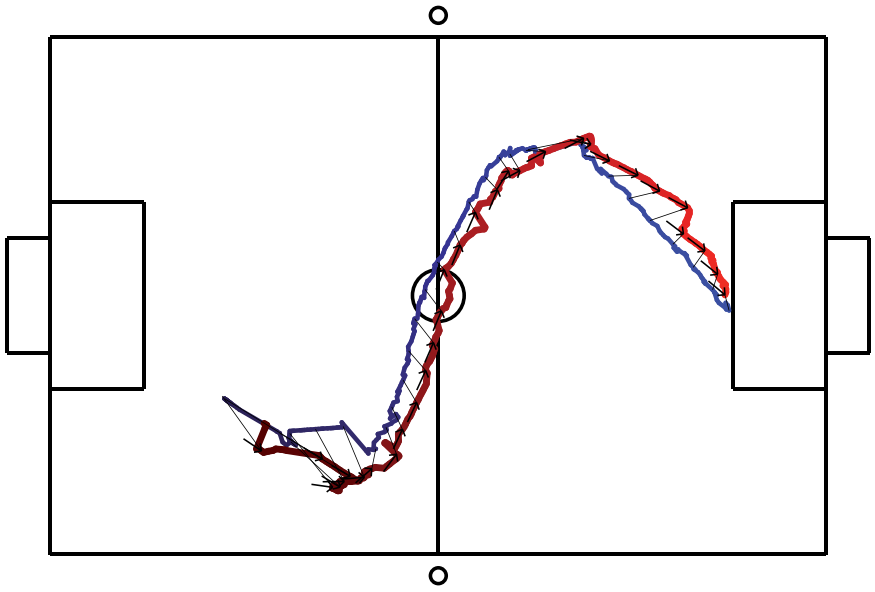


Figure 4.14: *Comparing ground truth robot position and localization result.* Blue line: ground truth robot position. Red line: self localization result. a) Particle filter, no sample templates used. b) New approach: Maximum of $F(x, y)$.

a)



b)

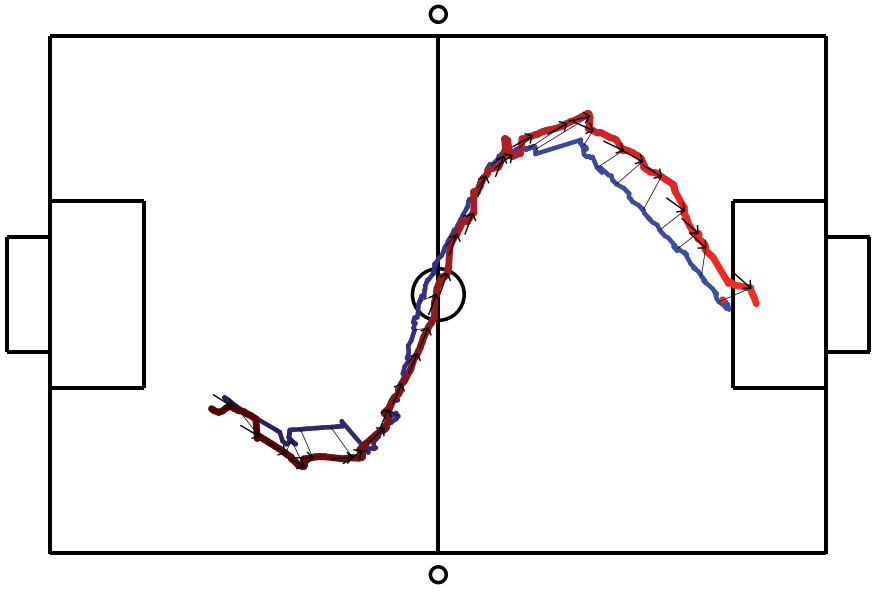


Figure 4.15: *Comparing ground truth robot position and localization result.* Blue line: ground truth robot position. Red line: self localization result. a,b) Two different runs of a particle filter using sample templates generated by the new approach.

num. of reseeded samples	0	1	2	10	20	plain
deviation in cm	54.8±21.6	21.7±13.1	19.1±13.0	19.5±12.6	22.4±17.0	30,8±20,7
deviation relative to field length	9,13%	3,63%	3,18%	3,25%	3,74%	5,13%

Table 4.16: *Results of localization tests.* In the experiment, the position obtained by the approach introduced in this chapter was compared with that obtained by the ceiling camera. The table shows the average distance between the two positions for the whole run, repeated six times. To show how reseeding influences the localization quality, the experiment was conducted with different re-sampling rates (top row). The table shows that even a single reseeded particle in each frame improves self-localization, drastically. Adding more samples has almost no effect. The last row gives the results for the plain maximum-method (no particle filter).

using one sample per frame, the resulting trajectories were smoother and closer to the ground truth.

4.4 Discussion

In this chapter, an approach for bearing-only self-localization incorporating odometry was presented. It does not depend on distance measurements which are often inaccurate, when obtained by the size of far-distant landmarks. The new approach works on an observation buffer and the robot's odometry history. The robot's position can be estimated directly, by this data, without alternating sensor and motion updates.

However, the method also provides good positions for sensor resetting in particle filters. Tests in simulation and on a real Aibo robot, with ground truth by a ceiling camera, showed the robustness of the approach. Further experiments have to show whether the localization method can cope with larger odometry errors, caused by strong physical interaction with opponents in RoboCup games.

The method does not need an internal position estimate representation updated by alternating sensor and motion updates. The observation and motion information from the robot's sensor history is processed directly. A big advantage is that no wrong model from the past can disturb the current pose estimation. However, false observations lead to a false position. Even simple strategies of landmark selection perform very well.

5 A Vision-Based Compass for Soccer Robots

A compass is a very useful sensor. It was the most important navigational instrument for many centuries. Nowadays, many autonomous robots make use of compasses, as the direction relative to the ground, is an important information for decision making. This also holds true for soccer robots. However, sometimes the use of a magnetic compass is not possible because of too many environmental influences or disturbances by the robot itself. Alternatives like gyrocompasses also can not be used in most cases.

In this chapter, a vision-based compass for soccer robots is described. The system is designed for soccer robots equipped with a camera. It uses the white lines of the soccer field as features. The output is the robot's orientation relative to the field. The system was implemented and tested on an Aibo robot, as well as on a humanoid robot.

The advantage of a precise compass is that it simplifies the localization task. It reduces a three-dimensional problem to a two-dimensional one, which increases performance. How a compass helps navigation at sea was presented in chapter 3.1.1. Additionally, bearing measurements contain more information when combined with a compass. Chapter 6 shows this.

The system introduced in this section consists of two components. Section 5.1 describes the vision system. Section 5.2 describes the rotation estimation that uses the output of the vision system and follows the *memory-based paradigm*. A discussion follows in section 5.3.

5.1 Field Line Detection

This section describes a vision system able to detect field lines in images provided by a robot's camera. The challenge is to detect field lines reliably, despite many disturbances. Such disturbances can be other robots which can be as white as the field lines (e.g Sony's Aibo, or Aldebaran's Nao). The soccer field's environment can also be very cluttered and may contain objects that look like field lines. Many soccer robots have movable cameras that look around while the robot plays, to detect, track, or scan for certain objects, like the ball or the goals. The system described here does not require any special head motion. Especially, it works with a typical robotic soccer head moving strategy; which looks at the ball most of the time and scans for goals, other robots, and landmarks, in between. This technique does not require any calibration to lighting conditions. It makes use of the contrast between the field and field lines.

Section Outline The vision system uses scan lines to analyze an image. Section 5.1.1 describes how scan lines are distributed over an image. Based on these scan lines, the vision system extracts features using a sequence of representations. These sub-steps are introduced in sections 5.1.3 to 5.1.10.

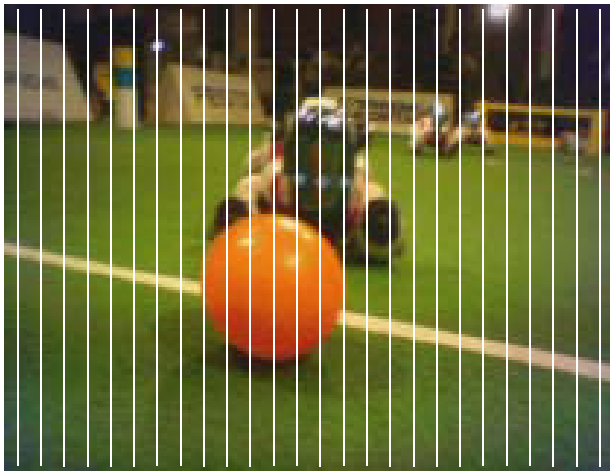


Figure 5.1: *Scan lines.* An image captured by an Aibo camera with vertical scan lines.

5.1.1 Utilizing Scan Lines for Image Analysis

The image is scanned vertically using many scan lines. Vertically means perpendicular to the horizon. If the robot's cinematic chain ensures that the horizon is almost parallel to the upper and lower image borders, the scan lines can also be placed parallel to the image's left and right borders to simplify implementation. Figure 5.1 shows how the scan lines are distributed over an image. The distance between two neighboring scan lines is 8 pixels, in this case.

The width of the images provided by the Aibo is 208 pixels which results in 26 scan lines per image. As the Aibo's horizontal opening angle is 57°, the angular spacing between two neighboring scan lines is approximately 2°. This spacing is appropriate for field line detection, as well as for other scanning purposes, like ball and opponent detection. If the camera used has a different resolution or a different opening angle, the distance (in pixels) between the scan lines should be adapted such that the resulting angular spacing is approximately 2°, to get similar results.

5.1.2 Layered Representations

To detect field lines, an image is analyzed along scan lines as described above. The vision system uses a lot of representations in a sequential order of refinements. With each refinement, those representations contain more accurate information about field lines. Figure 5.2 shows this sequence of representations.

The remainder of this section contains a detailed description of all those intermediate representations. Section 5.1.3 shows how brightness changes in the image are extracted. Section 5.1.4 shows how scan line parts are labeled, depending on such brightness changes. Section 5.1.5 describes how candidates of pixels on scan line edges are generated and pre-filtered. While until this step, just vertical relations are analyzed, from the next step on, also horizontal relations will be taken into account. Section 5.1.6 describes how a *neighborhood graph* is created which represents all field line candidates in an image. Section 5.1.7 describes how this graph is filtered.

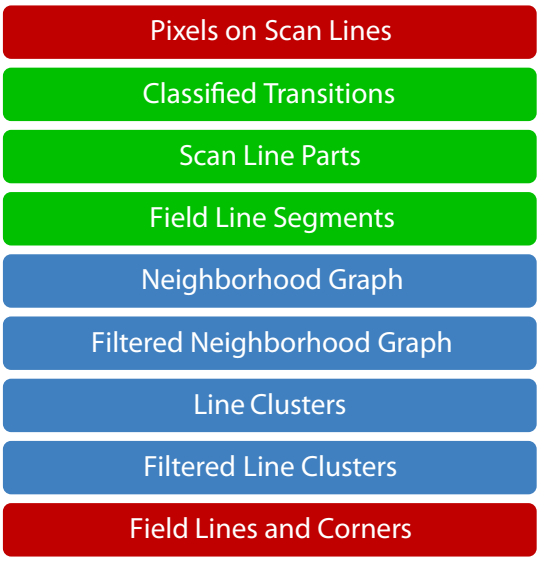


Figure 5.2: *Sequence of representations.* All representations used while detecting field lines. The source representation is pixels on scan lines. The target representation is field lines and corners, relative to the robot. The first three representations (green background) are scan-line based. This means only vertical relations of pixels and detected objects are used. Starting from the fourth representation (blue background) horizontal relations are also taken into account. This distinction is important when the complexity of the system is analyzed. The subsections in this section describe how the representations shown are calculated in this sequence.

Sections 5.1.8 and 5.1.9 show how, from the graph representation, different subgraphs are extracted which belong to different field lines. Finally, section 5.1.10 describes how field lines and corners are extracted.

5.1.3 Classified Transitions

Each scan line is divided into segments based on the y-channel intensity of the image, which represents the brightness. If the camera provides images in a different color space, the brightness has to be calculated accordingly. Figure 5.3b) shows an intensity diagram for the scan line highlighted in figure 5.3a). For segmentation, the scan lines are analyzed from bottom to top.

Each transition from one pixel to the next is classified as *up*, *down*, or *neutral*, based on an appropriate threshold t. When the intensity difference from one pixel to the next is above the threshold, the transition is classified as *up*, when it is below the negation of the threshold, it is classified as *down*. In all other cases, the transition is classified as *neutral*.

The threshold should be chosen such that field lines in the image cause transitions between pixels to be classified as *up* or *down*. The threshold should not be too low to avoid noise on the field to be classified as *up* or *down*. Likewise, the threshold should not be too high to classify all transitions caused by a field line as either *up* or *down*.

Figure 5.4 shows the result of such a classification for a good, a too low, and a too high threshold. A threshold of $t = 10$ proved to be good for almost any lighting condition on any field, at different competition sites, as well as in different labs. So, the threshold t is considered a constant that doesn't need calibration. Figure 5.5 shows the intensity diagram of a single scan line and the corresponding transition classification.

5.1.4 Scan Line Parts

Based on the transition classifications, the scan lines are partitioned. Let $t_1, t_2, ..., t_n$ be the transition sequence along a scan line with $n+1$ pixels. A subsequence $t_i, ..., t_j$ with $1 \leq i$, $i \leq j$, and $j \leq n$ is called a *scan line part* when all transitions belong to the same class (*up*, *down*, or *neutral*) and the transitions before and after the subsequence belong to a different one:

$$\begin{aligned} &class(t_a) = class(t_b) \quad (\forall (a,b) : i \leq a \leq j \wedge i \leq b \leq j) \\ &class(t_i) \neq class(t_{i-1}) \\ &class(t_j) \neq class(t_{j+1}) \end{aligned}$$

With this notion of a *scan line part*, each scan line falls into a sequence $s_1, s_2, ..., s_m$ of m parts. Each part s_i has a class with $class(s_i) \in \{up, down, neutral\}$, a start point, an end point, a minimal intensity, and a maximal intensity. The starting point is the starting point of the first transition and the end point is the end point of the last transition. Therefore, the end point of one part is identical with the starting point of the next part. Figure 5.6a) shows such *scan line parts* in an intensity diagram.

With the method described so far, a field line crossing a scan line can be detected by finding an *up* part followed by a *neutral* part and a *down* part. For a distant field line, which has a

a)

b)

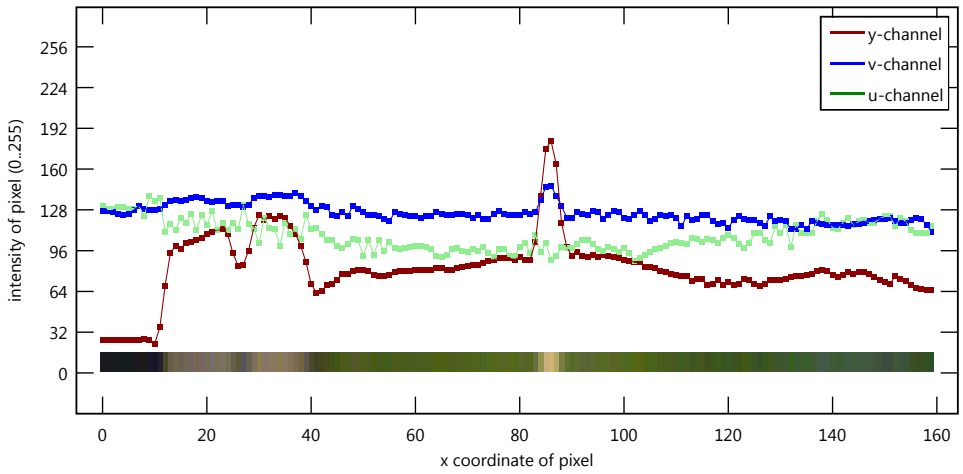


Figure 5.3: *Intensity diagram.* a) Scan line. b) Intensities of all three channels along that scan line. The colored bar at the bottom shows the color of each pixel along the scan line. The peak near x-coordinate 85 is caused by the field line. (The x-axis is vertical, the top-most pixels have x-coordinate 0 and the-bottom most pixels 159).

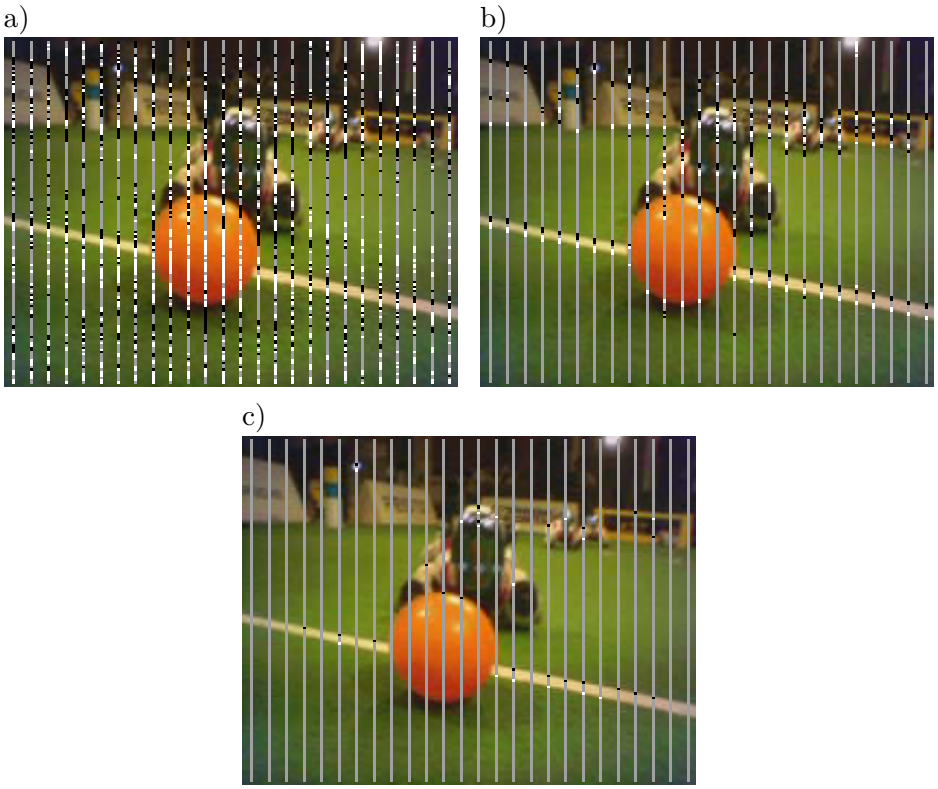


Figure 5.4: *Transition classification.* Transitions between two adjacent pixels are classified, based on the difference of intensities and the threshold t. In all above images, the transitions labeled gray are labeled *neutral*, white are the *up*-transitions and black the *down*-transitions. The scan lines are analyzed from bottom to top. a) $t = 1$: Noise on the field is classified as *up* or *down*. b) $t = 10$: All transitions at field lines are classified as *up* or *down*. c) $t = 50$: Only some transitions at field lines are classified as *up* or *down*.

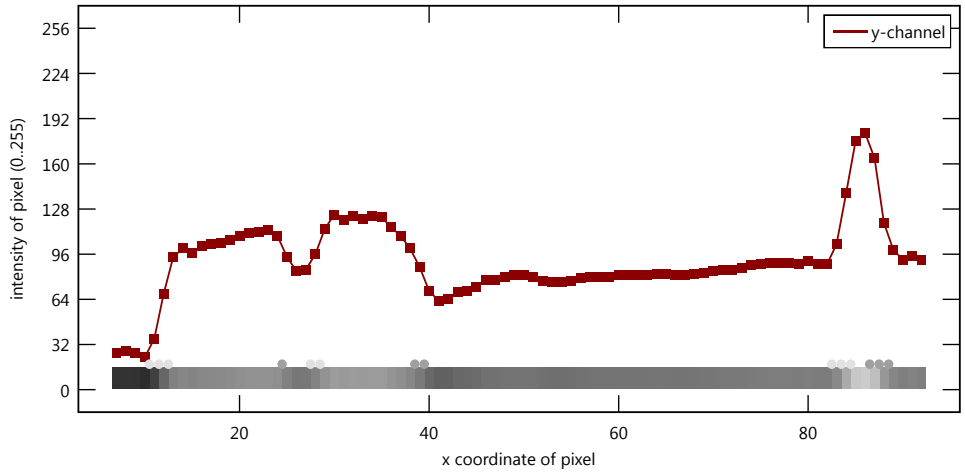


Figure 5.5: *Intensity diagram and transition classification.* The graph shows the intensity of the upper part of the scan line highlighted in figure 5.3a). The different shades of gray in the bar, at the bottom, shows the brightness of the pixels along the scan line. To classify the transitions, the scan line was analyzed from bottom to top, which corresponds to from right to left in this figure. The dark gray circles mark transitions classified as *up* and the light gray circles mark transitions classified as *down*.

small width in the image, the *neutral* part can be missing. However, also objects other than the field line can cause such a pattern of parts.

A simple filter can reduce the number of false detections when a field line was detected earlier on the same scan line. For this filter, the intensity range $range(s)$ of a part s is defined to be the difference between the maximum and minimum intensity of that part. The filter is applied while scanning all parts for a *up* - *down* sequence. It memorizes the maximum *intensity range* of all *up* and *down* parts examined on that scan line so far and marks parts with an *intensity range* lower than half the maximum range as *neutral*. This filter relies on two facts: First, when an image is scanned from bottom to top the first high contrast found on a scan line is usually caused by a field line. Second, field lines cause the highest contrast compared to other objects.

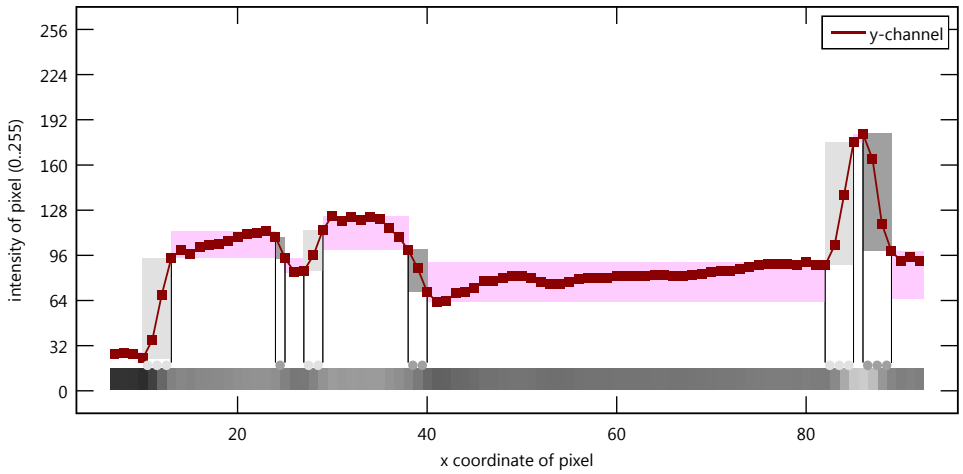
The filter can be understood as a calibration of the typical *intensity range* caused by a field line. Note that this calibration is re-initialized for each scan line. Figure 5.6b) shows the parts shown in figure 5.6a) after the filter has been applied.

5.1.5 Field Line Segments

The next step to detecting field lines is finding *field line segments*, these are based on the segmented scan lines. The *neutral* segments are not considered. Let $s_1, ..., s_n$ be the sequence of *up* and *down* segments along a single scan line. With condition

$$C_0(i) := class(s_i) = up \wedge class(s_i + 1) = down \tag{5.1}$$

a)



b)

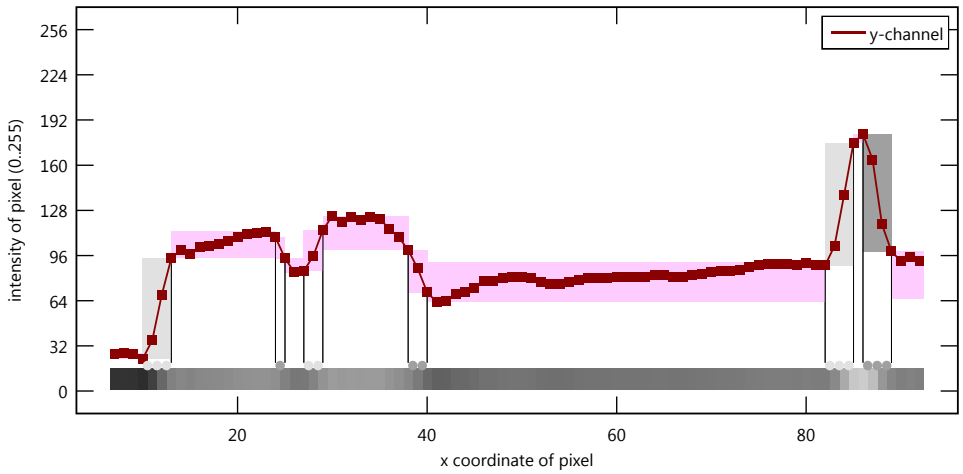


Figure 5.6: *Intensity diagram and scan line parts.* The graph shows the intensity of the upper part of the scan line highlighted in figure 5.3). The rectangles show the according scan line parts. The rectangle width represents the length of the *scan line parts*. The rectangle height represents the intensity range. The *up* parts are dark gray, the *down* parts light gray, and the *neutral* parts pink. a) shows the parts created from transition sequences with the same classification (see the circles at the bottom). b) shows the parts after the filter described in this section was applied. Note that in b) there are more *neutral* parts.

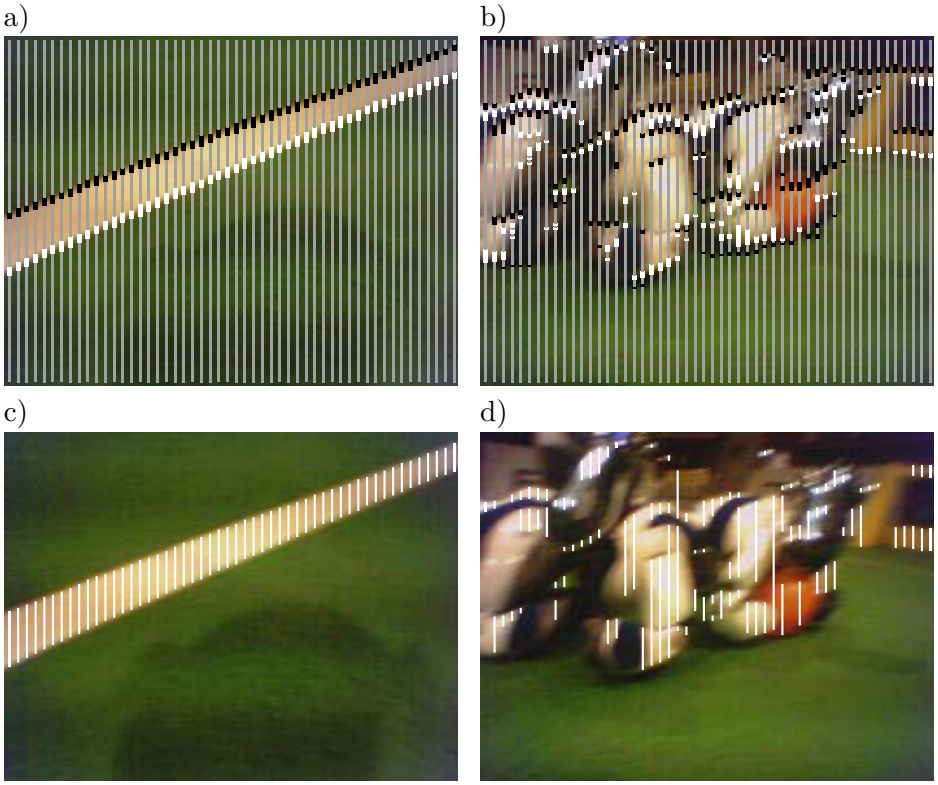


Figure 5.7: *Scan line segments and field line segment candidates.* a) and b) Images of a field line and other robots. Scan line segments are shown gray (*neutral)*, white (*up)*, and black(*down).* c) and d) Field line segment candidates. The field line causes the creation of correct candidates. The other robots also cause a lot of candidates.

being true for all i where s_i is an *up* segment followed by a *down* segment, a set C of all such *up-down* pairs can be defined:

$$F_0 := \{(s_i, s_{i+1}) : C_0(i)\}. \tag{5.2}$$

These pairs are called *field line segments.*

Each such *field line segment* (s_i, s_{i+1}) has a start coordinate (the *up* segment's start point) and an end coordinate (the *down* segment's end point). For later usage, the Euclidian distance between the start and the end coordinate is denoted with $width(s_i, s_{i+1})$. Figure 5.7 shows sample images with segmented scan lines and *field line segments* created as described above. As shown by this figure, *field line segments* are created for all intersections of field lines and scan lines. However, such *field line segments* are not only created for field lines. Also other objects, like robots, can cause pairs of *up-down* segments along scan lines. Therefore, more criterions are used to distinguish field lines from other objects. These criterions are formalized in additional conditions for pairs of *up-down* segments (s_i, s_{i+1}). The conditions are defined

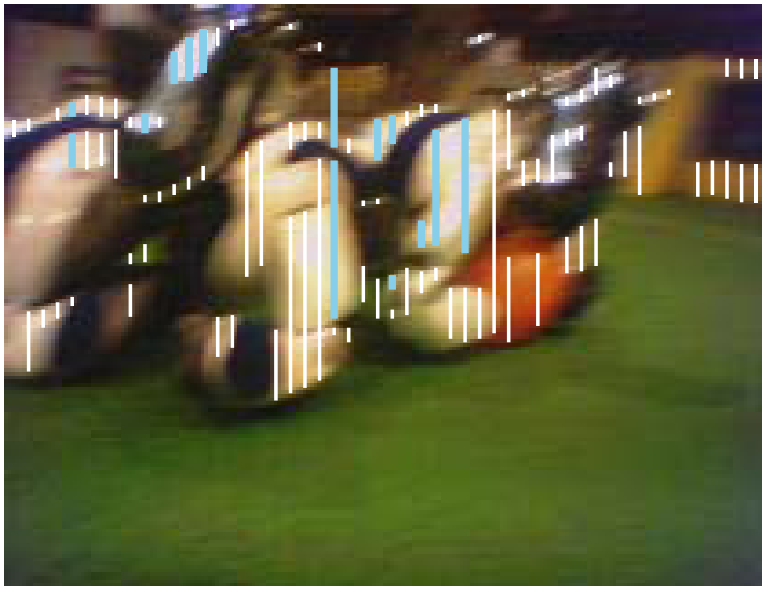


Figure 5.8: *Conditions to remove false positives.* Condition 1: *The line segment width decreases towards the horizon.* The sky-blue segments are longer than another segment below on the same scan line. This is why they are removed by C_1.

as Boolean functions $C_m(i)$ with the index of the *up* segment as a parameter. With such an additional condition the set of *field line segments* can be specified as

$$F_1 := \{(s_i, s_{i+1}) : C_0(i) \wedge C_m(i)\}.$$

Below, four such additional conditions are introduced.

Condition 1 *The line segment width decreases towards the horizon.* The law of perspective says: field lines farther away from the robot appear smaller in the image and closer to the horizon. As a consequence, the field line width decreases, along a scan line, from bottom to top. This gives the first additional condition:

$$C_1(i) := \forall 0 \leq k < i : width(s_i, s_i + 1) \leq width(s_k, s_k + 1). \tag{5.3}$$

Figure 5.8 shows which false positives can be filtered using this condition.

Condition 2 *There is no* down *segment with a high intensity range outside an* up-down *pair below* s_i. Large decreases in the brightness, along a scan line, without a preceding increase are

considered a disturbance and suppress further field line detections, along this scan line:

$$\begin{aligned} C_2(i) := \forall 0 \leq k < i : & \; class(s_k) = up \vee \\ & class(s_k) = down \wedge class(s_{k-1}) = up \vee \\ & range(s_k) < 30. \end{aligned}$$

With this condition bright robot parts with a strong shadow below can be avoided as being recognized as field lines, as well as bright robot parts situated above dark parts. Figure 5.9a) gives an example image with segmented scan lines, figure 5.9b) shows the intensity diagram for this example.

Condition 3 *For* (s_i, s_{i+1}) *and all pairs below, the* up *and the* down *segment have a similar intensity range.* As for field lines, the brightness change on both sides is similar, condition

$$C_3(i) := \forall 0 \leq k \leq i : \neg C_0(k) \vee 0.5 \leq \frac{range(s_k)}{range(s_k + 1)} \leq 2$$

can be used to remove other objects. $C_0(k)$ is the condition defined in (5.1) and is true when (s_k, s_{k+1}) is an *up-down* pair. Figure 5.10a) gives an example image with segmented scan lines. Figure 5.10b) shows the intensity diagram for that example.

Condition 4 *The width of* (s_i, s_{i+1}) *matches the expected width.* If the camera's position and rotation relative to the ground are known, the expected field line width in the image can be calculated based on its position in the image. Figure 5.11a) shows the expected field line width for different positions in the image.

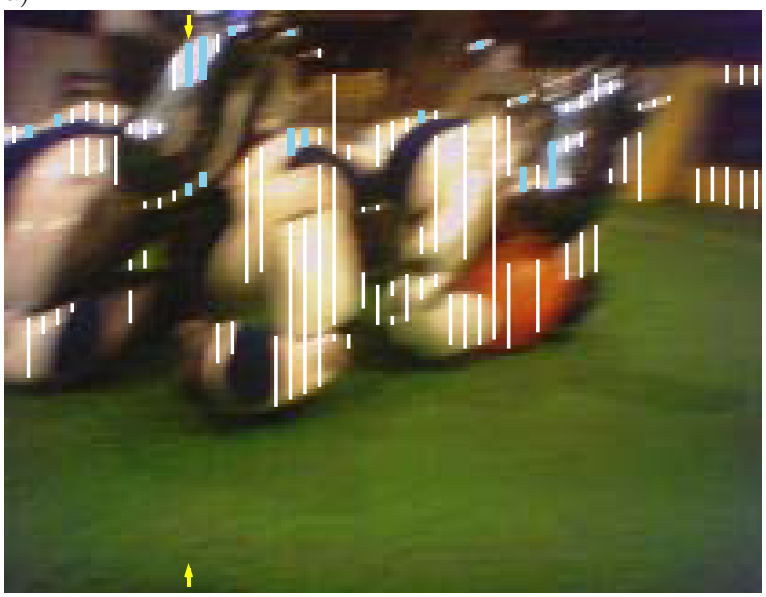
Condition 1, given in (5.3), compares the relative width of candidate segments. It can be applied without knowing the head's position relative to the ground. As long as it is known which image edge is the upper, even the head's rotation is not needed for that condition to be checked. However, when the camera's position and rotation relative to the ground are known, the width of field line segment candidates can be compared to the expected width, leading to the additional condition

$$C_4(i) := width(s_i, s_{i+1}) \leq width_{expected}(s_i, s_{i+1}) * c,$$

where $width_{expected}(s_i, s_{i+1})$ is the expected width for the position of the pair (s_i, s_{i+1}). The constant c is used to compensate for the fact that scan lines aren't necessarily perpendicular to the field lines. The value of $c = 2.5$ proved to be adequate allowing the scan lines to intersect the field lines at an angle of up to 70° as $\arctan(2.5) \approx 70$.

Note that condition C_4 only restricts the maximum width of segments (s_i, s_{i+1}). For simplicity, the expected width is also approximated by an upper estimate. To calculate $width_{expected}(s_i, s_{i+1})$, the center point of (s_i, s_{i+1}) is projected to the ground plane. Then the distance between the camera and this projected point is used to estimate the field line width in the image. It is assumed that the plane containing the field line is perpendicular to the camera's optical axis. Of course this only holds true when the camera is looking down (the optical axis is perpendicular

a)



b)

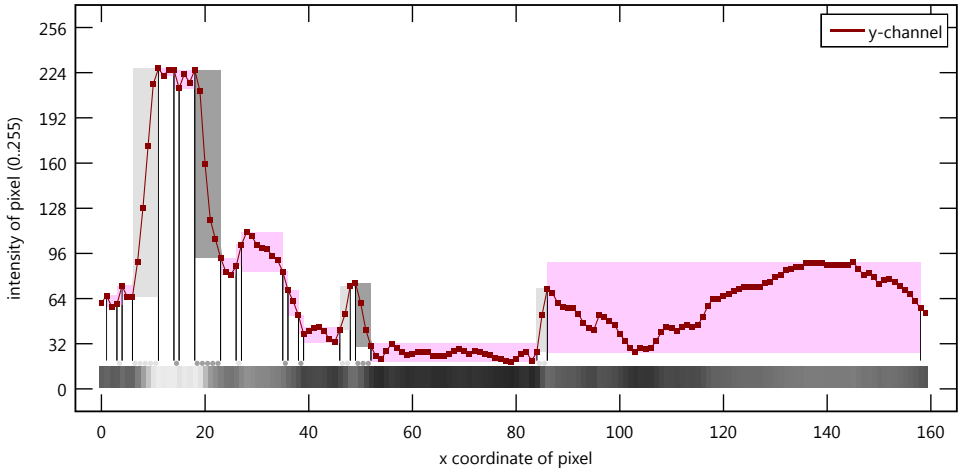


Figure 5.9: *Conditions to remove false positives.* a) Condition 2: *There is no* down *segment with a high intensity range outside an* up-down *pair below* s_i. Below the sky-blue segments, there is a large decrease in brightness not paired with an increase just before. This is considered a hint for a robot's shadow or some other disturbance and causes C_2 to be false for the sky-blue segments. b) Intensity diagram for the scan line marked with yellow arrows in a). The *down* segment near $x = 85$ causes the pairs of *up* and *down* segments near $x = 15$ and $x = 50$ not to be considered as field lines, see the sky-blue segments in a)

a)

b)

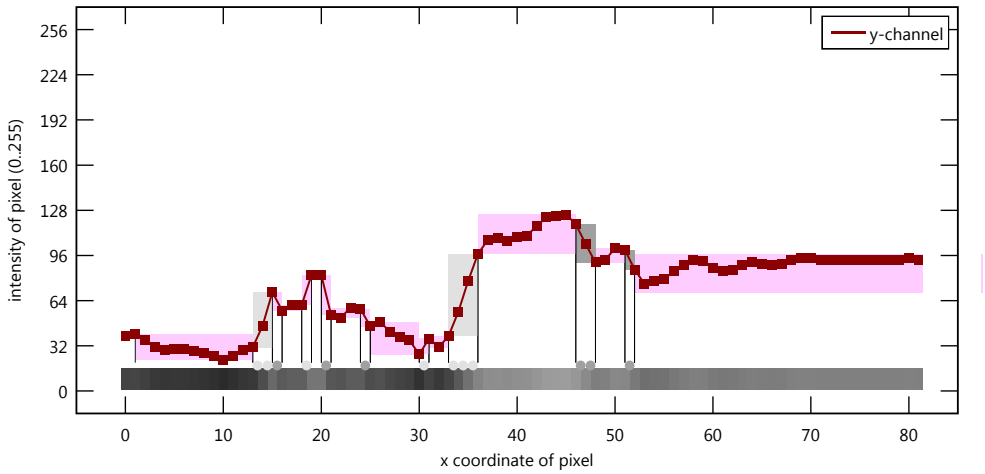


Figure 5.10: *Conditions to remove false positives.* Condition 3: *For* (s_i, s_{i+1}) *and all pairs below, the* up *and the* down *segment have a similar intensity range.* a) The sky-blue segments can not be field lines as the intensity range of the *up* segment differs too much from the intensity range of the *down* segment. b) Intensity diagram for the upper part of the scan line marked with yellow arrows in a). The *up* segment around $x = 47$ and the *down* segment around $x = 35$ differ too much in their intensity range to be considered field lines.

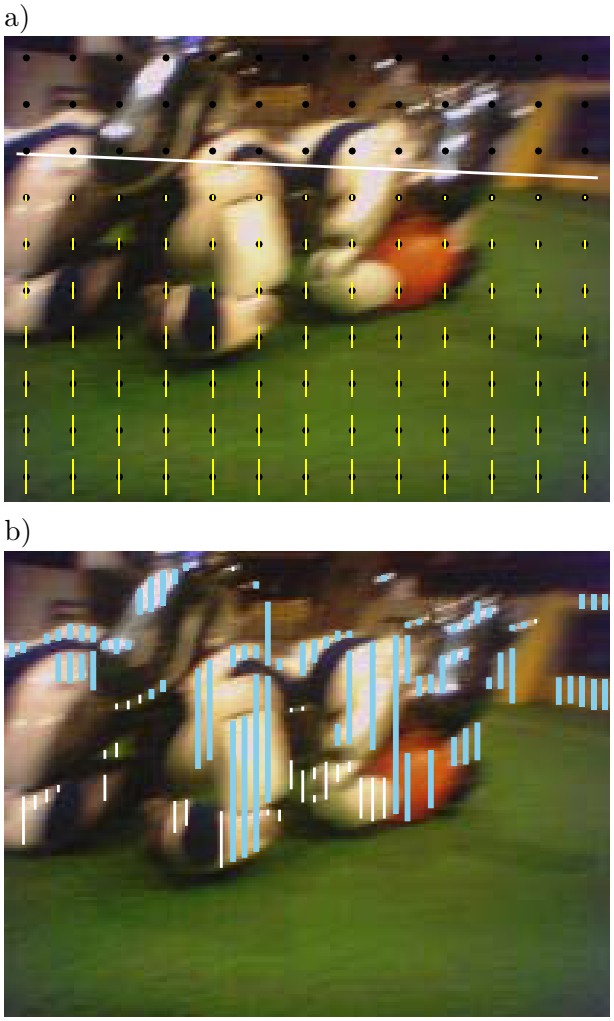


Figure 5.11: *Conditions to remove false positives.* Condition 4: *The width of* (s_i, s_{i+1}) *matches the expected width.* a) Expected field line width. The length of the yellow lines illustrates the expected field line width at the corresponding black dots. b) The sky-blue segments can't be caused by field lines as they don't match the expected field line width at the respective positions.

to the field). However, all other configurations lead to a smaller field line width in the image. So, the expected size, calculated this way, can be used as an upper bound.

Applying all Conditions Let F be the set of *field line segments* defined using all those constraining conditions:

$$F := \{(s_i, s_{i+1}) : C_0(i) \wedge C_1(i) \wedge C_2(i) \wedge C_3(i) \wedge C_4(i)\}. \tag{5.4}$$

Then, F contains less false positives than the set F_0, defined in (5.2), without losing real *field line segments*. Figure 5.12 gives an example. While most of the false positives can be filtered based on processing single scan lines, not all false positives can be excluded in the way described above.

The next section describes how the spatial configuration of detected *field line segments* is used to find the actual field lines and to determine their position and extension in the image.

5.1.6 Neighborhood Graph

To calculate the *neighborhood graph*, the *field line segments* of all scan lines are incorporated. It is the first in the series of representations (cf. 5.1.2) for whose calculation a horizontal relation is used.

Before defining the *neighborhood graph*, some notation is introduced: in accordance with the notation common to graph theory, and given for example in [8], let a *directed graph* G be an ordered pair of disjoint sets (V, E) such that E is a subset of the set $V \times V$ of ordered pairs of V. If G is a graph, then $V = V(G)$ is the set of vertices of G, while $E = E(G)$ is the set of edges of G.

Furthermore, let $v \in F$ be a segment in the set of *field line segments* and let $I(v)$ denote the index of the scan line that contains v, with the index 0, for the left-most scan line. Then the function

$$N_1(v) := \arg\min_w(|v, w|), w \in F, I(w) = I(v) + 1$$

denotes the closest neighbor to v on the next scan line to the right. The function

$$N_2(v) := \arg\min_w(|v, w|), w \in F, I(w) = I(v) + 1, w \neq N_1(v)$$

denotes the second closest neighbor to v on the next scan line to the right.

The *neighborhood graph* $N = (V, E)$ is a directed graph where the set of vertices $V(N)$ is equal to the set F of *field line segments*:

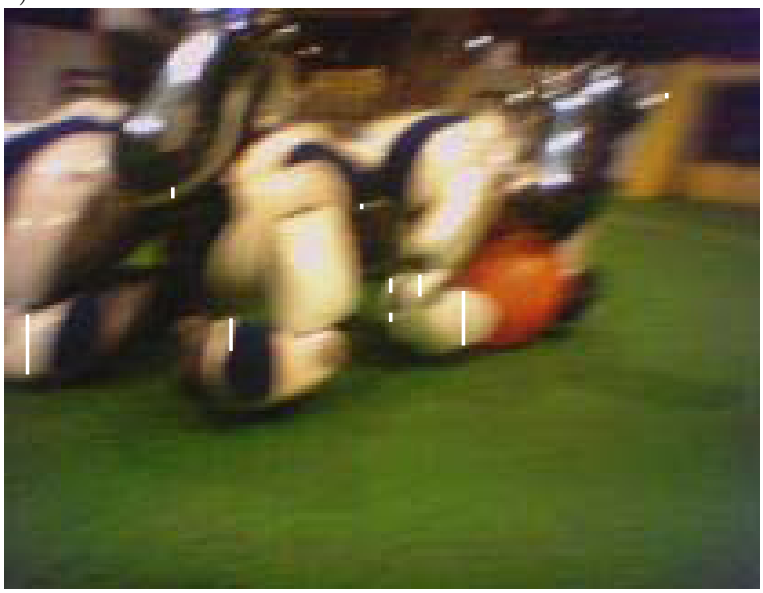
$$V(N) := F.$$

The set of edges $E(N)$ is defined using the neighbor functions introduced above:

$$E(N) := \{(v, w) \in V \times V : w = N_1(v) \vee w = N_2(v)\}.$$

Therefore, for each *field line segment* v the set of edges E contains one directed edge to the

a)



b)

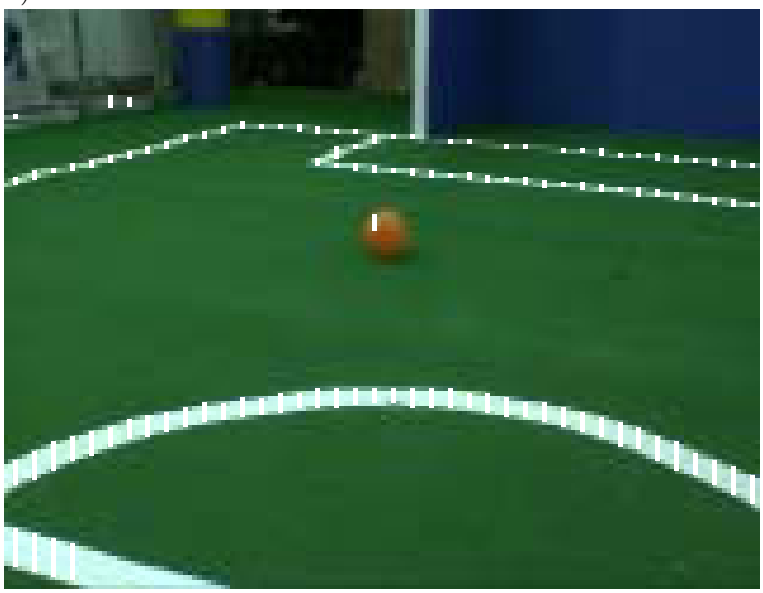


Figure 5.12: *Field line segments.* The *field line segments* as defined by (5.4) is represented by the white lines in the images. To obtain these sets, all conditions described in this section were applied. a) Some false positives remain. These are filtered by further processing. b) Image with a lot of field lines. Almost every intersection of a scan line and a field line caused a *field line segment*. There are only a few false positives.

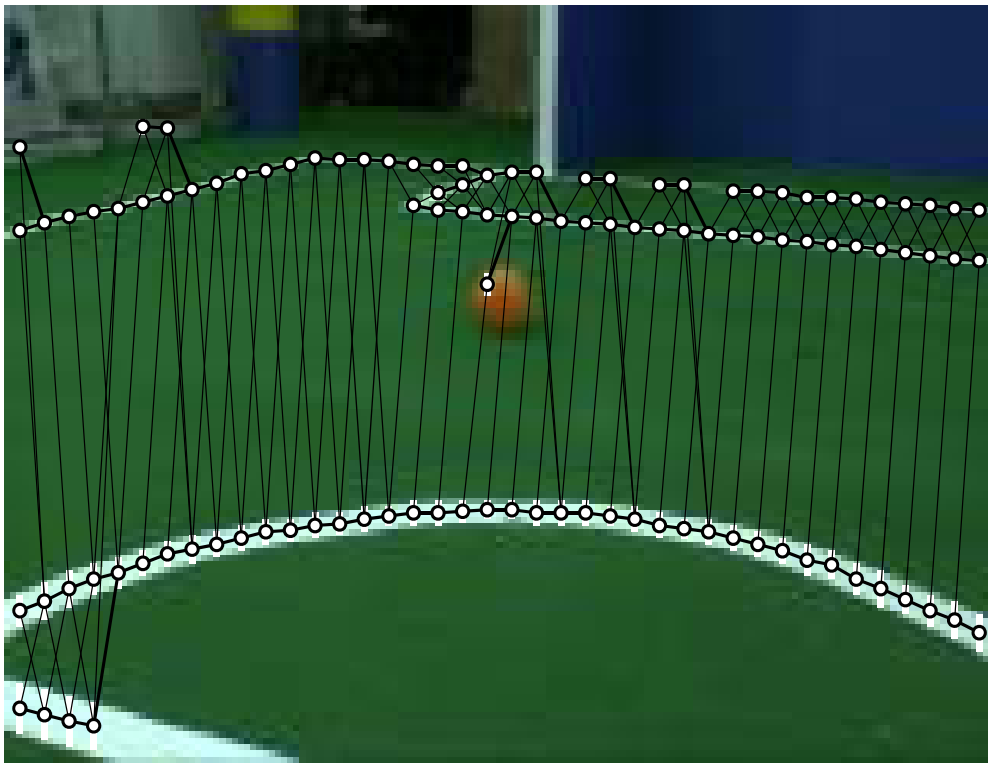


Figure 5.13: *Neighborhood graph.* The white lines represent the *field line segments*. The vertices of the *neighborhood graph* are represented by white circles, the edges by black lines. Note that the *neighborhood graph* is a directed graph. As all edges are directed from left to right, the direction is not represented graphically to avoid the figure becoming cluttered. Thick lines represent the edges connecting a vertex v with its closest neighbor $N_1(v)$ on the next scan line to the right, thin lines show the edges to the second closest neighbor $N_2(v)$.

closest and another one to the second closest neighbor on the next scan line to the right. If the scan line right of the one containing v contains no *field line segments*, there is no edge leaving v. If the scan line right of the one containing v contains just one *field line segment* w, there is just the edge (v, w) leaving v. Figure 5.13 shows the *neighborhood graph* $N = (V, E)$ for the sample image introduced in figure 5.12b).

Properties of Neighborhood Graphs As can be seen in figure 5.13, some of the neighborhood graph's edges cover field lines in the corresponding image and some don't. Under the assumption that there are *field line segments* for all intersections of scan lines with field lines, all field lines are covered by edges of the *neighborhood graph*. If for some field line parts the creation of *field line segments* failed, these parts are not covered by edges.

Note that for the majority of vertices v, the edge $(v, N_1(v))$ covers a field line and the edge $(v, N_2(v))$ does not cover one. However, $(v, N_1(v))$ might not cover a field line in some cases, for

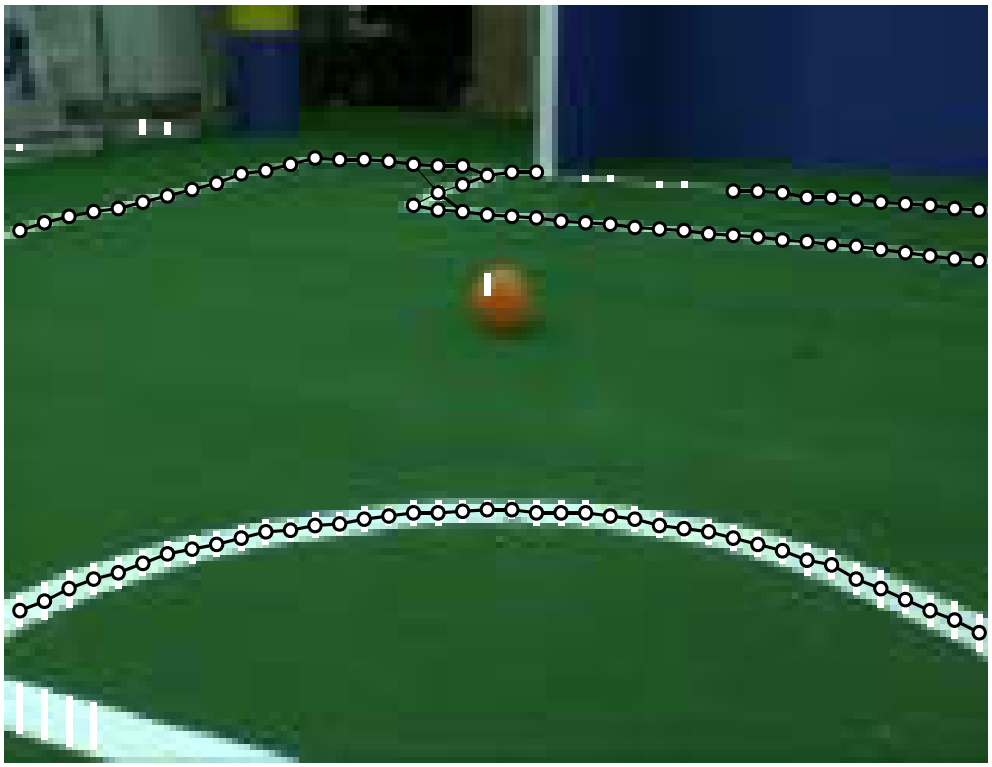


Figure 5.14: *Filtered neighborhood graph.* The *filtered neighborhood graph* N' represented by the white circles and the black edges was created based on the *neighborhood graph* N, shown in figure 5.13. The graph N' contains only those edges from N contained in at least one *Aligned Path* of length 3. Nodes not contained in any *Aligned Path* are removed. Note that also the *filtered neighborhood graph* is a directed graph.

example, when v or $N_1(v)$ is not on a field line or when v and $N_1(v)$ are on different field lines. There are also cases where the edge $v, N_2(v)$ covers a field line. This happens when the first neighbor already covers another field line (e.g. at field line corners) or a false positive. Adding edges to a third neighbor does not increase the coverage of field lines by edges, as there are no configurations of field lines where more than two field lines meet each other at a single point.

5.1.7 Filtered Neighborhood Graph

The *neighborhood graph* is filtered in order to remove those edges which do not cover a field line in the image. To define the *filtered neighborhood graph* $N'(V', E')$, some preliminary definitions are made.

As common in graph theory, a *directed path* is a directed graph $P(V, E)$ of the form

$$V(P) = \{v_0, ..., v_l\}, E(P) = \{(v_0, v_1), ..., (v_{l-1}, v_l)\}.$$

Figure 5.15: *Filtered neighborhood graph containg edges that cover parts of an image without a field line.* This is a very rare case. However, it does not cause the creation of a false *field line percept*, as no *line cluster* is created (cf. section 5.1.8).

The length $l(P) := |E(P)|$ is defined as the number of edges in P.

A graph $G' = (V', E')$ is called a *subgraph* of $G = (V, E)$ if $V' \subseteq V$ and $E' \subseteq E$. This relation can be denoted by $G' \subseteq G$.

Additionally, let $\vec{c}(v)$ define the vector that describes the center coordinates of a *field line segment* v in the image. Then, the angle between a pair of adjacent edges between *field line segments* $(v_i, v_{i+1}), (v_{i+1}, v_{i+2})$ is defined as:

$$\angle(v_i, v_{i+1}), (v_{i+1}, v_{i+2}) := \angle\vec{c}(v_i) - \vec{c}(v_{i+1}), \vec{c}(v_{i+1}) - \vec{c}(v_{i+2}).$$

Let P be a path with the set of vertices $V(P) = \{v_0, ...v_l\} \subseteq F$ being *field line segments*, let the path have at least two edges: $l(P) \geq 2$. Then the function $a(P)$ is defined as:

$$a(P) = \begin{cases} true & , \quad \forall 0 \leq i \leq l(P) - 2 : \angle(v_i, v_{i+1}), (v_{i+1}, v_{i+2}) < 30° \\ false & , \quad otherwise \end{cases}$$

A path P with $a(P) = true$ is called an *Aligned Path*.

The *filtered neighborhood graph* $N' = (V', E')$ is a subgraph of the *neighborhood graph* $N = (V, E)$ with

$$E' = E(N') := \{e \in E(N) : \exists P \subseteq N(e \in E(P) \wedge a(P) \wedge l(P) \geq 4)\}$$

and

$$V' = V(N') := \{v \in V(N) : \exists w \in N((v, w) \in E' \vee (w, v) \in E')\}.$$

The graph N' is obtained by deleting all edges from N not contained in an aligned path with at least four edges. Note that vertices not contained in any of the edges in E' are also removed from N.

Figure 5.14 shows the *filtered neighborhood graph* that corresponds to the *neighborhood graph* shown in figure 5.13.

Properties of Filtered Neighborhood Graphs Removing of edges from the *neighborhood graph*, as described above, results in a graph where almost all edges cover a field line in the image. It is very unlikely that objects not being field lines will cause the creation of aligned *field line segments*. Figure 5.15 gives an example of this very rare case. However, most of these cases are filtered by additional processing, cf. 5.1.8.

5.1.8 Line Clusters

While the *filtered neighborhood graph* is already a good representation of field lines in the image, it does not represent the individual field lines. Different field lines can be represented by a single connected region in the graph. The example in figure 5.14 shows three connected subgraphs. One of them represents the side line, a ground line part, and two lines of the penalty area. The second subgraph represents another ground line part. The third one represents a section of the center circle.

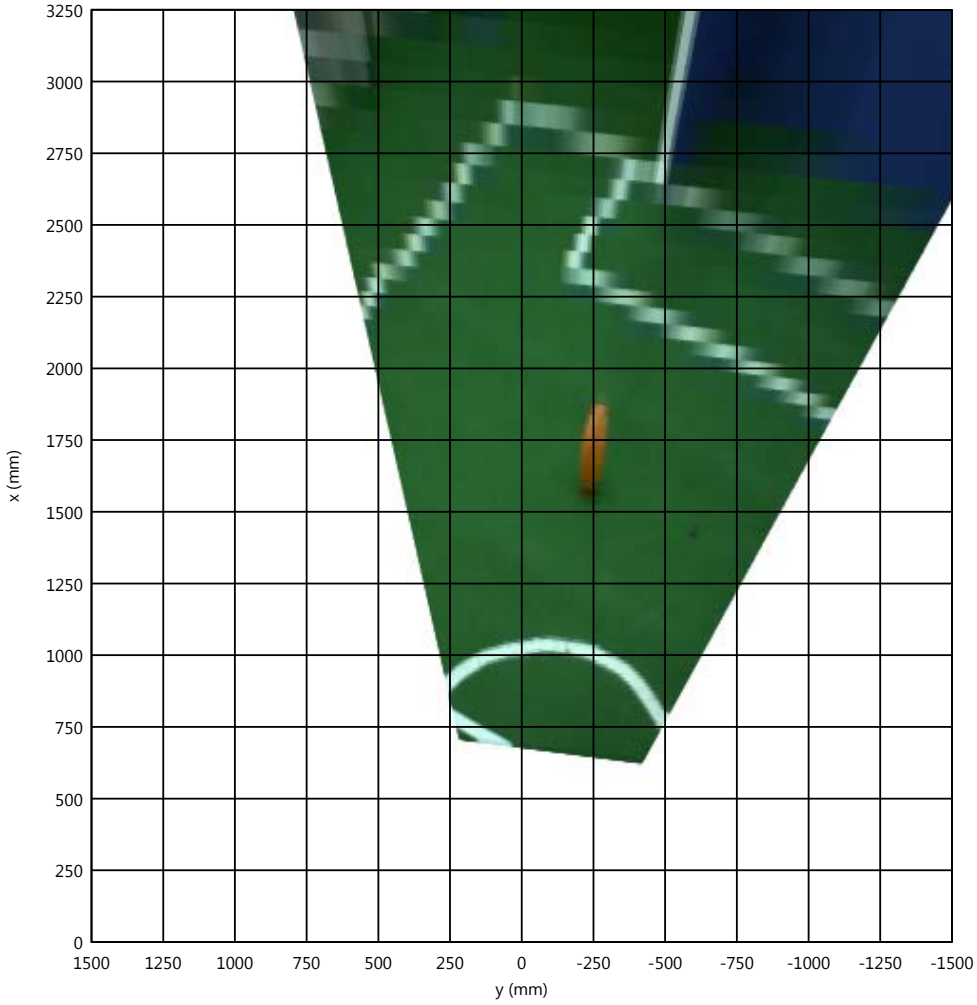


Figure 5.16: *Projection of the image from 5.14 to the ground.* In this projection, the angle between field lines is nearly 90°. The robot camera is vertically above the origin of the coordinate system. While in the original image field lines appear step-like, because of the low camera resolution, this effect is amplified in the projection. The amount of amplification is higher for distant field lines.

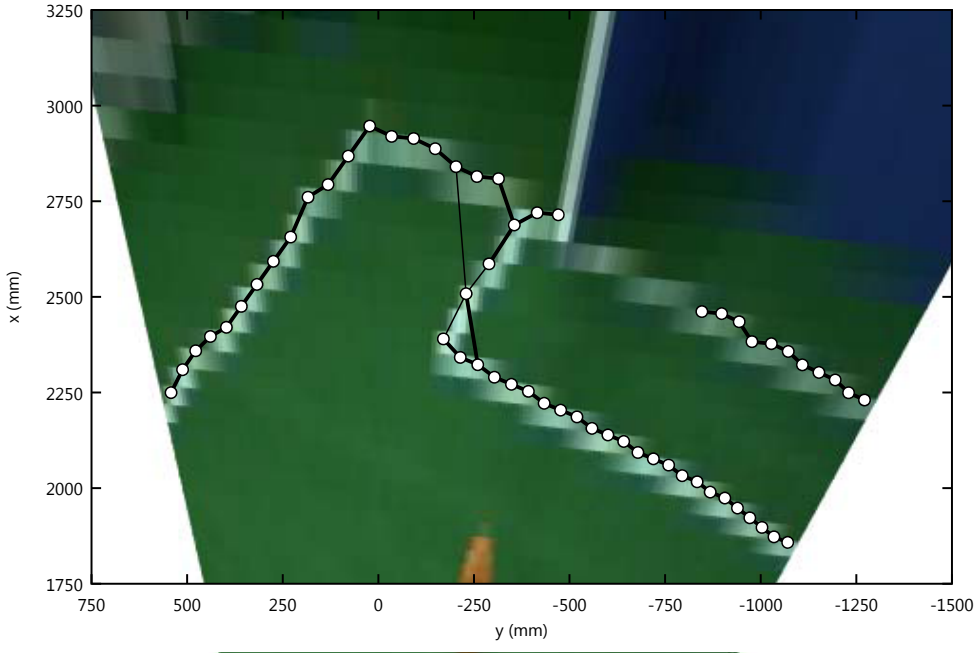


Figure 5.17: *Projection of the* filtered neighborhood graph *from figure 5.14 to the ground.* Note that only the graph parts close to the goal are shown; the center circle is outside the depicted area. The image in the background is there for illustration purpose only, the graph was not calculated based on the projected image but as described above. Just the single nodes of the graph were projected.

The next field line detection step is to split the *filtered neighborhood graph* into *line clusters* so each cluster contains only *field line segments* that belong to the same field line. For this, the fact that the angle between intersecting lines on the field is 90° can be used. However, the angle between field lines in the image differs depending on the perspective. Significantly, this angle can be quite low (for example, in the field corner in figure 5.14), making it difficult to detect the corner. So, the *filtered neighborhood graph* projected to the ground is used.

Figure 5.16 shows the image from figure 5.14, projected to the ground. In this projection, the angle between field lines is almost 90°. Note that only the image parts below the horizon can be projected to the ground. Such a projection is based on knowledge about the camera's position and rotation relative to the ground. The camera's opening angles are also needed.

However, not every pixel of the image needs to be projected to the ground. Just the center coordinate $\vec{c}(v)$ is projected for each vertex v of the *filtered neighborhood graph* N'. Figure 5.17 gives an example.

With such a projection of the vertices, a greedy algorithm can be defined that extracts subsets of edges from the *filtered neighborhood graph* N. This algorithm starts with the left-most node in the graph and adds edges to the current cluster by selecting the closest neighbor to the last added edge. A new cluster is started whenever the direction of the new edge has an angle of more

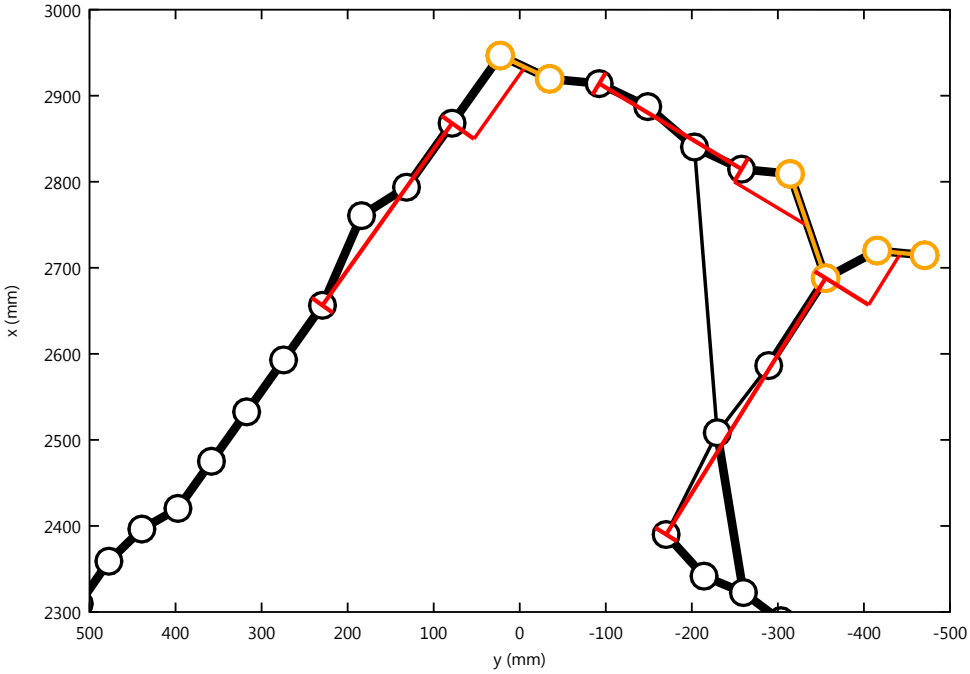


Figure 5.18: *Cluster creation.* To create edge clusters, angles in the projected graph are compared. Red lines: angle of the last three edges that were picked by the algorithm. Orange lines: angle of current edge. Angle between red and orange lines: When this angle is above the threshold of 45°, a new cluster is created. The figure shows all angle comparisons that led to the creation of a new cluster.

than 45° to the angle defined by the last three edges that were added to the cluster. When a new cluster is created, the old cluster is added to the list of found clusters and its edges are removed from the current graph, nodes without adjacent edges get deleted. Figure 5.18 illustrates the angle comparison for all cases where a new cluster was created. Figure 5.19 illustrates the *line clusters* generated by the cluster creation algorithm.

Properties of Line Clusters The main property of a *line cluster*, created as described above, is that all contained nodes belong to the same field line. Only in rare cases a line cluster can contain nodes from more than one field line. However, it can happen that objects like white robot legs cause a line cluster to be created. Figures 5.20a) and b) show examples of such false positives.

5.1.9 Filtered Line Clusters

While the *line clusters* are already a good representation for field lines, a simple filter can additionally reduce the likelihood of false positives. The filter operates on single line clusters. For each line cluster, the *line segments* that belong to the nodes in the cluster are taken into

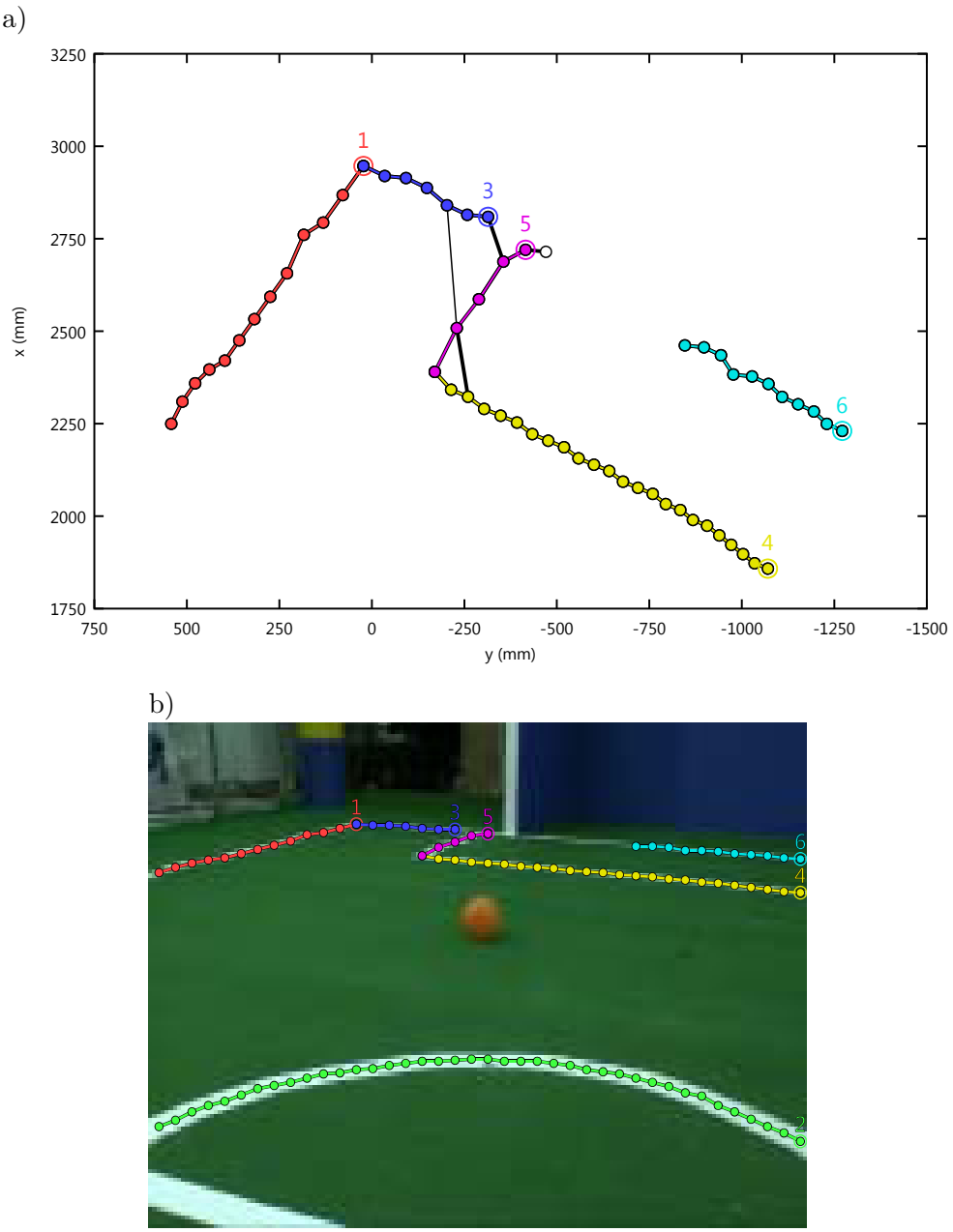


Figure 5.19: *Line clusters.* a) *Line clusters* as found by the greedy cluster creation algorithm. Each cluster is shown in a different color. The nodes where two clusters intersect belong to both clusters. This is illustrated by the small colored circle around the node. Note that the cluster with index 2 (caused by the center circle) is outside the depicted area. b) *Line clusters* shown in the image, each cluster is shown in a different color.

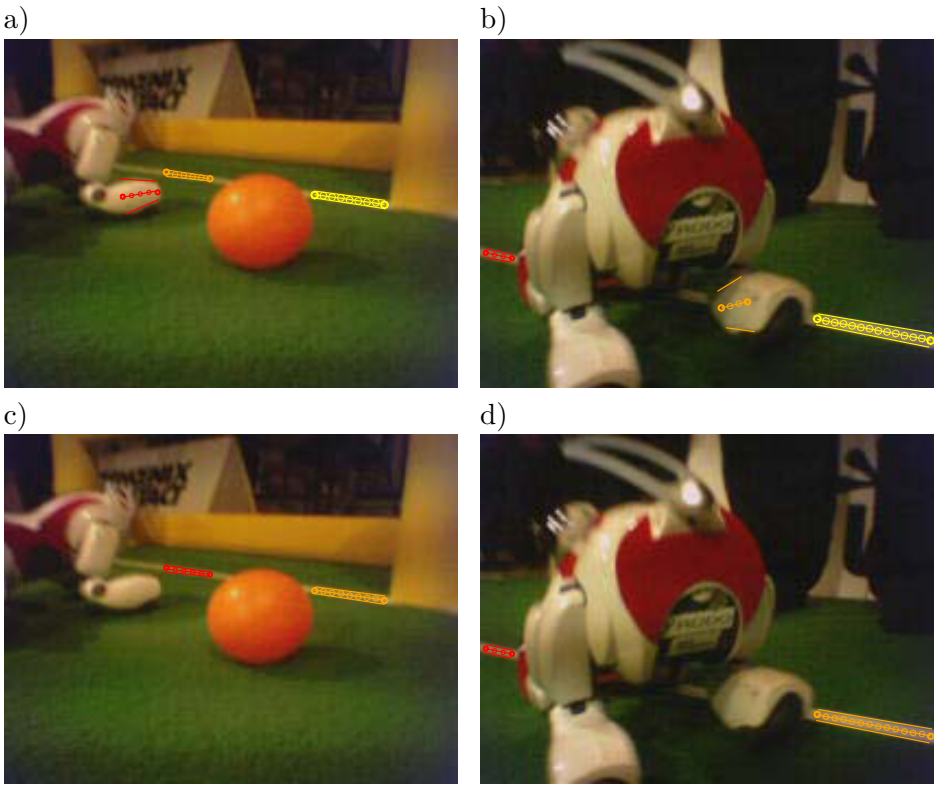


Figure 5.20: *Line clusters and filtered line clusters.* a,b) *Line clusters* as found by the greedy cluster creation algorithm. Each cluster is shown in a different color. c,d) *Filtered line clusters.* False positives were eliminated. The line clusters which were caused by the robot's legs were removed because their boundaries are not parallel (cf. condition 2).

account. Based on these, the upper and the lower boundary line for the line cluster are created by applying linear regression to all upper and all lower line segment points. The following conditions must be fulfilled for a line cluster to be considered for further processing:

Condition 1 *Boundaries don't deviate from straight lines.* Line segments are stripes of equal width. This is why single points which form the boundary line should not deviate more than a certain threshold from the line calculated by linear regression.

Condition 2 *Almost parallel boundaries.* The upper and the lower boundary have to be almost parallel. This, again, reflects the fact that field lines are stripes of equal width. Perspective distortion plays only a minor role because the field line width is small, compared to the camera height.

Condition 3 *Line cluster is not bent.* Curved lines (like the center circle, or other objects) should be filtered out. For this the angle created by a line cluster's center point and its end points should almost be 180°.

Properties of Filtered Line Clusters Filtered line clusters contain almost no false positives. A single field line might be represented by more than one line cluster. Figures 5.20c) and d) give examples of successfully removed line clusters which were caused by robot legs.

5.1.10 Field Lines and Corners

The result of all image processing are *field lines* and *field line corners*. Field lines are calculated based on the *filtered line clusters*. Line clusters located on straight lines close to each other are grouped together. Based on all line segments which belong to such a field line group, a straight line which represents the field line, in the image, is calculated using linear regression.

Field line corners are calculated based on those straight lines. A field line corner is characterized by its center (the point were the field lines meet) and its direction. There are L-like corners T-like corners and X-like corners. For an L-like corner, the direction is the bisection of the angle defined by the two lines. A T-like corner is the combination of two L-like corners, an X-Like corner is the combination of four L-like corners. The corner direction is important for the visual compass introduced in section 5.2. Note that there are also *virtual corners* at the intersections of straight lines representing field lines which don't meet each other (e.g. the side lines and the penalty line). This is intended, as those virtual corners are valuable localization landmarks.

Figures 5.21 and 5.22 show field lines and corners detected by the methods described above.

5.2 Memory-Based Direction Calculation

This section describes how field lines and corners detected by the vision system described in the previous section can be used to estimate *Compass Data* for a robot. Compass data, basically, gives information about the robot's rotation. Different kinds of compass data can be distinguished:

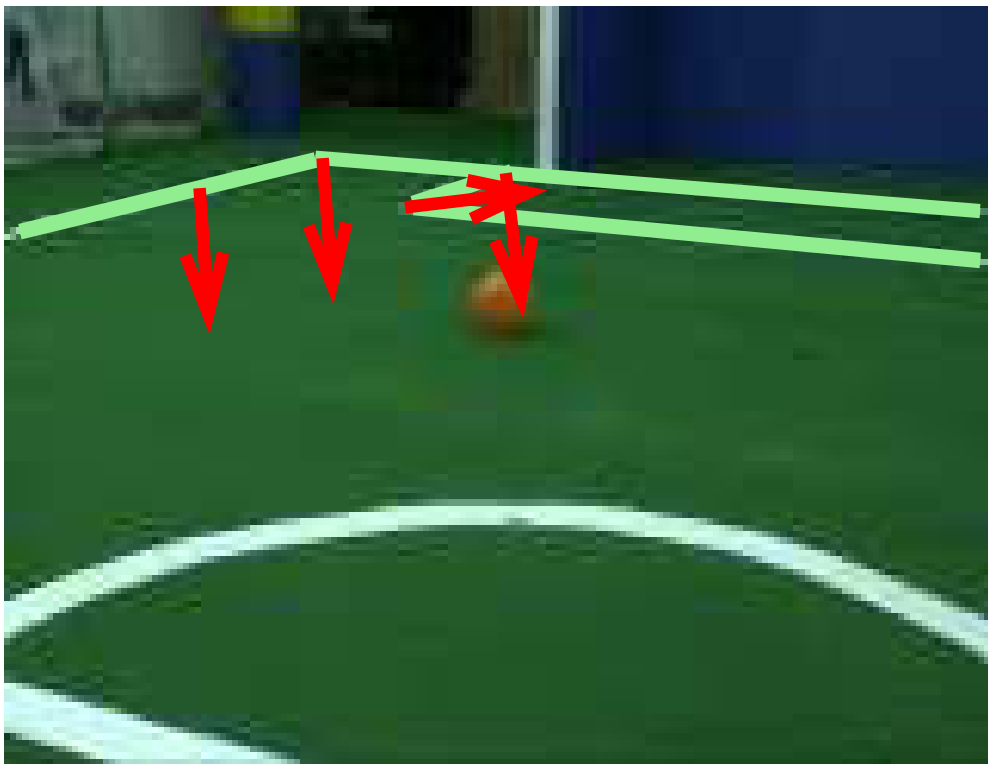


Figure 5.21: *Field lines and corners.* Green lines: detected field lines. Note that the center circle was successfully excluded. Red arrows: the starting points mark the corner locations. The heading marks the corners directions.

a) b)

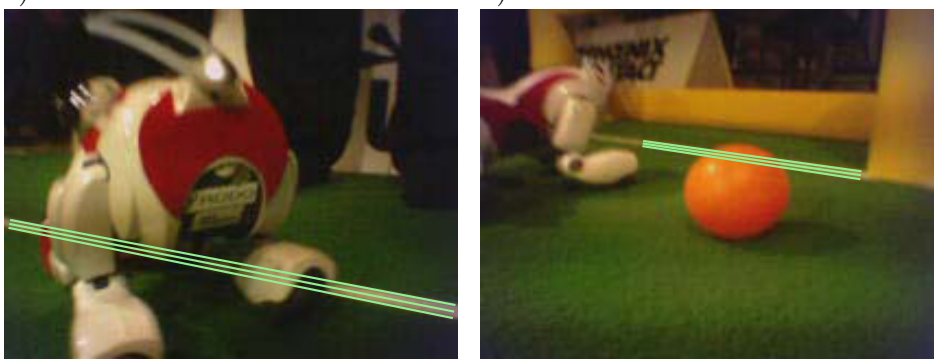


Figure 5.22: *Field lines as detected by the image processing method described in this section.*

Full compass data provides the robot rotation with respect to a reference direction (the angle provided is between $-180°$ and $+180°$).

Semi compass data is the kind of information provided by a compass where the needle's north and south are indistinguishable. The angle provided is between $-90°$ and $+90°$ and it is unknown whether there is a $180°$ offset to the reference direction.

Quarter compass data is the kind of information provided by a compass, instead of a needle, containing a cross pointing in four directions, with an angle of $90°$ between the directions. In this case, the angle provided is between $-45°$ and $+45°$ and it is unknown whether the offset to the reference direction is $0°$, $90°$, $180°$, or $270°$.

In this section, it is shown how *quarter compass data* can be determined. This is done following the *memory-based paradigm*. Section 5.2.1 shows how the paradigm is applied. In section 5.2.2, results of experiments conducted with a humanoid robot are discussed.

5.2.1 Applying the Memory-Based Paradigm

Following the *memory-based paradigm* (cf. 3.3), the robot state x can be estimated, based on past observations $\vec{z}$ and control data $\vec{u}$, using this function:

$$f_M(\vec{z}, \vec{u}) := \arg\min_k \sum_{t=1}^{n} (f_s(f_c^*(x_k, v_t)) - z_{n-t})^2.$$

For this, an observation function f_s which gives the expected observation z for each state is needed, cf. definition 3. A second prerequisite is a control function f_c which gives a new state x_i based on the previous state x_{x-1} and the performed control action u_i, cf. definition 4. Note that v_t denotes the accumulation of the last t control actions.

Defining the State Space, the Sensor and the Control Data The state estimation goal, described in this section, is to determine *quarter compass data*. The state space is one-dimensional.

The vector $\vec{z}$ of all observations is a sequence of angles derived from field line percepts. While a field line percept is a straight line in an image, using the information about the camera's position and heading relative to the robot, this can be transformed to a straight line on the ground relative to the robot. This straight line can be described using Hesse normal form, cf. [11]: $\vec{r} \cdot \vec{n} = c$. All field line percepts in $\vec{z}$ are represented by the angle of their normal vector $\vec{n}$; the distance to the origin is ignored. Note that all angles are normalized between $-45°$ and $+45°$.

The control action vector $\vec{u}$ is derived from the odometry data sequence. The control action u_i is defined as the relative odometry data at time t_i.

$$u_i := o_c(t_i, t_{i+1}).$$

Observation and Control Function With these definitions for the state, the kind of measurements, the control data characteristics, the observation and the control function specific for this application of *memory-based state estimation* can be defined. The observation function f_s calculates, for a robot rotation (α) (normalized between -45 and $+45$), the expected angle to any

detected field line. Due to the normalization and the use of the Hesse normal form, the function is trivial:

$$f_s(\alpha) := \alpha.$$

The control function f_c calculates, for a rotation α_{i-1} and a control action given by a robot's pose (x_o, y_o, α_o), a new position:

$$f_c(\alpha_{i-1}, \alpha_o) := \alpha_{i-1} + \alpha_o.$$

With the definition of these functions, *memory-based state estimation* can be applied.

Percept Selection Strategies While deriving the robot's angle relative to a field line from a field line percept is quite accurate when the robot is not moving, it can become quite inaccurate when walking. Especially humanoid robots might have an inaccurate representation of their camera's relative position and orientation while walking. This can lead to errors in the projection to the ground. However, because of the cyclic and symmetric nature of walking motions, the effect of the errors can be eliminated by incorporating more than one measurement. A low number of percepts were enough (for humanoid robots and Aibo robots) to get reliable results. This is discussed further in the next section. For all experiments, the number of field line percepts used for rotation estimation was set to five.

5.2.2 Experiments

This section describes the experiments which were conducted to test the rotation estimation method introduced in the previous section. The tests were done with a humanoid robot.

Setup The experiments were done using a humanoid robot constructed at TU Darmstadt [30]. The robot has two built-in cameras; one in the head and one in the chest. The head camera can be horizontally panned and vertically tilted to control the viewing direction. The horizontal and vertical angles of view are 44° and 34°. The chest camera has a wider angle of view; its viewing direction is fixed. All experiments were done on a *RoboCup Humanoid League* [17] soccer field which is 4.5 meters long and 3 meters wide. The field boundaries are marked by white lines of width 5 cm. Additionally, there are a center line and lines which form rectangular penalty areas, next to the goals.

Objectives The goal of the experiments is to test that the overall system, consisting of the field line detection as described in 5.1 and the rotation estimation described in this section, works as expected. To do so, estimation results are compared to ground truth. An important aspect of the experiments is to show that faulty odometry data, especially systematic error, do not affect the accuracy of the rotation estimation. Additionally, the estimation results are compared to a reference estimation done by a particle filter.

Recorded Data For the experiments, repeatedly, the robot was approaching a ball from varying starting positions. This was realized by activating the standard soccer behavior which walks

in a curve behind the ball and then tries to kick it to the opponent goal. The standard vision and localization methods of the Darmstadt Dribblers [30] were active to achieve this. While the robot walked to the ball, the ball was moved around on the field to create motion patterns typical for soccer games. Each of the single robot runs had an approximate length of 5 meters.

This data was recorded in the experiments to off-line test memory-based rotation estimation:

- The *rotation matrix* which describes the spatial position and orientation of the head camera relative to the ground.
- *Images* taken by the head camera.
- *Odometry data* derived from the executed motions of the robot.

This data was recorded to validate the estimation results:

- *Ground truth data* obtained using a ceiling camera based tracking system [13] which detects a colored marker attached to the robot. Ground truth data contains the real robot's position and orientation.
- *Reference position and rotation* created by the particle filter based localization system running on the robot.

All data was recorded at a rate of approximately 8 frames per second. Note that the particle filter which calculated the reference rotation uses goals, beacons, and field lines detected in images of both cameras (by the Darmstadt Dribblers vision system); while the memory-based rotation estimation only uses field lines detected in the image of the head camera (by the system introduced in 5.1).

Data Analysis All experimental runs consist of approximately 110 frames of data. Based on this, the rotation was estimated for each frame. Note that no rotation estimate is created until five field lines or corners are detected. Usually, this happens within the first 10 frames. All estimated rotations are compared to ground truth. Additionally, the particle filter reference rotation is compared to ground truth. For each run, the mean and standard deviation of the angle error are calculated based on the, approximately 100, frames containing a rotation estimate.

Results The analysis of the reference rotation shows that it can be affected by systematic error: While most runs had a mean angle error of 2°, some runs had a mean angle error of up to 10°. In none of the runs, the maximum mean angle error of the memory-based rotation estimation was above 3° which is close to the expected 0° mean angle error. For both the memory-based and the reference estimation, the standard deviation of the angle error was approximately 15° for all experimental runs.

Figure 5.23 shows the estimation results for an experiment where the robot walked in a curve towards the ball. The results are compared to ground truth data and odometry data. To illustrate the deviation, the relative odometry data was initialized with the first value of ground truth data.

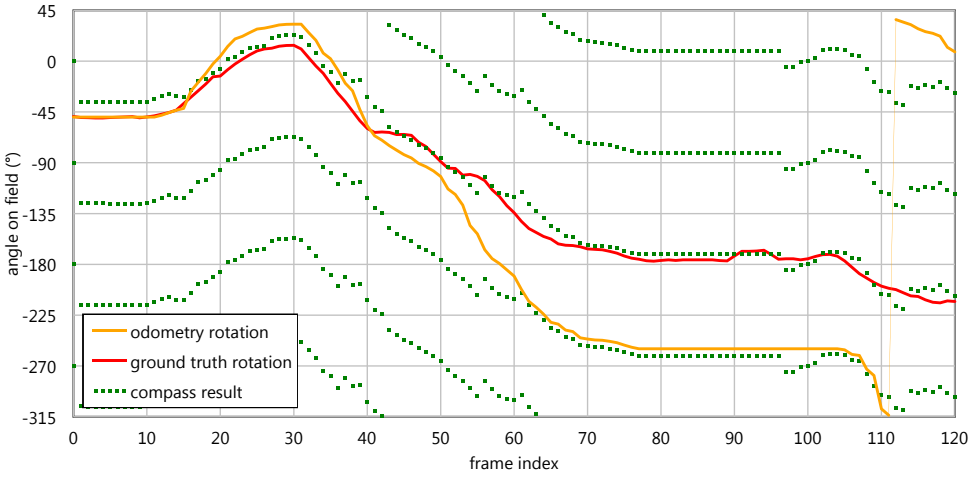


Figure 5.23: *Experimental results.* The rotation estimation's result is *quarter compass data*. To compare it to the ground truth rotation, all four possible directions are illustrated. The estimation result and the ground truth match very good. The faulty odometry has no negative influence on the estimation's accuracy.

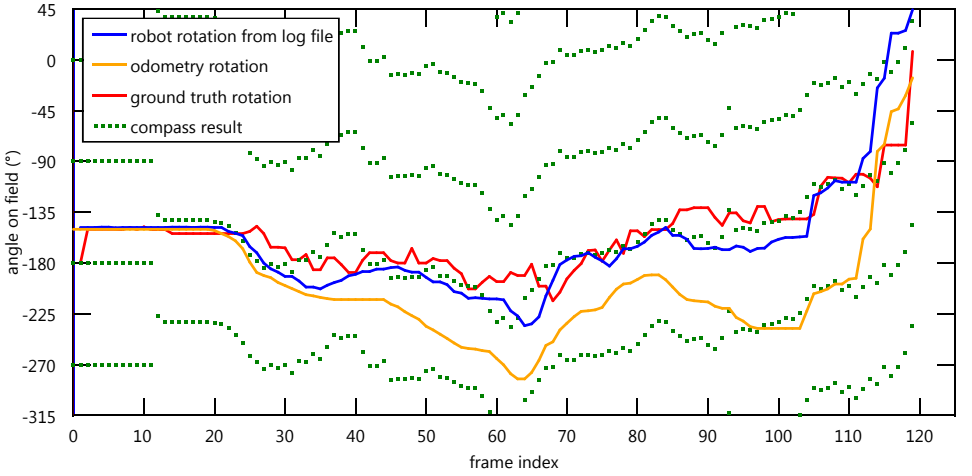


Figure 5.24: *Experimental results.* The rotation estimation's result is *quarter compass data*. To compare it to the ground truth rotation, all four possible directions were illustrated. The estimation's result and the ground truth match very well. The faulty odometry starting at frame 110 has affected the particle filter (blue curve). Memory-based rotation estimation (green curve) was not affected.

Figure 5.24 shows a different run, where the robot walked to the ball and then performed a sharp turn left because the ball moved away. It can be seen that odometry data for this turn is affected by a large systematic error. This error led to a poor estimation using the particle filter. However, the memory-based rotation estimation performed very well in that case. In section 3.1.3, this advantage of *memory-based state estimation* was discussed.

Note that the tests done on an Aibo robot revealed that, because of the low height of the camera above the ground, field lines are not seen often enough to enable a continuous rotation estimation. The only exception is the goal keeper which constantly sees the penalty lines. However, unfortunately, the ground truth tracking system didn't cover the goal areas of the field. That's why no qualitative data is given here.

5.3 Discussion

In this chapter, a vision-based compass for soccer robots is presented. One part of it is a vision system which recognizes field lines and corners in images. The output of the vision system are the position of field lines and corners relative to the robot. The second component is a memory-based rotation estimation which uses the vision system's output and odometry data. The result is the robot's heading on the field.

Vision System The vision system works based on scan-lines and thus has low computational costs. It does not need any lighting specific calibration. A variant of it was used in the 2007 and 2008 RoboCup four-legged league competition. The layered filtering ensures that false positives are very unlikely. A design principle of the system is that a not detected field is better than a wrongly detected one. So, only a small number of perceptions has to be selected by the memory-based rotation estimation; while the estimation accuracy is already high.

State Estimation The three advantages of *memory-based state estimation*, as outlined in chapter 3, are its ability to cope with sparse information, with kidnappings, and with systematic error in the state transition model.

Information can be sparse for two reasons: high noise or low contribution compared to the state space. Because of the strict false positive filtering, field line perceptions are not very noisy. They also have a high contribution: a single percept already provides rotation information. So, the ability of *memory-based state estimation* to process sparse information is not needed in this case.

However, the ability to cope with kidnappings is shown by the fact that, in all experimental runs, the rotation estimation is reliable after the first couple of frames containing field line percepts. Furthermore, the experiments show that the memory-based rotation estimation can cope with systematic error in odometry data. The mean angle error (determined using ground truth data) is always close to zero, in contrast to the particle filter approach which accumulated this error. The visual compass described in this chapter is used for the localization method given in the next chapter.

6 Compass and Bearing Localization

In this chapter, a localization method is described which uses compass data, as introduced in chapter 5, and horizontal bearings. It is the *memory-based state estimation* equivalent to the nautical example from section 3.1.1.

Ground localization is typically a three dimensional problem (x-position, y-position, and rotation have to be determined). This is why all calculations dealing with the state to be estimated are three dimensional. The localization method described in this chapter splits this in two subproblems: rotation estimation (one dimensional), and position estimation (two-dimensional). This leads to less complex calculations which can be performed very quickly. The rotation estimation component has already been described in the previous chapter.

This chapter is structured as follows. A way to correct faulty odometry using compass data is given in section 6.1. This is an important prerequisite for the position estimation described in section 6.2. Experiments were done with a humanoid robot on a RoboCup soccer field, using the same setup as described in 5.2.2. For horizontal bearings, the goal posts were used. A discussion follows in section 6.3.

6.1 Odometry Correction

This section describes how to use compass data to correct faulty odometry data. Section 6.1.1 describes why odometry can be faulty and how this can influence localization. Section 6.1.2 describes how the rotation component of odometry data can be corrected while a robot moves. In section 6.1.3, it is shown how the correction performs on real-world data.

6.1.1 Cause and Effect of Faulty Odometry Data

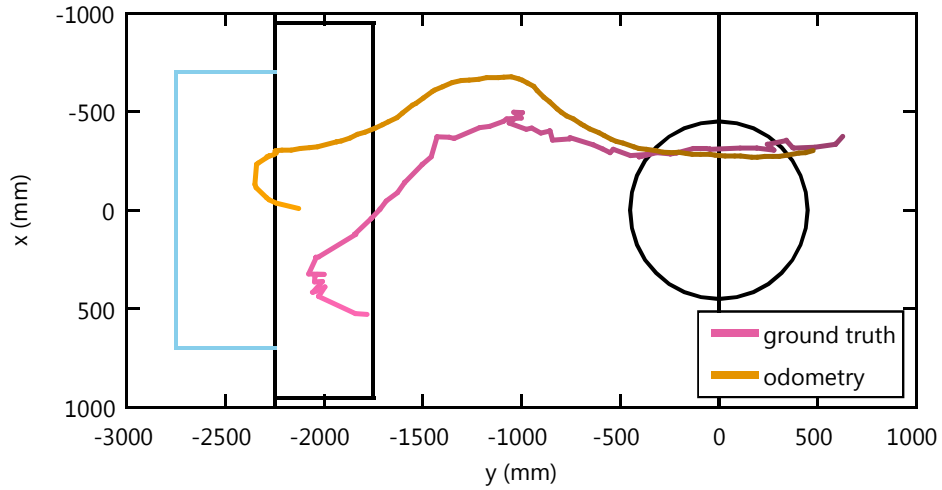
Good odometry data is an important prerequisite for mobile robot localization. For wheeled robots, odometry data can be achieved by measuring the number of revolutions for each wheel. With calibration explicit for factors like ground slipperiness good results can be achieved.

Issues are more complex for legged robots. The deviation between intended movement and movement actually executed is much higher for such robots. This deviation can depend on many factors. Some of them are static; while others may change over time. Examples are ground slipperiness, the degree of joint wear out, battery level, and joint temperature.

It is hard to correct such factors with pre-run calibrations. Figure 6.1 gives examples of how this deviation typically looks, even though the robot was calibrated.

The better odometry data matches ground truth data, the better it works for self-localization. Errors in the rotation component of odometry data have had the biggest impact on the deviation between ground truth and odometry data. Even slight rotation errors can quickly lead to large distance deviations.

a)



b)

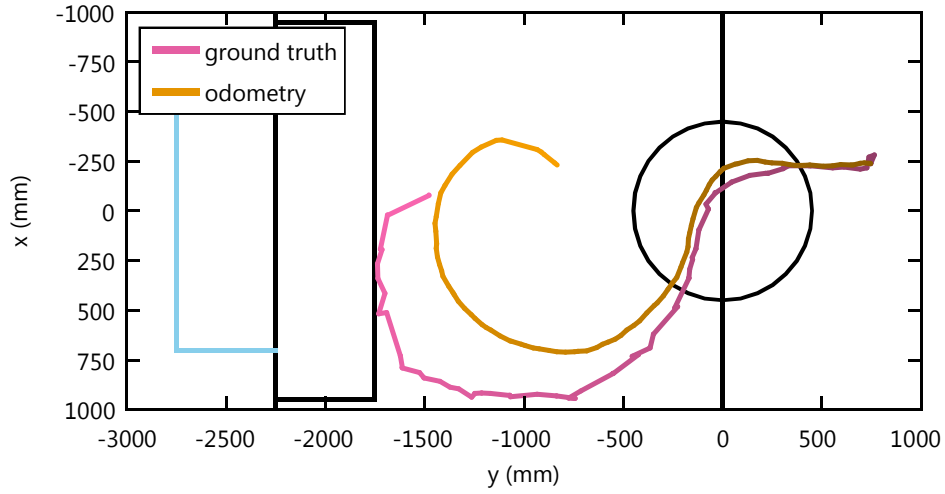


Figure 6.1: *Faulty odometry data.* Two typical odometry data recordings. The pink curves show the paths a humanoid robot walked over a soccer field while following a ball. This ground truth data was obtained using a camera mounted from the ceiling, watching a colored marker attached to the robot. The orange curves depict the recorded odometry data. While odometry data is relative, for this depiction, it was initialized with the first ground truth pose to visualize the deviation.

Unfortunately, rotation errors are hard to compensate for with pre-run calibration. They depend on factors which can constantly change. Joint wear out and temperature changes are not distributed evenly within the robot. This can lead to asymmetric behavior, resulting in rotation which deviates from the anticipated one. The nature of the ground can also lead to unforeseeable deviations in the rotation. Especially carpets have the property that the amount of slipperiness depends on the robot's walking direction. The result is a robot executing motions with constant rotation, following an ellipse instead of a circle.

6.1.2 Recursive Odometry Correction

This section describes how to use rotation estimates, as described in the previous chapter, to correct odometry.

The Relative Nature of Odometry Data Odometry data provides the predicted accumulated effect of robotic motion. For a given time t_i the odometry data

$$o(t_i) := (x_i, y_i, \phi_i)$$

is a pose that describes how the robot moved, relative to the pose it had at time t_0. The difference

$$o(t_i, t_j) := o(t_j) - o(t_i)$$

describes the relative movement between times t_i and t_j. This relative movement is also a pose:

$$o(t_i, t_j) = (x_{(i,j)}, y_{(i,j)}, \phi_{(i,j)}).$$

This means there are two ways to store an odometry data sequence. One possibility is the *absolute way*, with a sequence of poses relative to the pose at time t_0, given as:

$$o(t_1), ..., o(t_n).$$

The *relative way* is to give a sequence of poses, relative to each other:

$$o(t_0, t_1), o(t_1, t_2), ..., o(t_{n-1}, t_n)$$

How to obtain relative from absolute data was shown above. Obtaining absolute from relative data is also straightforward. The odometry data for a given time is the sum of all previous relative movements:

$$o(t_n) = \sum_{i=0}^{n-1} o(t_i, t_{i+1}).$$

This can also be described in a recursive fashion:

$$o(t_n) = o(t_{n-1}) + o(t_{n-1}, t_n).$$

Odometry Correction The odometry data correction goal at time t_n is to provide a sequence of corrected odometry data, in the *absolute way*:

$$o_c(t_1), ..., o_c(t_n).$$

The correction is done recursively. To calculate the corrected odometry $o_c(t_i)$ at time t_i, the current rotation estimation $\alpha(t_i)$, the current relative (not corrected) odometry $o(t_{n-1}, t_n)$, and the previous corrected odometry $o_c(t_{i-1})$ at time t_{i-1} are needed.

The corrected odometry $o_c(t_0)$ at time t_0 is defined as:

$$o_c(t_0) := (0, 0, \alpha(t_0)).$$

This is a pose at the origin of the coordinate system, heading in the direction given by the estimated rotation at this time t_0.

For the recursive calculation, some prerequisites are calculated. The angle

$$\delta(t_i) := \alpha(t_i) - \phi(o_c(t_{i-1}))$$

is the difference between the estimated angle and the angle of the last absolute corrected odometry pose. The length of the current relative odometry is given as:

$$l := |o(t_{i-1}, t_i)|.$$

This is used to define the vector

$$\vec{d} := (l\cos(\delta(t_n)), l\sin(\delta(t_n))).$$

With this, the corrected odometry is defined as:

$$o_c(t_i) := (|o_c(t_{i-1}) + \vec{d}|, \alpha(t_i)).$$

This correction results in the robot's rotation, given by the odometry, always matching the estimated rotation. The length of the relative odometry is not changed, only the heading of the pose is corrected to match the estimated rotation.

Figure 6.2 shows a sequence of odometry data and how it gets corrected using the mechanism introduced in this section.

6.1.3 Real-World Odometry Correction Examples

In this section, how good the correction method described above works on real-world data is discussed. Figure 6.1 depicts faulty odometry data. This data was obtained by a humanoid robot following a moving ball on a robot soccer field. The real robot position was tracked by an external camera using the method described in [13]. This allows accuracy evaluation of the odometry data and its correction.

Note that the ground truth data is a sequence of absolute poses, while the odometry data is a sequence of relative poses. To show the accuracy of odometry data, it is transformed to a

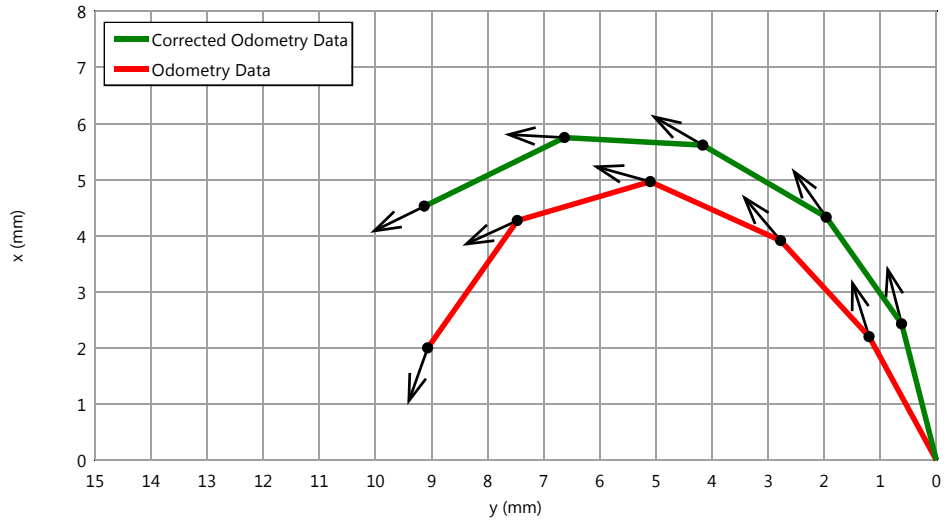


Figure 6.2: *Odometry data correction.* The red lines show a sequence of poses representing odometry data. This sequence is corrected using a rotation information sequence which corresponds to the arrows at the green line. The corrected odometry data is depicted in green.

sequence of absolute poses by adding the first ground truth pose to all odometry poses.

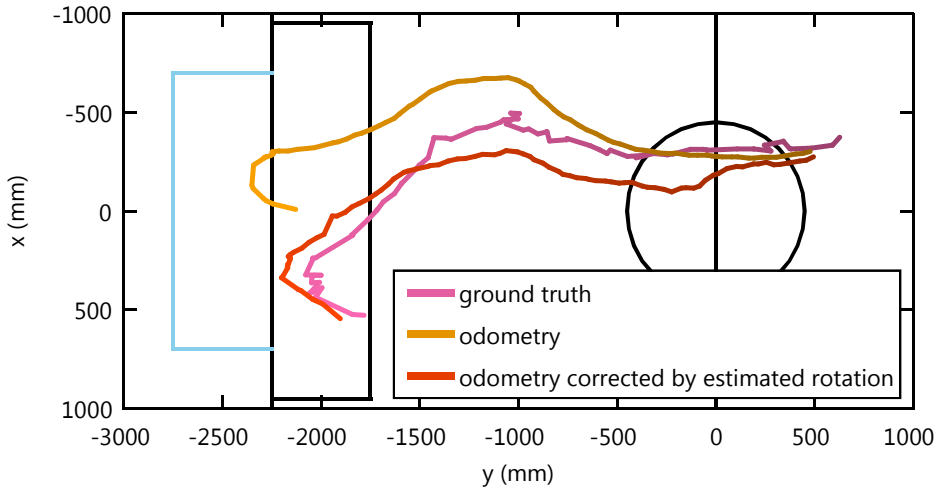
It can be seen very easily that the odometry data reflects the distance the robot walked quite well. This was achieved through pre-run calibration. However, the odometry data's rotational component is affected by an obvious error. The two robot runs chosen for illustration have a right prone deviation. The robot measured a stronger right turn and a weaker left turn than actually performed.

Figure 6.3 shows the correction results for the two runs of the humanoid robot. As expected, the correction is so good that when initialized with the first ground truth data, the odometry data and the ground truth data are very close. The corrected odometry data is a perfect prerequisite for localization.

6.2 Localization

This section shows how *memory-based state estimation* can be applied to the localization problem where compass data and horizontal bearings are given. In 6.2.1 the localization method is given; in 6.2.2 the outcome of experiments is described.

a)



b)

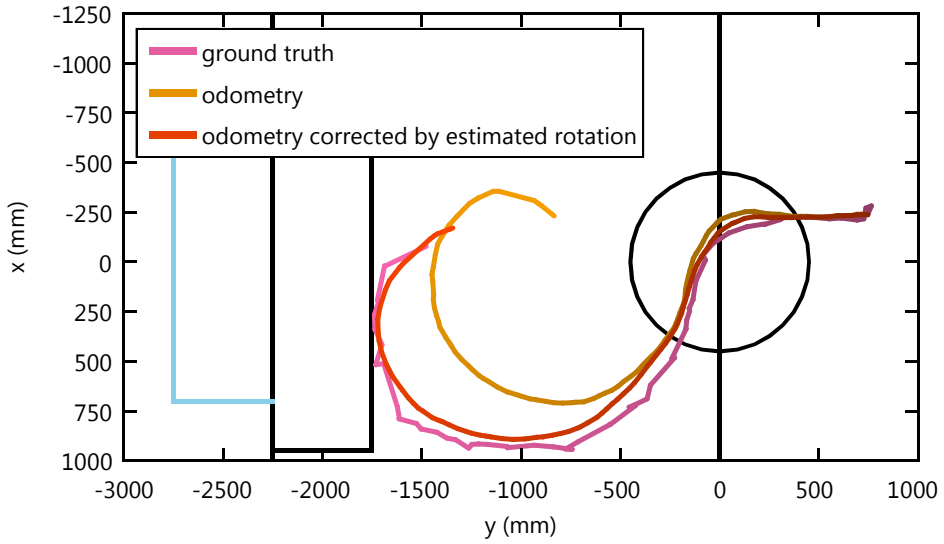


Figure 6.3: *Corrected odometry data.* Two typical recordings of odometry data. The pink curves show the paths a humanoid robot walked over a soccer field while following a ball. This ground truth data was obtained using a camera mounted on the ceiling, watching a colored marker attached to the robot. The orange curves depict the recorded odometry data. The red curves show the odometry data corrected by the estimated position. The faulty and the corrected odometry data were both initialized with the first ground truth pose, to depict the deviation.

6.2.1 Applying the Memory-Based Paradigm

Following the *memory-based paradigm* (cf. 3.3), the robot state x can be estimated, based on past observations $\vec{z}$ and control data $\vec{u}$, using this function:

$$f_M(\vec{z}, \vec{u}) := \arg\min_k \sum_{t=1}^{n} (f_s(f_c^*(x_k, v_t)) - z_{n-t})^2.$$

For this, an observation function f_s that gives the expected observation z for each state is needed, cf. definition 3. A second prerequisite is a control function f_c that gives a new state x_i based on the previous state x_{x-1} and the performed control action u_i, cf. definition 4. Note that v_t denotes the accumulation of the last t control actions.

Defining the State Space, the Sensor and the Control Data The localization method described in this chapter is based on a compass data sequence, horizontal bearings to landmarks of known position, and (corrected) odometry data. As the rotation is given by the compass data, just the translational component of the robot's pose has to be estimated. The state space is two-dimensional.

To simplify all calculations, the compass data and the bearing measurements relative to the robot are combined, such that the vector $\vec{z}$ of all observations is a sequence of absolute landmark bearings. The vector of control actions $\vec{u}$ is derived from the sequence of corrected odometry data. The control action u_i is defined as the translational component of the relative odometry data at time t_i.

$$u_i := t(o_c(t_i, t_{i+1})).$$

These simplifications can be illustrated utilizing the navigation example given in section 3.1.1. All bearings are stored relative to north; odometry is a sequence of two-dimensional vectors in a north-aligned coordinate system. The advantage is that the information provided by the compass is not stored separately but integrated in vectors $\vec{z}$ and $\vec{u}$. For practical reasons, on a soccer field, north can be defined as the direction from one goal to the other.

Observation and Control Function With these definitions for the state, the kind of measurements, and the control data characteristics, the observation and the control function specific to this application of *memory-based state estimation* can be defined. The observation function f_s calculates, for a position (x, y) and a landmark l with known coordinates (x_l, y_l), the expected angle between the robot's heading and the landmark:

$$f_s(x, y) := \arctan(y_l - y, x_l - x).$$

The control function f_c calculates for a position (x_{i-1}, y_{i-1}) and a control action, given by a two-dimensional vector (x_o, y_o), a new position:

$$f_c((x_{i-1}, y_{i-1}), (x_o, y_o)) := (x_{i-1} + x_o, y_{i-1} + y_o).$$

With the definition of these functions, *memory-based state estimation* can be applied. Fig-

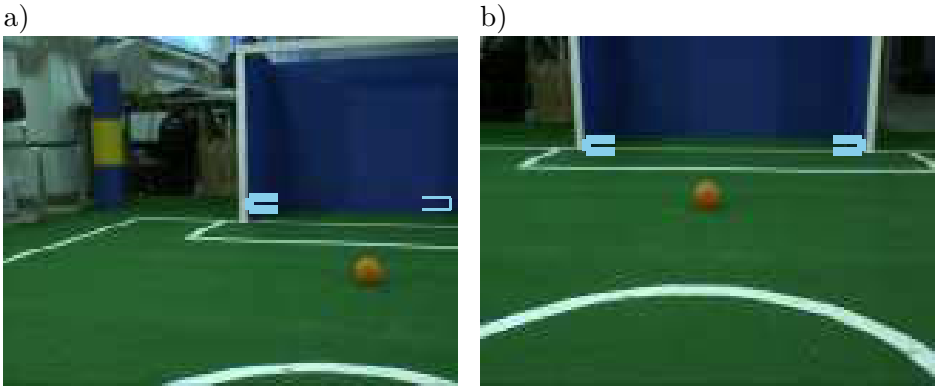


Figure 6.4: *Goal percepts.* Two images of a goal. a) Horizontal bearing to the left goal post was determined. b) Horizontal bearing to both goal posts was determined.

ure 6.4 shows goal posts recognized in images using a method similar to the one given in [45]. Figure 6.5 shows the resulting function $f_M(\vec{z}, \vec{u})$ for these goal observations. Each landmark observation contributes to f_M with a function having minima along a straight line given by the angle to the landmark and the motion since the observation. These straight lines conform to the *lines of position (LOP)* introduced in section 3.1.1.

Figure 6.6 shows the corresponding LOPs for the functions depicted in figure 6.5. Note that the calculation of these lines is not part of the localization method; they are just given for illustration.

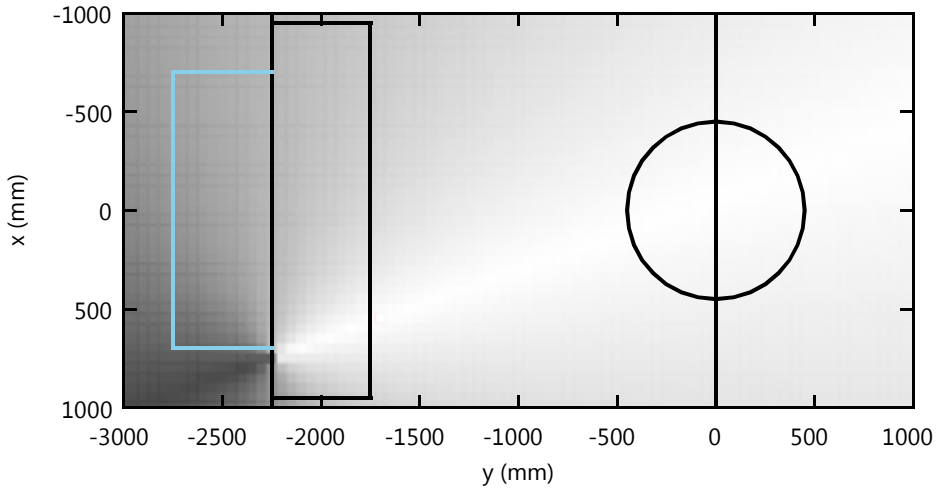
6.2.2 Experiments

This section describes the experiments which were conducted to test the localization method introduced in this chapter. The tests were done with a humanoid robot using the same data sets as described in section 5.2.2. The memory-based rotation estimation uses the estimated rotation (cf. 5.2), corrected odometry data (cf. 6.1), and horizontal bearings to goal posts (obtained as described in [45]). A very simple percept selection strategy was chosen: for each of the four goal posts (left, right; blue, yellow) the last five observations were selected to estimate the position.

Objectives The goal of the experiments is to test the localization accuracy. This is done by comparing the estimation results to ground truth. Additionally, the estimation results are compared to a reference estimation done by a particle filter. Important aspects of the experiments are to show that the method is able to process the sparse information provided by the bearings and that the initial position is estimated as soon as enough bearings are available.

Data Analysis All experimental runs consist of approximately 110 frames of data. Based on this, the pose was estimated for each frame. The minimum was searched for on a 10 *cm* spaced grid. Note that no pose estimate is created until at least two goal posts are seen and the rotation is estimated. Usually, this happens within the first 20 frames. All estimated poses are compared

a)



b)

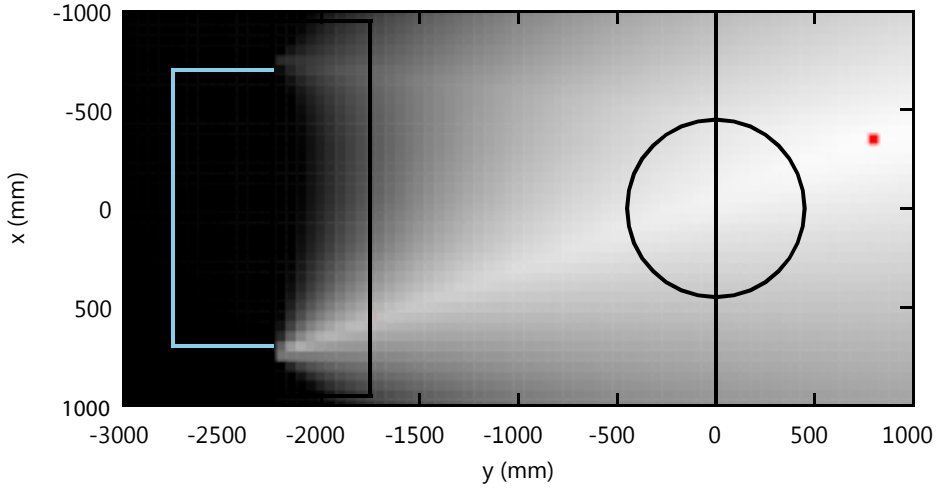
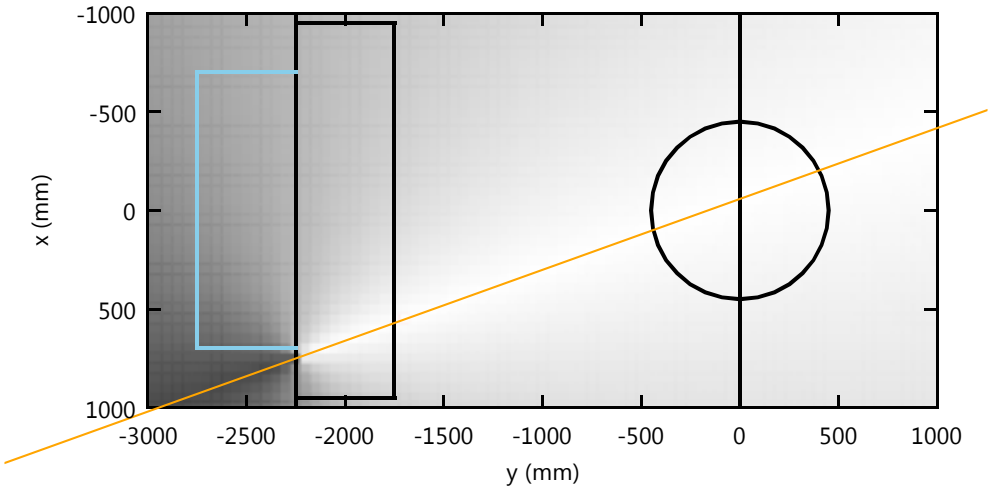


Figure 6.5: *State estimation function.* Black: $f_M = 1$; White: $f_M = 0$. a) Function $f_M(\vec{z}, \vec{u})$ after observing the left goal post (cf. figure 6.4a). b) Function $f_M(\vec{z}, \vec{u})$ after one observation of the left goal post and one observation of both goal posts (cf. figure 6.4b). The red dot marks the global minimum of f_M.

a)



b)

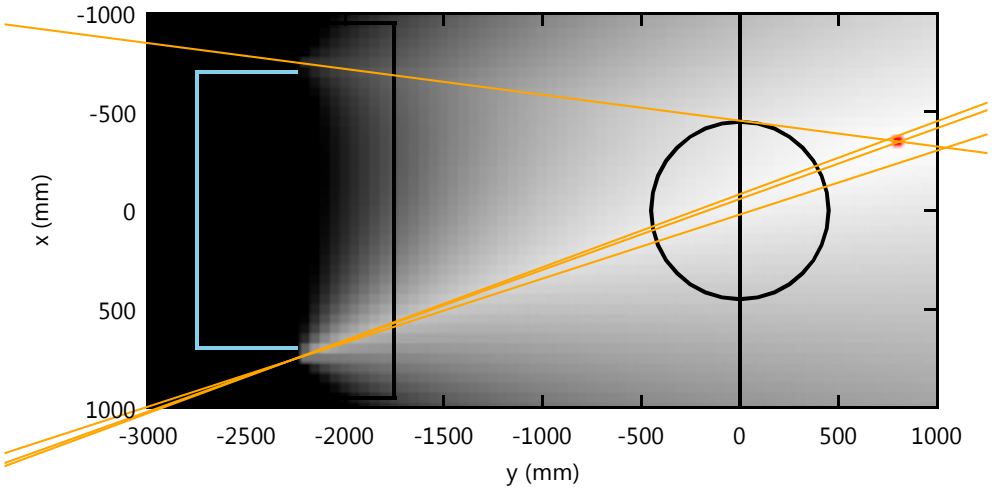


Figure 6.6: *Lines of position.* LOPs created based on the controls and sensor readings used to calculate f_M (background) a) after observing the left goal post. b) after two observations of the left goal post and one observation of both goal posts.

to ground truth. For this, a coordinate system is used which has its origin at the ground truth position and its x-axis parallel to the ground truth heading. For each estimate, the vector from the origin to the estimated position describes the localization error. In the same way, the particle filter poses are compared to ground truth.

Results The mean position error is, as expected, close to zero for the memory-based localization (40 mm) and for the particle filter (55 mm). However, the standard deviation for the memory-based localization is lower (164 mm) compared to the particle filter (245 mm). Compared to the field length, the memory-based method's standard deviation is 3.3%, the particle filer's 4.9%. These numbers were calculated based on a 192 position comparisons along a total distance of 8.7 m. The error distribution, for both methods, is shown in figure 6.7. Figure 6.8, for two experimental runs, depicts a comparison between ground truth data and the results of the memory-based localization and the particle filter.

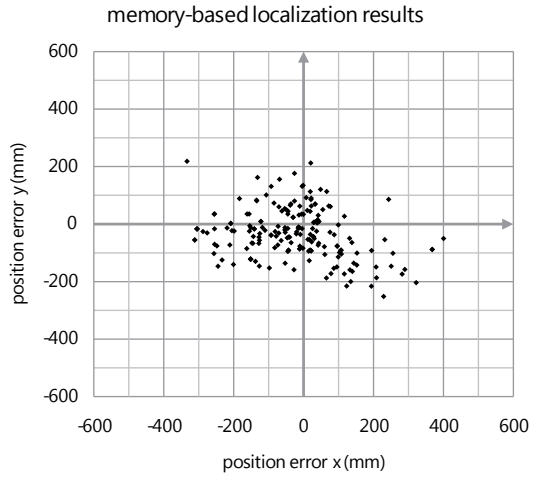
6.3 Discussion

In this chapter, a memory-based localization method was introduced that works on compass data, horizontal bearings, and odometry data.

Odometry Correction An important part of the system is the odometry correction which uses vision-based compass data and compensates for error in the rotation component. The idea is, to overcome the faultiness of odometry data based on intrinsic measurement by continuously incorporating rotation data obtained by visual perception. In the RoboCup example, this works very good because field lines are seen in almost every image. While the memory-based approach helps to avoid the accumulation of systematic error in general, the correction is particularly helpful when the robot moves between the observation of bearings used for position estimation.

Localization The three advantages of *memory-based state estimation*, as outlined in chapter 3, are its ability to cope with sparse information, with kidnappings, and with systematic error in the state transition model. These advantages are verified by the experiments. The system is able to process the sparse bearing information. Because of the good odometry correction, there is no systematic error to handle. However, the ability to cope with kidnappings is shown by the fact that, in all experimental runs, the position estimation is reliable after the first couple of frames containing goal posts.

a)



b)

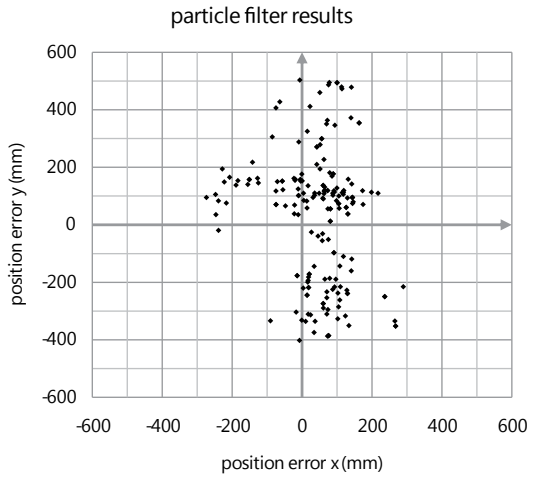
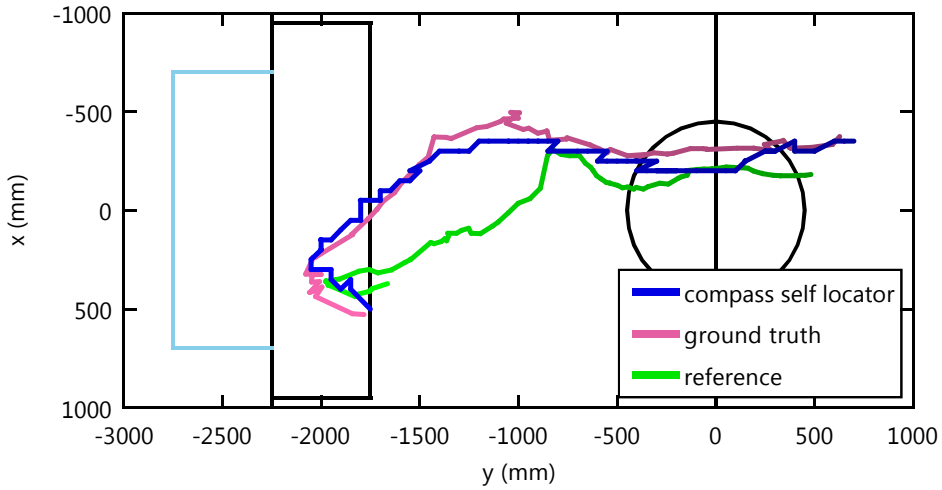


Figure 6.7: *Localization error distribution.* The diagrams show the distribution of localization error. Each dot represents a single position estimate's offset to the corresponding ground truth pose, which is the localization error. a) Error distribution for memory-based localization. b) Error distribution for particle filter results.

a)



b)

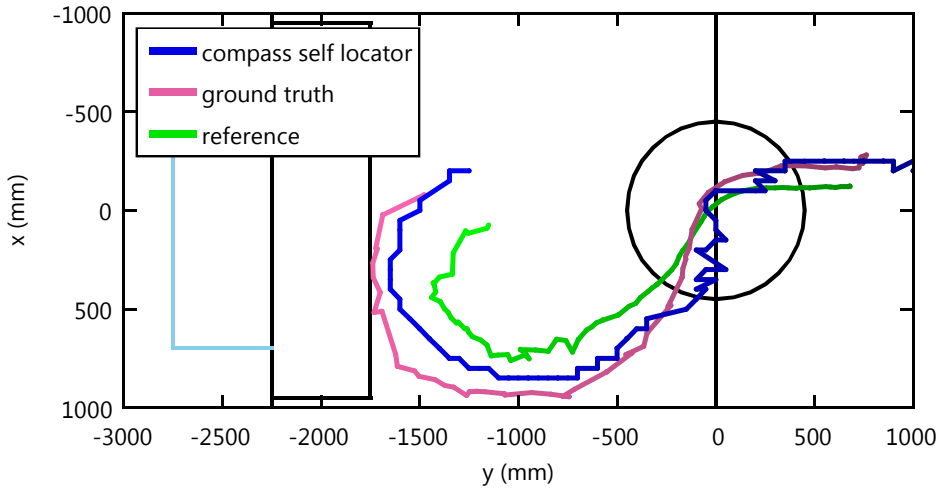


Figure 6.8: *Localization results.* Results of two experimental runs. A humanoid robot walked over a soccer field while following the ball. Pink: real paths the robot walked (obtained by a camera mounted on the ceiling). Blue: localization results. Green: particle filter results.

7 Discussion

7.1 Conclusions

In this thesis, the state estimation problem is discussed. After presenting related work, a new state estimation approach is motivated and described. Based on this new approach, several localization applications for soccer robots are introduced.

Related Work Chapter 2 presents the existing methods based on Bayesian filtering and Hidden Markov Models. These have weaknesses when sparse information has to be processed, kidnappings occur, or there is systematic error.

Methods based on Bayes filters accumulate data gained from perceptions and actions in the *belief*. The belief is a representation of the probability density function that describes the state. All such methods approximate this function in one or another way. The weaknesses of the different approaches are caused by the data-loss which comes with the approximation. The degree of data-loss depends on the problem's specifics and the approximation method. In general, non-parametric filters can support a wider range of problems than parametric filters. However, increasing the approximation accuracy leads to higher computation times.

The Memory-Based Paradigm The main contribution of this thesis is the *memory-based paradigm*, given in chapter 3. It is a guide to how to solve state estimation problems. Memory-based state estimation reuses aspects of the Bayesian techniques and is inspired by the principles of ship navigation at sea. The key difference to techniques based on Hidden Markov Models is how information is stored and processed. Methods which follow the *memory-based paradigm* store perceptions and actions, and calculate the state or aspects of it when needed.

Important properties of this approach are: its generality (independence of specific state estimation problems), the usage of forward models (which are easily obtainable), the capability to cope with systematic error, the capability to quickly recover from kidnapping, and the capability to process sparse data.

Memory-Based State Estimation Applications This thesis describes two localization methods for robot soccer which follow the *memory-based paradigm*. They are able to process the sparse information provided by landmark bearings and can cope with unreliable odometric information. The method given in chapter 4 is able to localize using only horizontal bearings to landmarks. Several landmark selection strategies are evaluated.

The method given in chapters 5 and 6 incorporates information obtained from field lines. Based on a novel image analysis method, a vision-based compass is introduced. Rotation esti-

mates provided by this compass are used to correct faulty odomety data and are combined with horizontal bearings for position estimation.

All these methods are validated with experiments on real robot data obtained from four-legged and humanoid robots. The localization results were compared to ground truth data.

7.2 Outlook

This work shows alternatives and supplementations to state estimation techniques based on Hidden Markov Models and Bayesian filtering. Some aspects of the concept introduced in this thesis are worth deeper investigation. The techniques introduced here are a good starting point for further research.

Negative Information The main example for sparse information used in this thesis are landmark bearings. Another example is *negative information*. The notion of negative information and how particle filters can be adapted to be able to process that kind of information are discussed in [35, 37, 36, 38]. Negative information is the information provided by detection sensors about the absence of certain objects. For instance, information like *there is no landmark in the current image* can be considered negative information. Parametric filters are usually not designed to cope with this kind of information. Nonparametric filters need a high resolution to store negative information. This leads to high computational costs. Methods which follow the *memory-based paradigm* are perfectly suited to process negative information. However, this was not done in the examples given in this thesis. Using negative information could further improve localization accuracy on soccer fields.

Percept Selection Strategies In sections 3.3.5 and 4.3, landmark selection strategies are discussed. The strategies given there are of a pragmatic nature. A more thorough examination of the subject could bring forth rules for better percept selection strategies. Some basic work on that has already been done in our research group [52]. Percept selection helps to solve several problems: the fewer percepts to be processed, the higher the computation speed. Selecting the right combination of percepts can also reduce the estimation error. Lastly, the negative effect of kidnappings on the state estimation accuracy can be minimized by not selecting percepts prior to kidnappings. The research question of interest is how to obtain a good subset of all percepts.

Optimization Techniques Any implementation of memory-based state estimation has to calculate the minimum position of the function given in definition 3.4. Several methods are suggested in this thesis. In section 3.3.5, a grid-based approach and two methods based on gradient descent are introduced. The experiments in chapters 4, 5, and 6 use the grid-based method. For cases with more dimensions, the chosen optimization algorithm might be more important. The interesting question here is how the problem's specifics can be used to select an optimization algorithm and its parameters.

7.3 Final Remarks

It is amazing how the capabilities of robots have improved since my first encounter with RoboCup in 2001. Looking at how robot soccer games have changed since than, I'm confident that the RoboCup initiative will achieve its aim and hold a match between humanoid robots and the 2050 human soccer world champion. I am happy that with this thesis the goal that the robots will win comes a bit closer.

Acknowledgements

This thesis was created while I was a research assistant at the *Artificial Intelligence lab* at *Humboldt-Univerität zu Berlin*. I would like to thank my supervisor Prof. Dr. Hans-Dieter Burkhard who is an enthusiastic advocate of robotics and the *RoboCup*. He always encouraged me to question existing methods and to search for new solutions.

Financial support for me and our project came from the *German Research Foundation (DFG)*. The special priority program 1125 on *Cooperating Teams of Mobile Robots in Dynamic Environments* not only brought us monetary support but also contacts to outstanding researchers in Germany.

My first contact to robotics is due to Martin Loetzsch who talked me into the university's RoboCup team. Joscha Bach, Uwe Düffert, Ralf Berger and Michael Gollin helped me get started with robotics, to whom I am especially grateful.

I would also like to thank all *Aibo Team Humboldt* members: Benjamin Altmeyer, Kateryna Gerasymova, Daniel Göhring, Viviana Goetzke, Andreas Heinze, Jan Hoffmann, Thomas Krause, Heinrich Mellmann, Michael Spranger, Irene Winkler, and Wojciech Wojcikiewicz, for being such an outstanding team.

The efforts of the *GermanTeam* to create a solid robotic software platform made the implementation of the techniques described in this thesis possible. I want to thank Max Risler and Thomas Röfer for being such great cooperation partners. I learned how a good team works in the *GermanTeam*.

I would also like to thank the *Simulation, Systems Optimization, and Robotics Group* members at *Technische Universität Darmstadt*, *Department of Computer Science*. Special thanks go to Martin Friedmann, Stefan Kohlbrecher, and Karen Petersen who helped me recording data with their fantastic humanoid robot.

I thank Andreas Wollstein and Manfred Hild for inspiring scientific discussions and D.D. for his language consulting.

For their support throughout the years, I thank my parents and Eike.

Bibliography

[1] Arnaud, Doucet, and Johansen. A tutorial on particle filtering and smoothing: Fifteen years later. Technical report, 2008.

[2] A. Bais and R. Sablatnig. Landmark based global self-localization of mobile soccer robots. In *Computer Vision - ACCV 2006*, volume 3852 of *Lecture Notes in Computer Science*, pages 842–851. Springer Berlin / Heidelberg, 2006.

[3] A. Bais, R. Sablatnig, J. Gu, Y. M. Khawaja, M. Usman, G. M. Hasan, and M. T. Iqbal. Stereo vision based self-localization of autonomous mobile robots. In *Robot Vision*, volume 4931 of *Lecture Notes in Computer Science*, pages 367–380. Springer Berlin / Heidelberg, 2008.

[4] A. Bais, R. Sablatnig, J. Gu, and S. Mahlknecht. Active single landmark based global localization of autonomous mobile robots. In *Advances in Visual Computing*, volume 4291 of *Lecture Notes in Computer Science*, pages 202–211. Springer Berlin / Heidelberg, 2006.

[5] Baltic Nest Institute. Depth distribution in the baltic sea. http://nest.su.se/baltic96/depth.htm, 1996.

[6] D. Becker, J. Brose, D. Göhring, M. Jüngel, M. Risler, and T. Röfer. GermanTeam 2008: The german national robocup team. In L. Iocchi, H. Matsubara, A. Weitzenfeld, and C. Zhou, editors, *RoboCup 2008: Robot Soccer World Cup XII*. Springer, 2008.

[7] A. T. Bharucha-Reid. *Elements of the Theory of Markov Processes and Their Applications*. Dover Publications, 1997.

[8] B. Bollobas. *Modern Graph Theory*. Springer, 1998.

[9] V. Braitenberg. *Vehicles: Experiments in Synthetic Psychology*. MIT Press, Cambridge, MA, USA, 1984.

[10] A. Bredenfeld, H.-D. Burkhard, M. Riedmiller, and R. Rojas. Ki auf dem fußballfeld. *c´t*, 13:102–109, 2006.

[11] I. N. Bronstein, K. A. Semendjajew, G. Musiol, and H. Muehlig. *Taschenbuch der Mathematik*. Thun, Frankfurt am Main: Verlag Harri Deutsch, 1993.

[12] R. Brunn, U. Düffert, M. Jüngel, T. Laue, M. Lötzsch, S. Petters, M. Risler, T. Röfer, K. Spiess, and A. Sztybryc. Germanteam 2001. In A. Birk, S. Coradeschi, and S. Tadokoro, editors, *RoboCup 2001: Robot Soccer World Cup V*, volume

2377 of *Lecture Notes in Artificial Intelligence*, pages 705–708, Seattle, WA, 2002. Springer. more detailed in GermanTeam RoboCup 2001. Technical Report (41 pages, http://www.tzi.de/kogrob/papers/GermanTeam2001report.pdf).

[13] R. Brunn and M. Kunz. Ein globales sichtsystem zur unterstützung der entwicklung autonomer fußballroboter. Master's thesis, TU-Darmstadt, FB Informatik, Januar 2005.

[14] H.-D. Burkhard, U. Visser, M. Jüngel, A. Bredenfeld, and T. Christaller. Herausforderung für ki und robotik. *Künstliche Intelligenz: KI*, 20(2):5–11, 2006.

[15] O. Cappé, S. Godsill, and E. Moulines. An overview of existing methods and recent advances in sequential monte carlo. *Proceedings of the IEEE*, 95(5):899–924, 2007.

[16] S. Clark and H. F. Durrant-Whyte. Autonomous land vehicle navigation using millimeter wave radar. In *ICRA*, pages 3697–3702, 1998.

[17] H. L. Committee. Robocup humanoid league. http://www.tzi.de/humanoid, 2010.

[18] M. Deans and M. Hebert. Experimental comparison of techniques for localization and mapping using a bearingonly sensor, 2000.

[19] F. Dellaert, W. Burgard, D. Fox, , and S. Thrun. Using the condensation algorithm for robust, vision-based mobile robot localization. In *Proc. of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR)*, 1999.

[20] F. Dellaert, D. Fox, W. Burgard, and S. Thrun. Monte carlo localization for mobile robots. In *Proceedings of the 1999 IEEE International Conference on Robotics and Automation (ICRA)*, volume 2, pages 1322–1328. IEEE, 1999.

[21] U. Düffert, M. Jüngel, T. Laue, M. Lötzsch, M. Risler, and T. Röfer. Germanteam 2002. In G. A. Kaminka, P. U. Lima, and R. Rojas, editors, *RoboCup 2002: Robot Soccer World Cup VI*, volume 2752, pages 114–124. Springer, 2003. more detailed in GermanTeam RoboCup 2002. Technical Report (178 pages, http://www.tzi.de/kogrob/papers/GermanTeam2002.pdf).

[22] J. Diard, P. Bessiere, and E. Mazer. A survey of probabilistic models using the bayesian programming methodology as a unifying framework, 2003.

[23] A. Doucet, N. De Freitas, and N. Gordon, editors. *Sequential Monte Carlo methods in practice*. 2001.

[24] H. Everett. *Sensors for Mobile Robots*. A K Peters/CRC Press, 1995.

[25] D. Fox, W. Burgard, F. Dellaert, and S. Thrun. Monte Carlo localization: Efficient position estimation for mobile robots. In *Proc. of the National Conference on Artificial Intelligence*, 1999.

[26] D. Fox, W. Burgard, F. Dellart, and S. Thrun. Monte carlo localization: Efficient position estimation for mobile robots. In *Proceedings of the Sixteenth National Conference on Artificial Intelligence and Eleventh Conference on Innovative Applications of Artificial Intelligence (AAAI)*, pages 343–349. The AAAI Press/The MIT Press, 1999.

[27] D. Fox, W. Burgard, and S. Thrun. Active markov localization for mobile robots. *Robotics and Autonomous Systems*, 1998.

[28] D. Fox, W. Burgard, and S. Thrun. Markov localization for reliable robot navigation and people detection. In *Proceedings of the Dagstuhl Seminar on Modelling and Planning for Sensor-Based Intelligent Robot Systems, 1999*, 1999.

[29] D. Fox, J. Hightower, L. Liao, D. Schulz, and G. Borriello. Bayesian filtering for location estimation. *PERVASIVE computing*, pages 10–19, July–Sept. 2003.

[30] M. Friedmann, K. Petersen, S. Petters, K. Radkhah, D. Thomas, and O. von Stryk. Darmstadt dribblers: Team description for humanoid kidsize league of robocup 2008. Technical report, Technische Universität Darmstadt, 2008.

[31] Z. Ghahramani. An introduction to hidden markov models and bayesian networks. pages 9–42, 2002.

[32] N. J. Gordon, D. J. Salmond, and A. F. M. Smith. Novel approach to nonlinear/non-gaussian bayesian state estimation. *Radar and Signal Processing, IEEE Proceedings F*, 140(2):107–113, 1993.

[33] J. Grey and D. Caldwell. *Advanced Robotics & Intelligent Machines (IEE Control Engineering Series)*. Institution of Engineering and Technology, 1996.

[34] J.-S. Gutmann and D. Fox. An experimental comparison of localization methods continued. In *Proceedings of the 2002 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE, 2002.

[35] J. Hoffmann, M. Spranger, D. Göhring, and M. Jüngel. Negative information and proprioception in monte carlo self-localization for a 4-legged robot. *Nineteenth International Joint Conference on Artificial Intelligence (IJCAI)*, 2005. Workshop on Agents in Real-Time and Dynamic Environments.

[36] J. Hoffmann, M. Spranger, D. Göhring, and M. Jüngel. Exploiting the unexpected: Negative evidence modeling and proprioceptive motion modeling for improved markov localization. In A. Bredenfeld, A. Jacoff, I. Noda, and Y. Takahashi, editors, *RoboCup 2005: Robot Soccer World Cup IX*, pages 24–35. Springer, 2006.

[37] J. Hoffmann, M. Spranger, D. Göhring, and M. Jüngel. Making use of what you don't see: Negative information in markov localization. In *Proceedings of the 2005 IEEE/RSJ International Conference of Intelligent Robots and Systems (IROS)*. IEEE, 2006.

[38] J. Hoffmann, M. Spranger, D. Göhring, M. Jüngel, and H.-D. Burkhard. Further studies on the use of negative information in mobile robot localization. In *Proceedings of the 2006 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, 2006.

[39] M. Idris, H. Arof, E. Tamil, N. Noor, and Z. Razak. Review of feature detection techniques for simultaneous localization and mapping and system on chip approach. In *Information Technology Journal*, volume 8, pages 250–262, 2009.

[40] S. S. Iyengar. *Autonomous Mobile Robots: Control, Planning, and Architecture (IEEE Computer Society Press Tutorial)*. IEEE Computer Society, 1991.

[41] M. Jüngel. Using layered color precision for a self-calibrating vision system. In D. Nardi, M. Riedmiller, C. Sammut, and J. Santos-Victor, editors, *8th International Workshop on RoboCup 2004 (Robot World Cup Soccer Games and Conferences)*, volume 3276, pages 209–220. Springer, 2005.

[42] M. Jüngel. Bearing-only localization for mobile robots. In *Proceedings of the 2007 International Conference on Advanced Robotics (ICAR 2007), Jeju, Korea,*, August 2007.

[43] M. Jüngel. Memory-based localization. In L. Czaja, editor, *Proceedings of the Concurrency, Specification and Programming Workshop (CS&P 2007)*, volume 2, pages 333–344, 09 2007.

[44] M. Jüngel. Self-localization based on a short-term memory of bearings and odometry. In *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS 2007)*, San Diego, October 2007.

[45] M. Jüngel, J. Hoffmann, and M. Lötzsch. A real-time auto-adjusting vision system for robotic soccer. In D. Polani, A. Bonarini, B. Browning, and K. Yoshida, editors, *7th International Workshop on RoboCup 2003 (Robot World Cup Soccer Games and Conferences)*, volume 3020, pages 214–225. Springer, 2004.

[46] M. Jüngel and H. Mellmann. Memory-based state-estimation. *Fundamenta Informaticae*, 85(1-4):297–311, 05 2008.

[47] M. Jüngel and M. Risler. Self-localization using odometry and horizontal bearings to landmarks. In U. Visser, F. Ribeiro, T. Ohashi, and F. Dellaert, editors, *RoboCup 2007: Robot Soccer World Cup XI Preproceedings*, 2007.

[48] S. Julier and J. Uhlmann. A new extension of the kalman filter to nonlinear systems. In *Int. Symp. Aerospace/Defense Sensing, Simul. and Controls, Orlando, FL*, 1997.

[49] R. E. Kalman. A new approach to linear filtering and prediction problems.

[50] S. Lenser and M. M. Veloso. Sensor resetting localization for poorly modelled mobile robots. In *Proceedings of the 2000 IEEE International Conference on Robotics and Automation (ICRA 2000)*, pages 1225–1232. IEEE, 2000.

[51] P. S. Maybeck. *Stochastic models, estimation and control. Volume I.* 1979.

[52] H. Mellmann. Active landmark selection for vision-based self-localization. In *Proceedings of the Workshop on Concurrency, Specification, and Programming CS&P 2009*, volume Volume 2, pages 398–405, 2009.

[53] R. R. Murphy. *Introduction to AI Robotics*. MIT Press, Cambridge, MA, USA, 2000.

[54] R. Negenborn. Robot Localization and Kalman Filters, 2003.

[55] U. Nehmzow. *Robot Behaviour: Design, Description, Analysis and Modelling*. Springer, 2008.

[56] H. Overschmidt. *Sportbootführerschein See. Lehrbuch inkl. Beilage*. Delius Klasing Vlg GmbH, 2008.

[57] A. Papoulis. *Probability, Random Variables, and Stochastic Processes (McGraw-Hill Series in Electrical Engineering)*. McGraw-Hill Companies, 1984.

[58] L. R. Rabiner. A tutorial on hidden markov models and selected applications in speech recognition. pages 267–296, 1990.

[59] T. Röfer, J. Brose, E. Carls, J. Carstens, D. Göhring, M. Jüngel, T. Laue, T. Oberlies, S. Oesau, M. Risler, M. Spranger, C. Werner, and J. Zimmer. GermanTeam 2006: The German national RoboCup team. In *RoboCup 2006: Robot Soccer World Cup X, Lecture Notes in Artificial Intelligence*. Springer, 2007.

[60] T. Röfer, J. Brose, D. Göhring, M. Jüngel, T. Laue, and M. Risler. GermanTeam 2007: The german national robocup team. In U. Visser, F. Ribeiro, T. Ohashi, and F. Dellaert, editors, *RoboCup 2007: Robot Soccer World Cup XI Preproceedings*, 2007.

[61] T. Röfer, R. Brunn, S. Czarnetzki, M. Dassler, M. Hebbel, M. Jüngel, T. Kerkhof, W. Nistico, T. Oberlies, C. Rohde, M. Spranger, and C. Zarges. GermanTeam 2005: The German national RoboCup team. In A. Bredenfeld, A. Jacoff, I. Noda, and Y. Takahashi, editors, *RoboCup 2005: Robot Soccer World Cup IX*, Lecture Notes in Artificial Intelligence. Springer, 2005. Full team report can be downloaded at http://www.germanteam.org/.

[62] T. Röfer, R. Brunn, I. Dahm, M. Hebbel, J. Hoffmann, M. Jüngel, T. Laue, M. Lötzsch, W. Nistico, and M. Spranger. GermanTeam 2004: The German national RoboCup team. In D. Nardi, M. Riedmiller, C. Sammut, and J. Santos-Victor, editors, *RoboCup 2004: Robot Soccer World Cup VIII*, volume 3276 of *Lecture Notes in Artificial Intelligence*. Springer, 2005. Full team report can be downloaded at http://www.germanteam.org/.

[63] T. Röfer, I. Dahm, U. Düffert, J. Hoffmann, M. Jüngel, M. Kallnik, M. Lötzsch, M. Risler, M. Stelzer, and J. Ziegler. Germanteam 2003. In D. Polani, B. Browning, and A. Bonarini, editors, *RoboCup 2003: Robot Soccer World Cup VII*, volume 3020 of *Lecture Notes in Artificial Intelligence*, pages 114–124, Padova, Italy, 2004. Springer. more detailed in GermanTeam RoboCup 2003. Technical Report (199 pages, http://www.germanteam.org/GT2003.pdf).

[64] T. Röfer and M. Jüngel. Vision-based fast and reactive monte-carlo localization. In D. Polani, A. Bonarini, B. Browning, and K. Yoshida, editors, *Proceedings of the 2003 IEEE International Conference on Robotics and Automation (ICRA)*, pages 856–861. IEEE, 2003.

[65] T. Röfer and M. Jüngel. Fast and robust edge-based localization in the sony four-legged robot league. In D. Polani, A. Bonarini, B. Browning, and K. Yoshida, editors, *7th International Workshop on RoboCup 2003 (Robot World Cup Soccer Games and Conferences)*, volume 3020, pages 262–273. Springer, 2004.

[66] T. Roefer, R. Brunn, I. Dahm, M. Hebbel, J. Hoffmann, M. Juengel, T. Laue, M. Loetzsch, W. Nistico, and M. Spranger. GermanTeam 2004: The German national RoboCup team. In D. Nardi, M. Riedmiller, C. Sammut, and J. Santos-Victor, editors, *RoboCup 2004: Robot Soccer World Cup VIII*, volume 3276 of *Lecture Notes in Artificial Intelligence*. Springer, 2005. Full team report can be downloaded at http://www.germanteam.org/.

[67] D. Seidman. *The Complete Sailor - Learning the art of Sailing*. International Marine, Camden, Me., 1995.

[68] K. Seng and C. L. Kleeman. Sonar based map building for a mobile robot. In *Proceedings of the IEEE International Conference on Robotics and Automation*, pages 1700–1705, 1997.

[69] R. Siegwart and I. R. Nourbakhsh. *Introduction to Autonomous Mobile Robots (Intelligent Robotics and Autonomous Agents series)*. The MIT Press, 2004.

[70] R. Smith, M. Self, and P. Cheeseman. *Estimating uncertain spatial relationships in robotics*, pages 167–193. Springer-Verlag New York, Inc., New York, NY, USA, 1990.

[71] J. Snyman. *Practical Mathematical Optimization: An Introduction to Basic Optimization Theory and Classical and New Gradient-Based Algorithms (Applied Optimization)*. Springer, 2005.

[72] J. Sola, A. Monin, M. Devy, and T. Lemaire. Undelayed initialization in bearing only slam, 2005.

[73] H. W. Sorenson, editor. *Kalman Filtering: Theory and Application (IEEE Press selected reprint series)*. IEEE, 1985.

[74] M. Sridharan, G. Kuhlmann, and P. Stone. Practical vision-based monte carlo localization on a legged robot. In *IEEE International Conference on Robotics and Automation*, April 2005.

[75] S. Thrun. Probabilistic robotics. *Commun. ACM*, 45(3):52–57, 2002.

[76] S. Thrun. Robotic mapping: A survey. In *Exploring Artificial Intelligence in the New Millenium*. Morgan Kaufmann, 2002.

[77] S. Thrun, W. Burgard, and D. Fox. *Probabilistic Robotics (Intelligent Robotics and Autonomous Agents)*. The MIT Press, 2005.

[78] S. Thrun, D. Fox, and W. Burgard. Monte carlo localization with mixture proposal distribution. In *Proceedings of the National Conference on Artificial Intelligence (AAAI)*, pages 859–865, 2000.

[79] E. A. Wan and R. Van Der Merwe. The unscented kalman filter for nonlinear estimation. pages 153–158, August 2002.

[80] C.-C. Wang, C. Thorpe, and S. Thrun. Online simultaneous localization and mapping with detection and tracking of moving objects: Theory and results from a ground vehicle in crowded urban areas. In *In Proceedings of the IEEE International Conference on Robotics and Automation (ICRA*, pages 842–849, 2003.

[81] G. Welch and G. Bishop. An introduction to the kalman filter. Technical report, Chapel Hill, NC, USA, 1995.

[82] D. Willemsen. Advanced navigation courses - sailing schools greece and the greek islands. `http://www.sailingissues.com/navcourse0.html`.

[83] J. Wolf, W. Burgard, and H. Burkhardt. Robust vision-based localization for mobile robots using an image retrieval system based on invariant features. In *Proc. of the IEEE International Conference on Robotics and Automation (ICRA)*, 2002.

Selbständigkeitserklärung

Ich erkläre, dass ich die vorliegende Arbeit selbständig und nur unter Verwendung der angegebenen Literatur und Hilfsmittel angefertigt habe.

Berlin, den 13.07.2011 Matthias Jüngel