Chapter 2

Related Work

The following chapter is organized into two parts: In the first part, which consists of sections one to five, we describe current research from an application-based point of view. Here, we do not restrict the focus on computer vision, but ask more generally for perceptional robot navigation techniques including laser range scanners, sonars, etc. In the second part, sections 2.5 to 2.10, we change to an algorithmic point of view and concentrate on computer vision techniques, in particular techniques that relate to our specific RoboCup problem, namely robot self-localization by the field lines.

Navigation is an essential and old problem that has led to the founding of several institutions (i.e. "Deutsche Gesellschaft für Ortung und Navigation"[1](German Institute for Navigation) or the "US Institute of Navigation"[2]). Navigation essentially works by combining some kind of perception with prior knowledge of the environment, such as natural or artificial landmarks or properties of the environment like the magnetic field of the earth. In RoboCup, the environment is the playing field and it is well-known in advance. In different scenarios, i.e a robot exploring a planet, the robot must gain the knowledge itself, that is, it has to build a map that is then used as a frame of reference for localization. This problem of Simultaneous Localization and Map Building (SLAM) is a difficult and important issue in current mobile robotics research. Interestingly, the SLAM problem is, to a high degree, independent of the sensors that are used, and it is helpful to consider not only current research that is based on computer vision but also the work which is related to other sensor systems. Often, methods and means to solve

[1]http://www.dgon.de
[2]http://www.ion.org/

problems in one sensor modality can be transferred to other sensor systems. For instance, techniques that where originally applied using laser range scanners can be transferred to omni-directional vision systems, as it is the case with the “radial scan method”, which will be described later.

2.1 Navigation Using Laser Range Scanners

Using a laser range scanner, a laser beam rotates in a plane and measures distances to objects in the environment. The distance is measured by determining the time between emission and detection of the reflected light. The most common laser range scanners are from SICK AG[3] and RIEGL GmbH[4]. Some of their products are shown in figure 2.1.

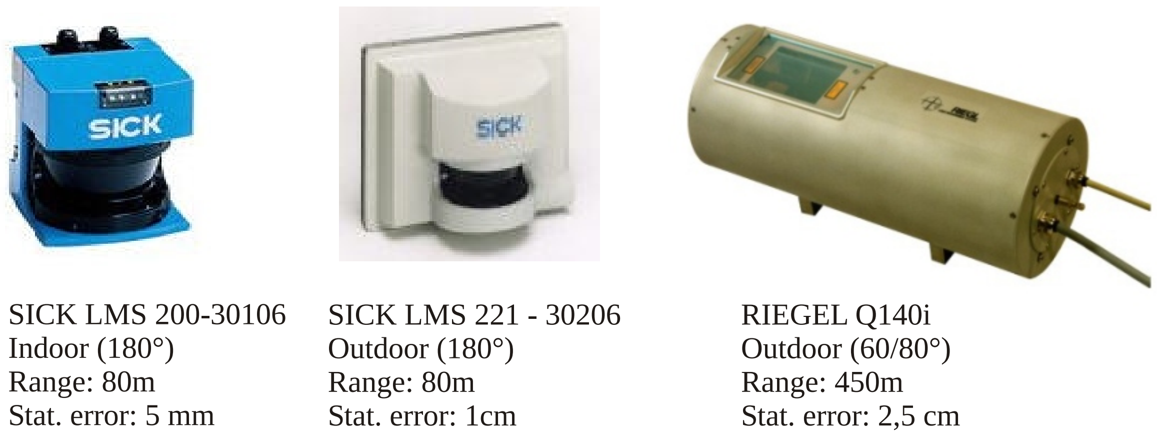

Figure 2.1: The most common laser range scanners are from SICK AG and RIEGL GmbH. Different products for indoor and outdoor applications exist.

A laser range scan consists of a list of points naturally represented in polar coordinates. Each point specifies the angle and corresponding range measurement at that angle. Since the laser beam rotates continuously, the points are sorted by their angles from the very beginning, which simplifies some important operations on the scans, i.e. when trying to extract straight line segments from a scan. Currently, laser range scanners are employed in many robotic systems. At the Australian Center for Field Robotics (ACFR), a utility vehicle (see figure 2.2) has been equipped with a scanning laser range finder and navigation

[3]http://www.sick.de/de/products/categories/auto/en.html
[4]http://www.riegl.com

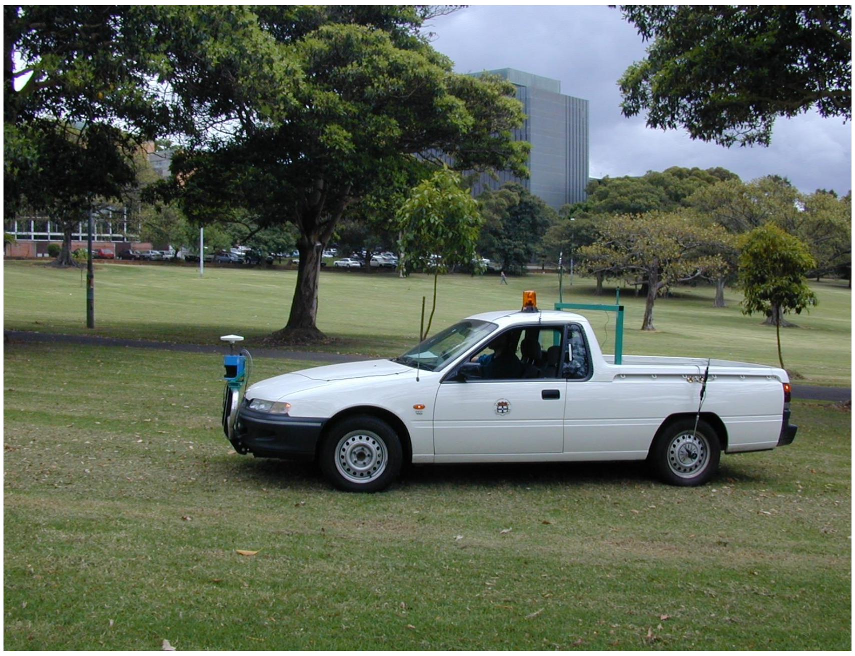

Figure 2.2: At the Australian Center for Field Robotics (ACFR) a utility vehicle is equipped with a laser range scanner for navigation in a park-like environment. Here, the focus of research is on Simultaneous Localization and Map Building (SLAM) algorithms. (with kind permission from Dr. E. M. Nebot)

and map-building algorithms are being investigated in a park-like setting [37, 46]. Most of the teams competing in DARPA's Grand Challenge use laser range scanners for obstacle avoidance, for instance see [80]. Here, the team whose robot completed the 143-mile course across the Mojave Desert from Barstow California to Primm Nevada (the course in 2004) in the least amount of time and at most 10 hours, would win one million dollars. Figure 2.3 shows Carnegie Mellon Univesity's Red Team's vehicle. They performed best among all teams in 2004. The Grand Challenge has revealed some disadvantages of laser range scanners: They can have difficulties with dusty environments. Also, vibrations of the vehicle, tilting the laser scanner, can produce serious problems. In non-planar environments, small angular disturbances can lead to largely varying intersections between the laser beam and the environment, producing highly instable, varying range scans.

Laser range scanners have been successfully used for indoor applications. In [33], laser range scanners were applied to localize robots within a museum. Also, laser range scanners were successfully applied in the MidSize league of RoboCup. In particular, they were useful at the time when the playing field was surrounded by walls. The profile of the

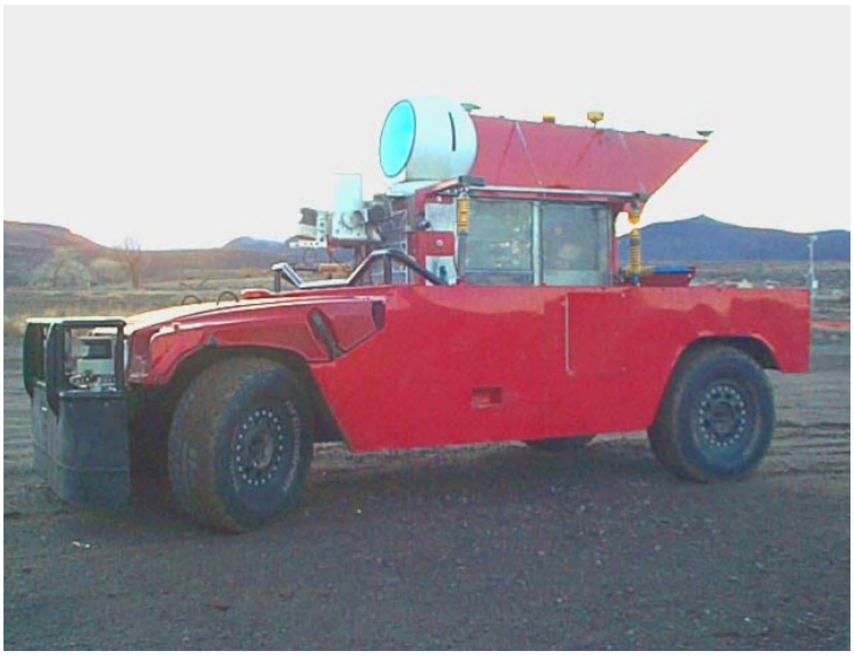

Figure 2.3: The Carnegie Mellon University team's race vehicle, Sandstorm, is a M998 HMMWV vehicle modified for autonomous race driving. It uses three SICK and one RIEDL laser range scanner, primarily for obstacle detection. (with kind permission from Prof. William L. "Red" Whittaker)

walls and the goals could easily and efficiently be extracted from the scans and matched with a model. The work of Steffen Gutmann [38] provides a good overview of existing methods to process the range scans and match them to simple, polygonal models.

Another nice approach was taken in [55]. Here, the idea was to register successive range scans against each other to correct for errors in odometry. The approach is similar to the well-known iterative closest point algorithm (ICP) [9], however it combines two different heuristics to obtain a more precise matching. The approach also works in curved environments.

2.2 Navigation Using GPS and DGPS

The global positioning system (GPS), also called "NAVSTAR", was developed and realized by the US Department of Defense. Its mission is defined as: "Provide the US Armed Forces and Allies with GPS user equipment for precise worldwide navigation anytime, anyplace, anywhere." [1]

Despite its military origins, GPS is currently used by an enormous number of civilian applications. The current worldwide GPS industry is estimated to be approximately \$8 billion and there are about four million GPS users worldwide.

Today, GPS operates with 24 satellites orbiting earth. Their orbits are steadily observed by a master control station located in Colorado-Springs and corrected if necessary. The idea was to constellate the orbits of the satellites in a way that the signals of at least three, better four satellites, can be received anywhere on earth. However, the satellite coverage has not completely succeeded. Leak areas are consistently reported. Moreover, orbits are changed during military conflicts to improve signal reception in specific regions and make reception difficult for the enemy.

Since GPS is a military system, it was not intended to provide full precision for all potential users. Thus, the satellites send signals on two different frequency bands, one for civilian users ($f_1 = 1575,42$MHz) and one for authorized users ($f_2 = 1227,60$MHz). The frequency f_1 is modulated with a C/A-Code (Coarse Acquisition), while f_2 is modulated with a P-Code (Precise Code).

The non-authorized user only has access to the C/A-Code, not to the P-Code. The C/A signal is a PRN-code (Pseudo Random Noise code). It consists of a sequence of 1024 bits that are generated by a simple state machine. At first glance, the sequence seems to be random, but it is not, as it precisely depends on the cycles of the internal clocks, thus the name "Pseudo random".

The satellites and the GPS receivers have synchronized clocks. In the case of the satellites, atomic clocks are used, which are synchronized to so-called GPS Time (offset by a constant from International Atomic Time (TAI)). International clock comparisons are routinely performed via GPS with accuracy on the order of 50 nanoseconds.

Both the satellites and receivers generate the same PRN-code signal, which depends on the clock cycle. When the receiver receives the signal of a satellite, it can calculate its time-of-flight by determining the shift between the received and self-generated signals. This shift can easily be determined by cross-correlation. Having calculated the time-of-flight, the distance to the satellite is given. That is, with a single satellite, the position of the receiver can be limited to a 3D sphere with the center being the position of the satellite. Intersecting the spheres from three satellites, the position on earth can be determined. Using a fourth satellite, even the elevation of the GPS-receiver can be calculated.

Although, the C/A signal does not allow full precision in localization, it was still believed to be too precise by the US military. Hence, the “Selective Availability” (SA) feature was introduced, which artificially scrambled the signals of the satellites. However, in February 2000, SA was ordered to be turned off by former president Bill Clinton. The reason was that GPS had evolved to an important commercial factor and competitive products like the Russian GLONASS should not be favored by potential users. Moreover, the military claimed to have the ability to selectively turn on SA at well-defined regions.

Within the last years, a method called Differential GPS (DGPS) has come into favor. The idea is simple: Since the errors of GPS are correlated within the same region, the idea is to chose a fixed position on land as a reference point, whose correct geographical location is known. A GPS-receiver at this position determines its position as indicated by the GPS satellites, and then calculates the distortion by comparing the GPS-determined position against its known position. The land-based reference point then broadcasts the difference. Users use the broadcast difference to improve the calculations of their positions. As a result, DGPS accuracy and integrity are better than GPS. Typically, a precision of approximately 10m is achieved. Today, a world-wide net of DGPS reference stations exists, which provides the respective correction data. In programs like “WAAS”, “EGNOS” or “MSAS”, the data from different stations is collected and rebroadcast via satellite. Interestingly, the correcting data is broadcast at the same frequency and format as GPS. Thus, in principal, any GPS-receiver can receive the differential information, but only modern receivers are able to use the information for their calculations.

2.3 Navigation Using Radar

Short-range millimeter-wave radar is emerging in the field of mobile robotics [32, 19, 18]. The ”Red Team”, participating in the Grand Challenge 2004, used the NavTech DSC200 Continuous Wave Frequency Modulated radar, which has a range of up to 200 meters. It provides 360° scanning at 2.5Hz and an Ethernet interface to the range measurements. In contrast to vision and laser range scans, RADAR operates at a wavelength that penetrates dust and other visual obscurants but it provides data that is more difficult to interpret. Objects are detected by finding amplitude peaks in the frequency-shifted return signal. The amplitude of the signal can be used to estimate the size, and thus significance, of an object. However, this process can be confounded due to surface properties of objects and the orientation of the object relative to the transceiver.

2.4 Navigation Using Infrared Proximity Sensors

Infrared distance sensors typically can resolve distances between 10 and 80 centimeters. The most popular application is "ROOMBA", a small robot cleaner from iRobot[5] that uses the sensors for navigation. Some teams in the MidSize league of Robocup (i.e. the team Attempto Tübingen) have experimented with these sensors. However the lights illuminating the playing field have an infrared contribution that corrupts the range measurements.

2.5 Navigation Using Ultrasonic Sensors

Ultrasonic sensors have been widely used in indoor applications [28], but they are not adequate for most outdoor applications due to range limitations and bearing uncertainties. In underwater robotics, sonar sensors are often used [22]. Current work on undersea vehicles concentrates on the development of accurate position and attitude estimation, as well as control methods using scanning sonar to provide terrain-aiding information from either a man-made structure or from the sea bed. Key elements of this work include the development of sonar feature models, the tracking and use of these models in mapping and position estimation [84], and the development of low-speed platform models used in vehicle control.

Ultrasonic sensors are subject to noise and sporadic false readings, just like any other sensor system. In most operating environments (except for some shop floors), environmental ultrasonic noise is fairly rare. However, robots with multiple ultrasonic sensors may introduce their own noise, a phenomenon known as crosstalk. Crosstalk differs from other environmental noise because it introduces a systematic error that will repeatedly cause similar erroneous readings. A related kind of "semi-systematic" error is crosstalk that may occur when multiple mobile robots with multiple ultrasonic sensors operate in the same environment. In most indoor applications, crosstalk is much more likely to occur than environmental ultrasonic noise.

In principle, sonar can be a basis for powerful sensing systems, as evidenced by certain animals such as bats or dolphins. However, the sonar systems used for mobile robots are usually rather simple ones, their simplicity and low cost being the very reason for choosing

[5]http://www.rwii.com

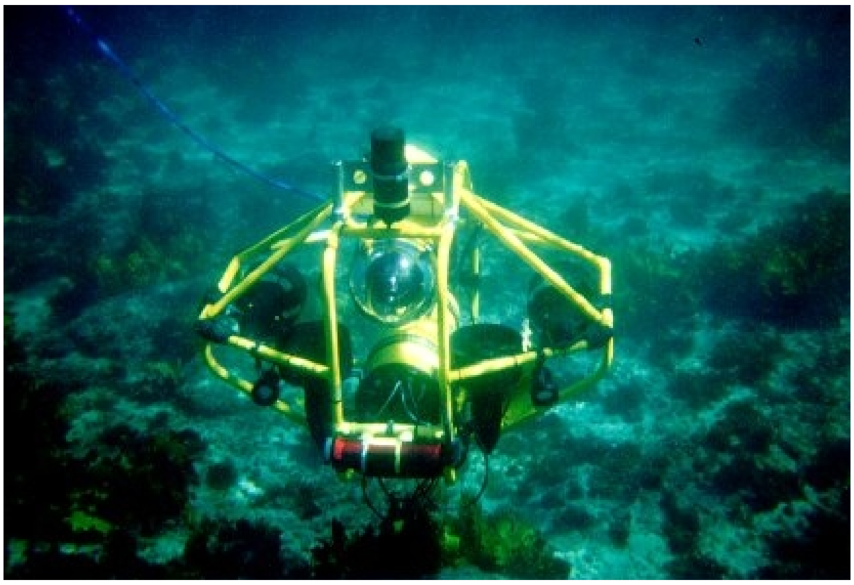

Figure 2.4: The submersible, *Oberon* of the Australian Center for Field Robotics (ACFR)

sonar as a sensing modality. It is not surprising that such systems are severely limited in their performance by low resolution, specular reflections, insufficient dynamic range and other effects.

2.6 Navigation by Vision

A vision system perceives the light of the environment using a two-dimensional field of receptors. The human retina consists of approximately 120 million rods for black-white perception and 6 to 7 million cones for color perception. Current video cameras still are restricted to approximately 1 million receptors. Although higher resolutions are possible (photo cameras that can yield about 10 million pixels already exist), the primary restriction is the data bandwidth that is necessary to transfer video information at 25 to 30 Hertz for real-time vision. With this huge amount of information, the visual sense provides the richest information on the environment from all sensors.

Such information is valuable for recognizing objects, understanding situations and controlling dynamic processes in real-time. When a human drives a vehicle, he mostly relies on his eyes. He uses his sense of vision not only for finding the path on which to drive and for estimating its condition, but also for detecting and classifying external objects such as other vehicles and obstacles, and for estimating their state of motion. Entire situations may thus be recognized, and expectations as to their further development in the near future may be formed.

The same is true for almost all animals. Many use vision as the main sensing modality for controlling their motions. Observing animals, for instance when they are pursuing prey or trying to escape a predator, may give an impression of organic vision systems´ performance in motion control. Some animals, like the Octopus, even use their visual sense to camouflage, changing their skin color or texture similar to their surroundings in order to hide (see figure 2.5).

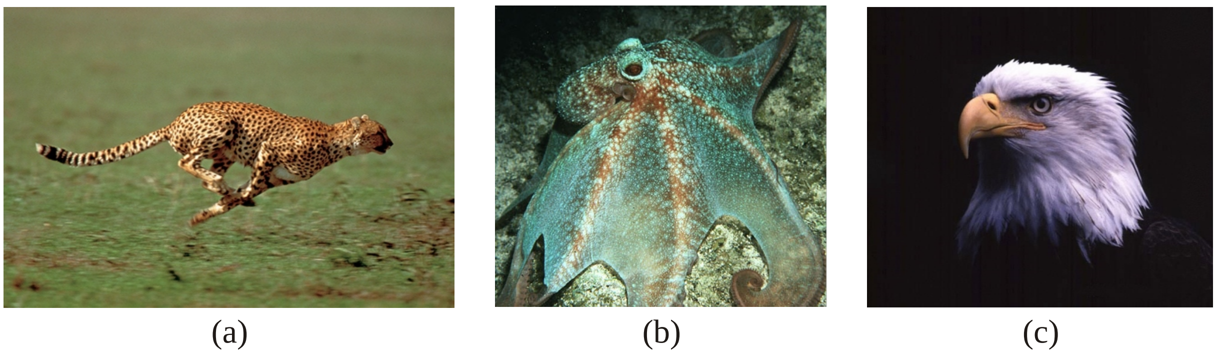

Figure 2.5: Many animals rely on their visual sense. The cheetah (a) uses vision for hunting. The octopus (b) uses its visual sense to camouflage and the eagle(c) is known to have one of the sharpest vision systems (with kind permission from John Forsythe, Copyright John Forsythe)

One apparent difficulty in implementing vision as a sensor modality for robots is the huge amount of data generated by a video camera: As mentioned, about 10 million pixels per second, depending on the vision system used. Nevertheless, it has been shown that even modest computational resources are sufficient for realizing real-time vision systems if a suitable architecture is implemented.

In the late 1970s the idea of motion recognition and vehicle control by computer vision emerged in the context of planetary rover vehicles [78]. Typically, the required computers needed a quarter of an hour for image analysis. The vehicles moved a little and had to wait, standing still as long as the image was analyzed. Each image was interpreted from scratch.

At that time, a different approach was taken by Dickmanns, coming from the field of control engineering. His approach emphasized the importance of a high image frequency (larger than 10Hz) to exploit temporal and visual continuity in the scene. Modeling the change in camera location would allow prediction of the feature positions in the images and search space would be reduced drastically. Extending the well-established methods of the 1960s for recursive estimation of incomplete measurement data, he put forth the "4-D approach", the full state reconstruction in 3-D space and time, which was finally widely accepted in the community dealing with autonomous vehicles [8, 50, 59, 83, 90].

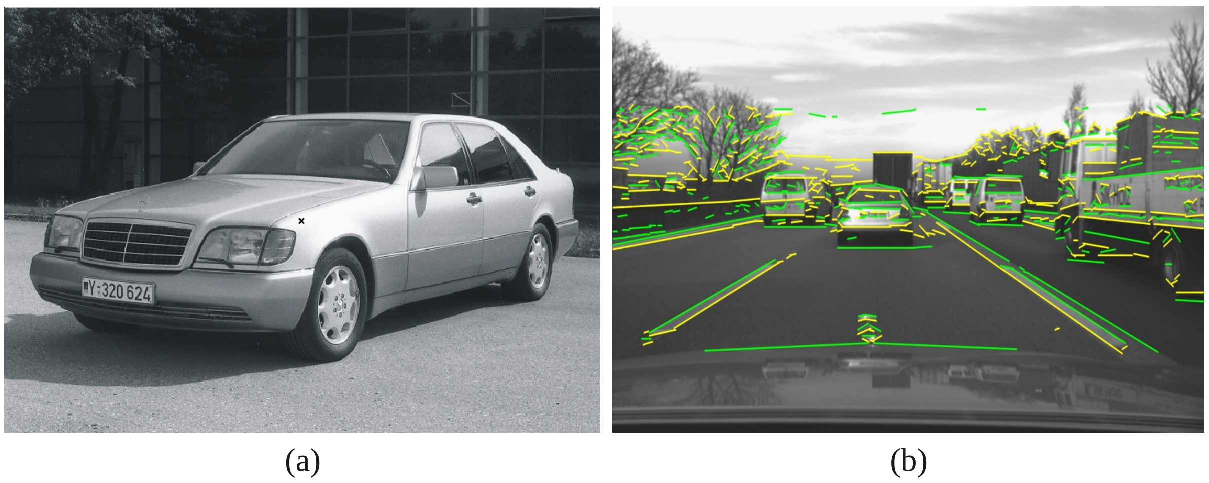

Figure 2.6: (a) The VaMP, Mercedes 500 SEL of the University of the Federal Armed Forces Munich, Germany. It is equipped with four video cameras, a millimeter wave RADAR and eight microprocessors for object recognition and autonomous vehicle guidance. (b) Linking profiles of similar gray values, hypothesis for objects are created. Once the hypotheses exist, tracking is based on the system dynamics approach using motion models of the objects. (with kind permission from Prof. E. D. Dickmanns)

The approach is based on the famous Kalman filter [58]. It combines a motion model with measurements and yields least-squares estimates of the system state. For motion control applications, the measurements are typically taken from sensors such as position encoders, inertial sensors, GPS, etc. Dickmanns was the first who applied the Kalman filter framework with visual sensors [26]. What at the first glance simply looks like an application of the Kalman filter evolved to one of the most promising and elaborated frameworks world-wide.

The primary issue is that instead of processing the whole image in order to retrieve some positional information, the observational model is used to predict the location and appearance of features in the image. In this way, specialized, highly-efficient detectors that investigate only very small portions of the images can be applied. With this approach, real-time state estimation using visual information was possible as early as the 1980s, although computational power was relatively low.

Another benefit of the approach is that it avoids the inverse projection problem. A point in the image plane corresponds to a ray in the 3D world and it is not possible to identify the depth at which the light was reflected by an object using only a single image. This problem results in singular matrices in many approaches. Using Dickmanns' approach, this problem is avoided because the discrepancy between predicted and measured features is directly fed back using a filter matrix, which is based on the motion and observation models of the system. The work of Wünsche [85, 86] was another important milestone, since it showed how features from which the system state can best be estimated can be selected.

The system dynamics approach requires the existence of an initial estimate of the system state, which typically includes the pose of the system. If this estimate is not available or not precise enough, the approach will fail. For this reason, an initial phase is run at the very beginning with the goal of yielding an initial estimate. The real-time phase follows, which is essentially the system dynamics approach. Initial and real-time phases don't have to be processed one after another; they can also be processed in parallel. Here, the real-time phase is executed in each frame while the initial phase can run at a lower rate. In more complex scenarios, as in the case of figure 2.6, the initial phase also has to yield hypotheses for other objects.

The efficiency of the approach lies in the fact that it uses as much prior information about the application as possible. This information includes a model of the environment, the dynamics of the system and the observation model, which describes the geometric relationship between objects in the world and their appearance in the image. It is of immense importance to use this prior knowledge. When a change in the images is detected, this change might be caused by sensing noise, by a movement of the system, or by a change of the environment. Use of prior knowledge enables us to identify which of the components most likely produced the change. Systems not using this information typically try to reduce noise in their position estimates by some smoothing operations. However, smoothing without taking a motion model into account, automatically produces a delay, which is poison for a control application.

The Dynamic Model

The motions of real mass bodies underlie physical laws, which are formulated in the form of differential equations since Newton (1642-1726). Each set of differential equations can be converted into a system of first order differential equations. In this form, the equations simply express how the change of a state value depends on the value of the other states at time t.

$$\dot{\mathbf{x}} = \mathbf{f}(\mathbf{x}(t), \mathbf{u}(t), \mathbf{z}(t), \mathbf{p}) \tag{2.1}$$

Here, the state variables are summarized in the state vector $\mathbf{x}$. The number of states depends on the system. An aircraft will typically have at least 12 components, for example. In the case of the soccer playing robot, which will be investigated later, the vector will comprise 6 components. The state vector also includes the position of the system. The vector $\mathbf{u}$ describes the control input. This vector is important since the dynamics of the system can be predicted by considering the last control commands. Vector $\mathbf{z}$ takes unknown noise into account. Finally $\mathbf{p}$ is a vector of constant parameters.

Using the dynamic model, it is possible to predict the state for a time period T.

$$\mathbf{x}^*(k) = \mathbf{x}^\wedge(k-1) + \int_{t_{k-1}}^{t_k} \mathbf{f}(\mathbf{x}(\tau), \mathbf{u}(\tau), \mathbf{p}) d\tau \tag{2.2}$$

However, the predicted system state $\mathbf{x}^*(k)$ will be subject to errors due to errors in modeling and the unknown noise $\mathbf{z}$.

The Model of the Environment

The goal of using the observation model and model of the environment is to be able to predict the location of features in the images. For instance, the environment can be modeled as a 3D wireframe model. The model has to include information about the material of the objects so that a prediction of the appearance in the images can be made. This model can be regarded as a 3D feature space.

The Observation Model

The observation model describes how the features $p_i(x_{pi}, y_{pi}, z_{pi})$ of the feature space are mapped onto the image plane. Here, x_{pi}, y_{pi}, z_{pi} are the 3D coordinates of the ith feature. This mapping depends on the actual system state $\mathbf{x}$ and other known constants $\mathbf{k}_c$ (focal length, camera mounting position, etc.). This mapping $\mathbf{g}$ is typically nonlinear (i.e in the case of a perspective camera).

$$\mathbf{y} = \mathbf{g}(\mathbf{p}_i, \mathbf{x}, \mathbf{k}_c) \tag{2.3}$$

Here, $\mathbf{y}$ is the vector that describes how the feature appears in the image.

Correcting the Estimate of the System State

Given the predicted system state $\mathbf{x}^*(k)$, we can now predict the appearance $\mathbf{y}^*(k)$ of a specific feature in the image by using equation 2.3 describing the optical mapping:

$$\mathbf{y}^*(k) = \mathbf{g}(\mathbf{p}_i, \mathbf{x}^*(k), \mathbf{k}_c) \tag{2.4}$$

Since $\mathbf{x}^*(k)$ will be subject to errors as described above, $\mathbf{y}^*(k)$ will also be incorrect. However, the error is typically small since the prediction is only made for a small time period T ($T \approx 1/30s$). In order to correct the system state, we first determine the real vector $\mathbf{y}$ from the image. One of the essential advantages of the overall approach is in this step: Instead of processing the whole image, a specialized detector can be applied to only a small part given by $\mathbf{y}^*(k)$. This notion is illustrated in figure 2.7. Depending on the feature, the detector can, for instance, search an edge along a small line fragment. The type of detector can also be predicted. Then the appropriate detector can be selected out of a pre-computed array of detectors and the detector finally measures $\mathbf{y}(k)$. Now we consider the discrepancy between predicted and real measurements

$$\Delta\mathbf{y}(k) = \mathbf{y}(k) - \mathbf{y}^*(k), \tag{2.5}$$

and we feed back the discrepancy in the image plane onto the estimate of the system state.

$$\mathbf{x}^\wedge(k) = \mathbf{x} * (k) + \mathbf{K}(k)\Delta\mathbf{y}(k). \tag{2.6}$$

Here $\mathbf{K}(k)$ is the so-called filter matrix. The Kalman filter theory provides a method to determine this matrix in order to get a new least-squares estimate for the system state. For an in-depth introduction, the book [11] can be recommended. The filter matrix is calculated by:

$$\mathbf{K} = \mathbf{P}^*\mathbf{C}^T(\mathbf{C}\mathbf{P}^*\mathbf{C}^T + \mathbf{R})^{-1} \tag{2.7}$$

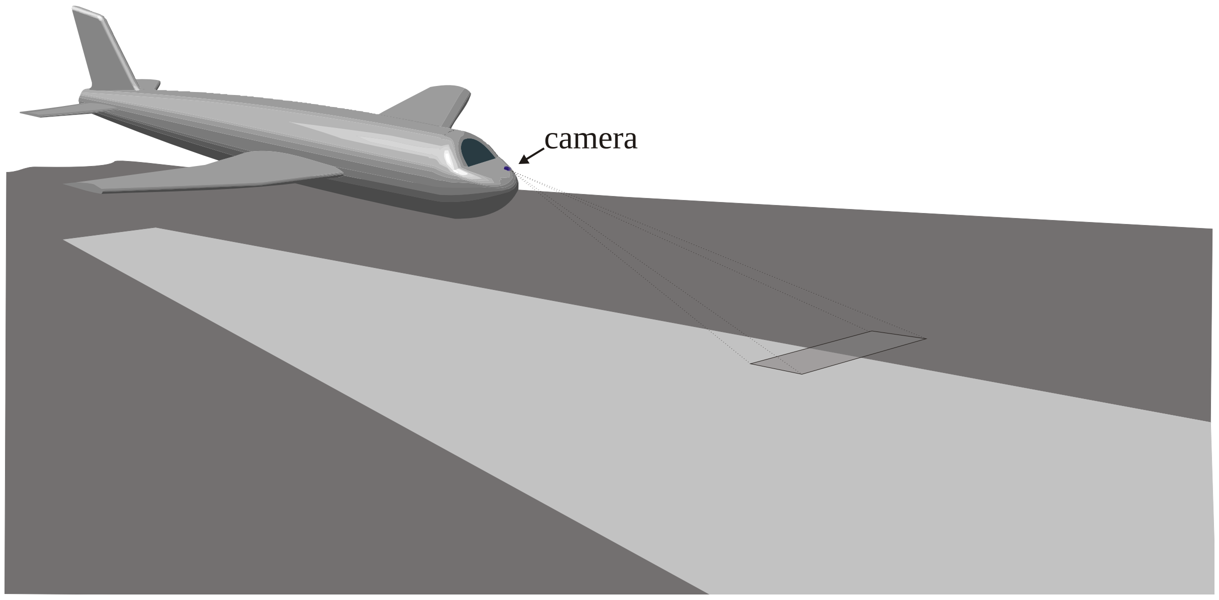

Figure 2.7: An aircraft is equipped with an video camera for automated landing approach. The basic concept of the system dynamics approach is to use a few reliable features to update the estimate of the system state. The rectangular frame at the side of the runway shows the area that corresponds to the image region, which is inspected by a feature detector. While typically more than one feature is used, the approach does not demand the presence of all features covering the degrees of freedom at the same time.

Here $\mathbf{P}^*$ is the actual error noise covariance matrix. The number of columns and rows of the matrix equals the number of system states. $\mathbf{R}$ is the measurement noise covariance matrix. Its number of rows and columns corresponds to the dimension of $\mathbf{y}$. Finally, matrix $\mathbf{C}$ is the Jacobian matrix, which specifies how the appearance of the currently-used features changes to first order when the system state changes. More formally, $\mathbf{C}$ contains the partial derivatives of $\mathbf{y}$ with respect to the system state $\mathbf{x}$. With $\mathbf{y} = (y_1, y_2, ...y_m)^T$ and $\mathbf{x} = (x_1, x_2, ..., x_n)^T$,

$$\mathbf{C} := \begin{pmatrix} \frac{\delta y_1}{\delta x_1} & \frac{\delta y_1}{\delta x_2} & \cdots & \frac{\delta y_1}{\delta x_n} \\ \frac{\delta y_2}{\delta x_1} & \frac{\delta y_2}{\delta x_2} & \cdots & \frac{\delta y_2}{\delta x_n} \\ \vdots & \vdots & \vdots & \vdots \\ \frac{\delta y_m}{\delta x_1} & \frac{\delta y_m}{\delta x_2} & \cdots & \frac{\delta y_m}{\delta x_n} \end{pmatrix} \tag{2.8}$$

Wünsches work [85, 86] shows how the Jacobi-Matrix can be used to decide which features to track. The underlying idea is that the column vectors should be linearly independent so that different features open up separate dimensions. Moreover, features with large partial derivatives should be preferred, since they indicate a close coupling between a component of the system state and the respective feature component. Wünsche developed a method that successively swaps currently-used features with non-used prominent features in order to find an optimal set.

2.7 Typical System Architectures in RoboCup

Most RoboCup teams do not implement the system dynamics approach. The main reason for this is probably the complexity of the approach. Remember that this approach allows the restriction of image analysis to small parts of the images by making use of a dynamic model of the robot and the environment. There are a few systems that restrict image analysis to parts of the image. For instance, in [47], a collection of 1200 jump points are used. They are distributed over the image, and object detection is carried out based on these points. Similarly, in [77], a method has been adopted that samples pixels from the image, adapting the sampling resolution to the expected size of objects. These methods do not access all pixels, but still processes all areas of the image, not concentrating on specific locations where features are expected.

Another example is [69] where prior knowledge about the orientation of the camera is used to predict the orientation of the horizon and respective perpendicular scan lines, which are applied for feature detection. However, they do not use a dynamic model of the robot and environment to feed back the measured discrepancy between prediction and measurement.

In contrast to the above mentioned systems, most systems in RoboCup are based on color segmentation of the whole image in each frame. This is done either in hardware (i.e. by the teams [7, 35]), or software (i.e. by the teams [5, 12, 16, 4]). A popular software method is described in [15].

Most existing systems in RoboCup do not distinguish between an inital phase and a real-time phase, but rather run an initial phase in each frame. The general idea of the two phases is that first hypotheses, for instance for the robot's position, are generated in a computationally more expensive initial phase, and that the change of the state of the hypotheses is kept track of in an efficient real-time phase. The real-time phase uses the hypotheses to predict the locations of features in the images and, therefore, only a small fraction of the image data has to be evaluated.

2.8 Existing Methods for Field Line Extraction

This section will review different methods to extract the field lines from the images. To illustrate the methods, we will use the same input image that is depicted in figure 2.8.

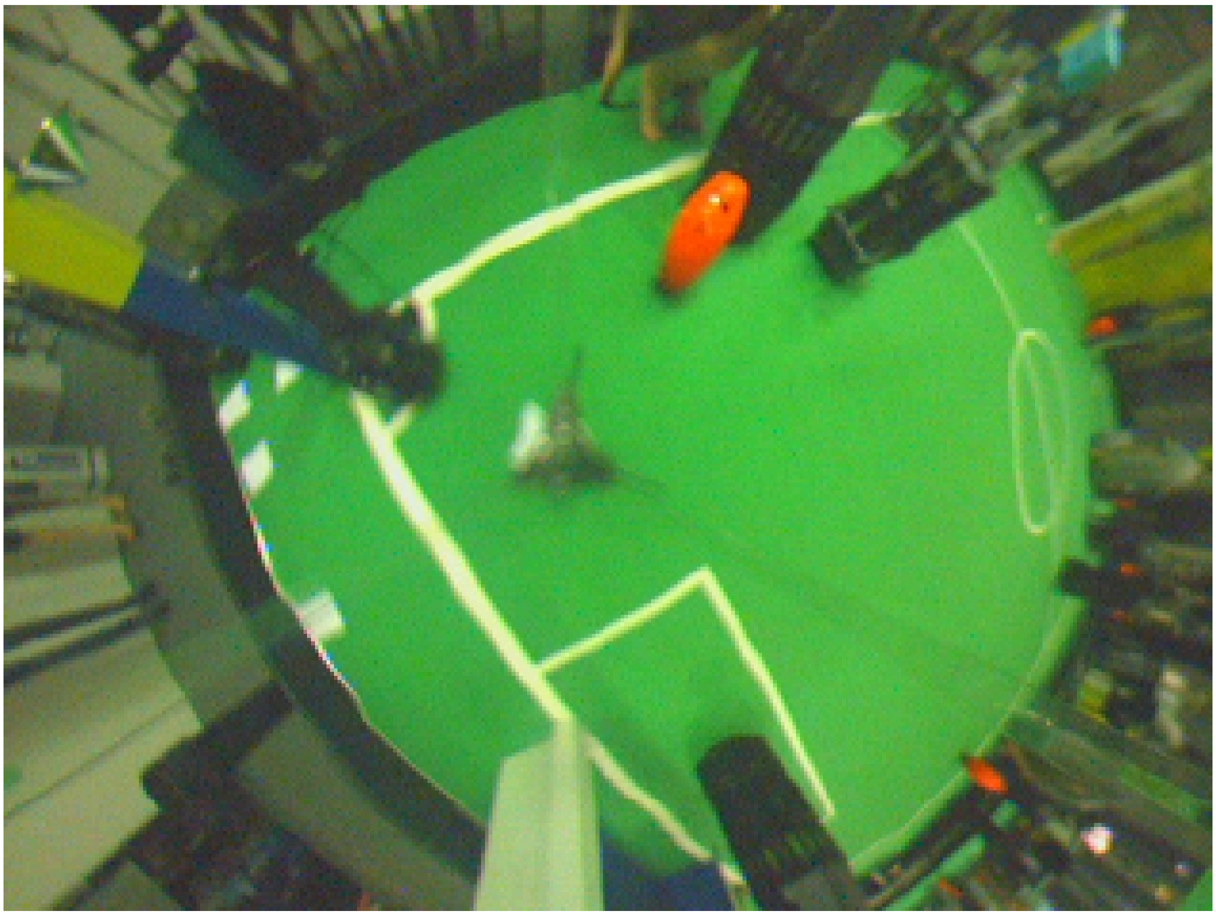

Figure 2.8: This image will be used to review existing methods of line extraction.

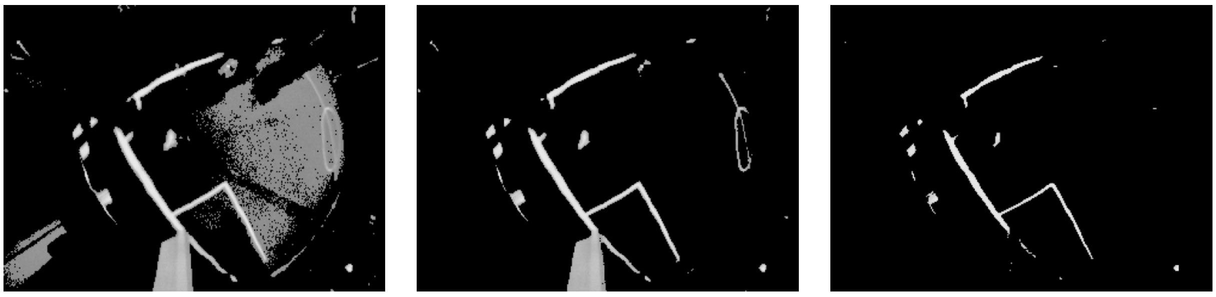
(a) t=140 (b) t=160 (c) t=200

Figure 2.9: Pixels whose intensities are below the threshold t have been drawn black. Applying different intensity thresholds demonstrates that a global threshold does not exist. In image (a) the threshold is too small to separate the centre circle from its surroundings. On the other hand, in image (b) the field line on the bottom right is missed. In image (c) false candidates like the wall of the goal are not present, but at the cost of missing field lines.

2.8.1 Applying Thresholding and Thinning to Extract the Lines

The field markings appear very bright in the images. Therefore, one of the most simple approaches for extraction is to use a threshold t to discriminate between field-line and non-field-line pixels. However, there are other bright objects, like the side walls of the goals for instance, and a pixel whose intensity exceeds t may not necessarily be a field line. Figure 2.9 shows the results of applying different thresholds and demonstrates that finding a global threshold that separates the field lines from the carpet is not possible. The reason is that the lighting changes on the playing field. In the worst case, the intensity of the green carpet at a bright location can exceed the intensity of a field line at a dark location. Moreover, the thresholds will vary depending on the robot's position and orientation on the playing field. We will ignore this problem for the moment and solve it later. Although the method touches all the pixel data of the image, it is very fast. The average running time was 1.3 milliseconds on our test platform[6](C++ code, no usage of MMX or SIMD instructions).

Having extracted candidates for field lines, it is important to reduce the amount of data to reduce the computational load for further processing. One possibility is to employ a thinning algorithm as described in [25]. The pseudo-code and a short description is provided in appendix A.

[6]Pentium III, 800 MHz, image resolution of 320×240

In a simple, non-optimized thinning algorithm, processing takes 4.4 milliseconds. In particular, when using a video format where the intensity is represented separately from the color components, like YUV for instance, the method can be implemented very efficiently. Figure 2.10 shows the result of thinning the lines of figure 2.9b). The advantage of thresholding and thinning is that it is simple and fast. The main disadvantages are that not all lines will be detected due to variations of the lightening and that false lines will be extracted due to bright objects like white walls outside the playing field. Experience has shown that the method worked well in the setup at the competitions in Padova 2003, because the field was brightly illuminated (1000 LUX) from above and the lines were brighter than other white objects outside the playing field (with the exception of the white bars and side walls of the goals). However, the artificial illumination of the playing field is going to be removed by the RoboCup organization within the next few years (in 2004 it was already reduced), and the problem of detecting false candidates will become more serious. On the other hand, the problem of false candidates can be coped with when the position of the robot is known, because the position of the lines in the image can be predicted, and detected lines can be discarded if they differ too much from the predicted ones.

Note, that the techniques described in this section are not applied in the real system. They are only described as a review. The techniques applied will be described beginning with chapter 3.

2.8.2 Using the Canny Edge Detector to Extract the Lines

Using the thresholding method, it is not possible to find a global threshold to extract the field lines, because of the different lighting conditions. However, each line should produce two edges, one at each side of the line, respectively. Detecting these edges should be more independent of the lighting, since for edge detection, we will compare the intensities of neighboring pixels, which should be affected similarly by the lightening. Extensive literature on edge detection is available. The most popular method is the Canny Edge Detector, which is due to [17].

The Canny operator performs several steps that are demonstrated in figure 2.11. First, the image is smoothed with a Gaussian filter. A typical example is the following 5×5 filter:

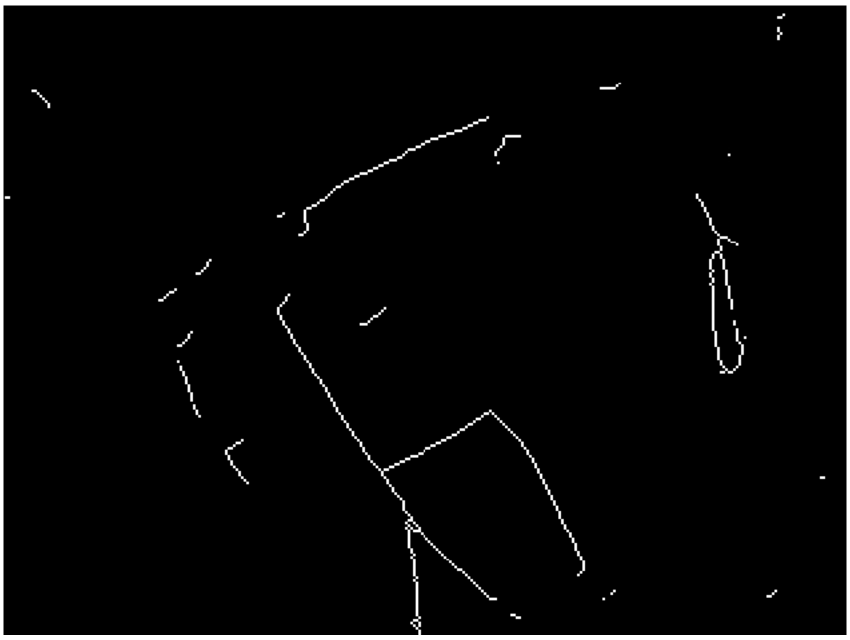

Figure 2.10: Result after having thinned the lines of figure 2.9b)

$$G = \frac{1}{159} \begin{pmatrix} 2 & 4 & 5 & 4 & 2 \\ 4 & 9 & 12 & 9 & 4 \\ 5 & 12 & 15 & 12 & 5 \\ 4 & 9 & 12 & 9 & 4 \\ 2 & 4 & 5 & 4 & 2 \end{pmatrix}$$

Then a simple 2D first derivative operator is applied to the smoothed image to detect regions of the image with high first spatial derivatives. This process requires the convolution of the image with two filters, one for the x and one for the y gradient component, respectively. A common filter pair is the Sobel operator.

$$S_x = \begin{pmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{pmatrix} \qquad S_y = \begin{pmatrix} 1 & 2 & 1 \\ 0 & 0 & 0 \\ -1 & -2 & -1 \end{pmatrix}$$

From the gradient components, we compute the gradient magnitude, which is often approximated by the sum of absolute values (instead of the Euclidean distance) to speed up the computation. Edges give rise to ridges in the gradient magnitude image. The algorithm then tracks along the tops of these ridges and sets to zero all pixels that are not actually on the ridge top so as to produce a thin line in the output, a process known as non-maximal suppression. The tracking process exhibits hysteresis controlled by two thresholds: T_1 and T_2, with $T_1 > T_2$. Tracking can only begin at a point on a ridge higher than T_1. Tracking then continues in both directions along a line from that point until the height of the ridge falls below T_2. This hysteresis helps to ensure that noisy edges are not broken up into multiple edge fragments. Figure 2.12 shows the final results for some different thresholds. Running the detector on our platform took 54 milliseconds. This slow performance is due to the many multiplications that have to be carried out during the three convolutions (5×5 smoothing filter, 3×3 x-Sobel filter, 3×3 y-Sobel filter). To detect the lines, different detectors might be applied. Possible solutions could be based on the Roberts Cross Edge Detector [70], the Sobel Edge Detector [66], the compass operator [71], edge detectors using the Laplacian of Gaussian, Gabor filters [62] or wavelet based solutions [57]. However, applying such a scheme to the whole image is time consuming. In particular, when processing 15 to 30 frames per second for real-time vision, it is important to restrict the area of the image to which a detector is applied.

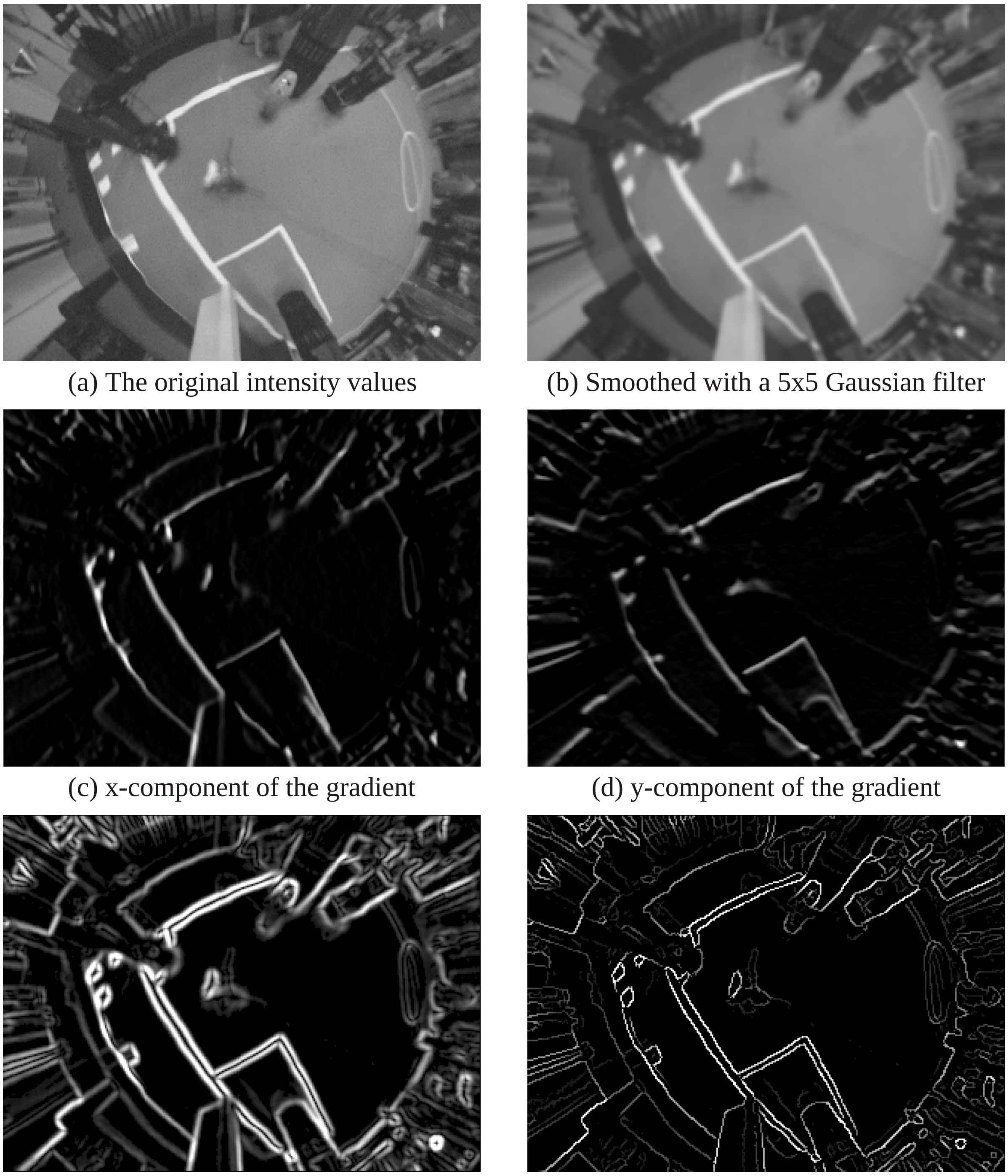
(a) The original intensity values
(b) Smoothed with a 5x5 Gaussian filter
(c) x-component of the gradient
(d) y-component of the gradient
(e) Magnitude of the gradient
(f) Nonmaximum suppression

Figure 2.11: The Canny Edge Detector. The original image (a) is smoothed (b) and the components of the gradient are computed by convolution with a Sobel mask ((c) and (d)). The gradient magnitude is calculated (e) and nonmaximum suppression is performed (f). The result of edge extraction is shown in figure 2.12 for different thresholds.

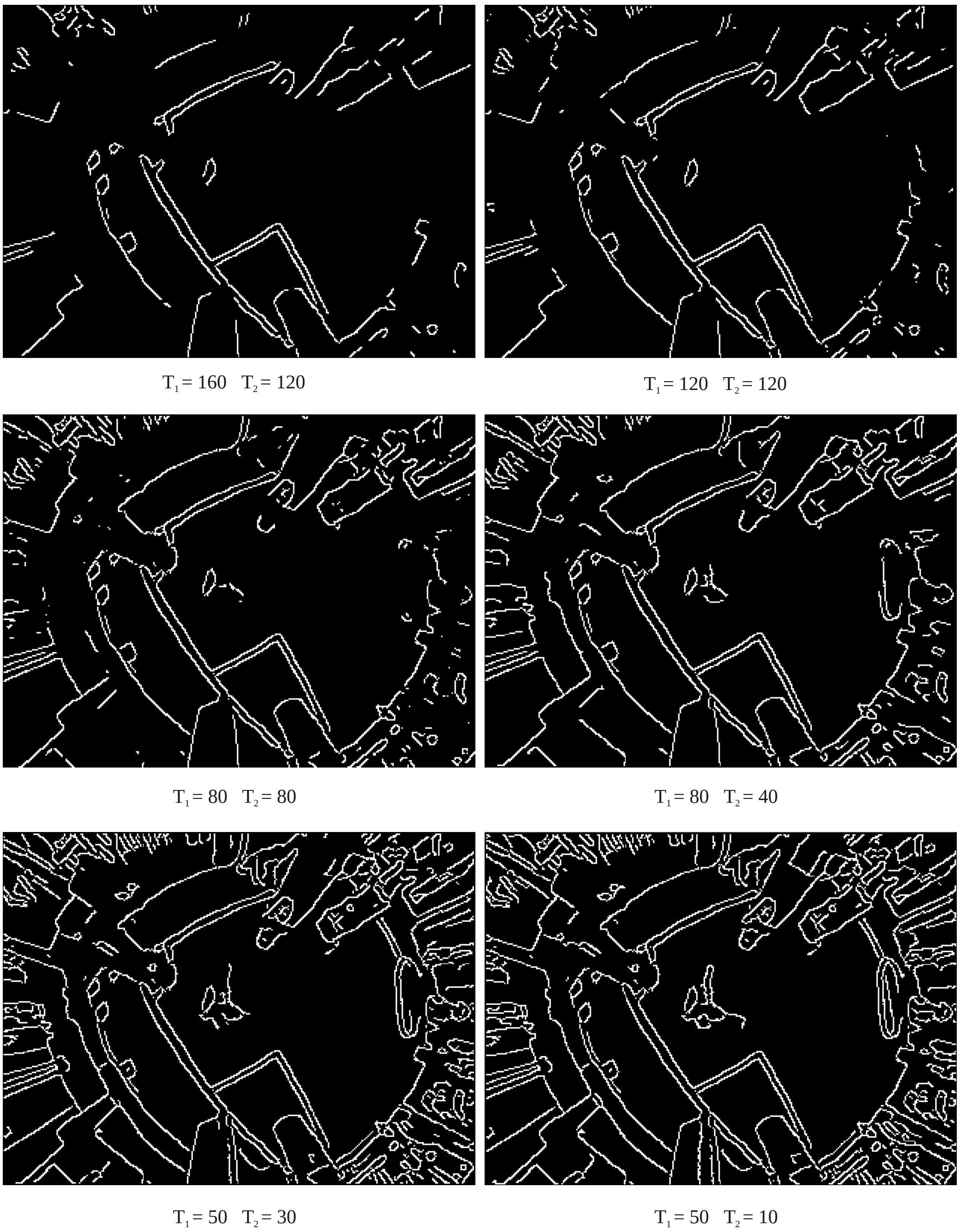
$T_1 = 160 \quad T_2 = 120$
$T_1 = 120 \quad T_2 = 120$
$T_1 = 80 \quad T_2 = 80$
$T_1 = 80 \quad T_2 = 40$
$T_1 = 50 \quad T_2 = 30$
$T_1 = 50 \quad T_2 = 10$

Figure 2.12: Results of the Canny Edge Detector using different thresholds. Edge tracking can only begin at intensities greater than T_1 and continues as long as intensities exceed T_2.

2.8.3 Using the Radial Scan Method to Extract the Lines

The idea of the radial scan method was first described in my diploma thesis [81]. The idea is to employ feature detectors along rays starting at the center of the image. For omni-directional vision systems, we use the center of the mirror in the image, which is identical to the center of the image in the ideal case. It is also possible to let the rays start at some distance from the center, rather that at the exact center. In our omni-directional images, for instance, the robot is projected to the center of the images, and when searching for the field lines, we want to exclude the area of the robot.

The advantages of the rays are first, that only a small fraction of the image data is accessed, and second that a direction is given. Therefore, direction-specific feature detectors can be applied. This is an advantage, because having a given direction, we can look for feature patterns along that direction. Here, feature pattern means that we can look for a specific order of features. To detect the field lines for instance, we can search for a color transition from green to white followed by a transition from white to green. Here, the order of transitions is along the direction of the ray. If this direction was not given, we would have to apply detectors for all possible directions, which would be computationally very expensive. An efficient method of how to find the color transitions is proposed in [29].

Figure 2.13a) shows a radial scan used for the purpose of detecting the white lines. In a first approach, we use the thresholding technique as a detection scheme. Figure 2.13b) shows the detected points. Since the pixel locations of the rays can be pre-calculated, the method can be efficiently implemented. The thresholding technique as applied in the example of figure 2.13, which uses 200 rays of approximately 150 pixels each, runs in 0.62 milliseconds[7], which is more than twice as fast as applying the thresholding technique to the whole image. One disadvantage of the method is that features that do not intersect with the rays are not detected. For instance, a field line that is nearly parallel to the rays will only be intersected a few times. In chapter 3, a new method that does not have this limitation will be proposed. Although the radial scan is no longer applied in our system, it has been adapted by several other teams (i.e Brainstormer Tribots).

[7]Pentium III, 800 MHz, image resolution of 320×240

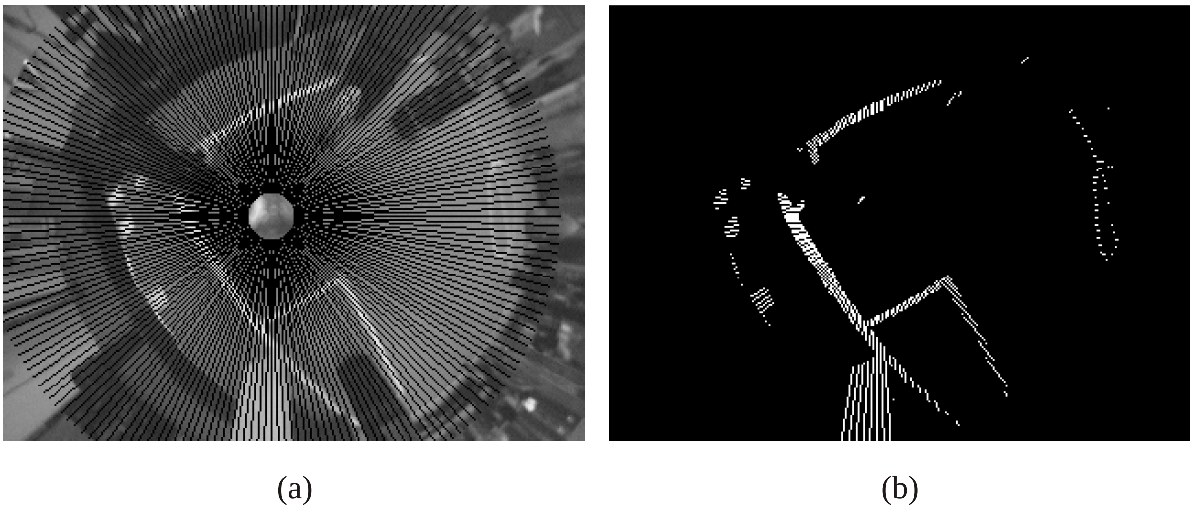

Figure 2.13: The Radial Scan method evaluates image information along rays. (a) Accessed pixels are drawn black. (b) Pixels on the rays that are brighter than $t = 160$ are extracted.

2.8.4 Using a Model to Extract the Lines

Another possibility is to predict the geometry of the field lines in the images, assuming that the distance calibration and position of the robot are known. This approach is similar to the system dynamics approach in that the internal model is projected onto the image plane. However, no direct coupling between measurements and the motion model exists.

It is important to be aware that due to the mirror, even straight lines become curved in the images (except the lines that appear radially). But there are also some originally curved lines as in the case of the center circle and quarter circles at the corners. To approximate this curvature, the field lines could be divided into a set of small, straight line segments. To predict the positions of the lines in the image, the endpoints of each line segment can be transformed from world to image space. This only requires image inspection in the neighborhood of the predicted line segments. The lines can be detected if the difference between predicted and existing lines is small enough.

To inspect the image in the neighborhood of the predicted positions, one method is to search along lines perpendicular to the line segments. This has the same advantage as the radial scan, that the search direction is given. Therefore, one can search for green-white-green transitions, for instance. Figure 2.14 illustrates the overall idea. The advantage of this method is that only a small fraction of the image data must be accessed. This

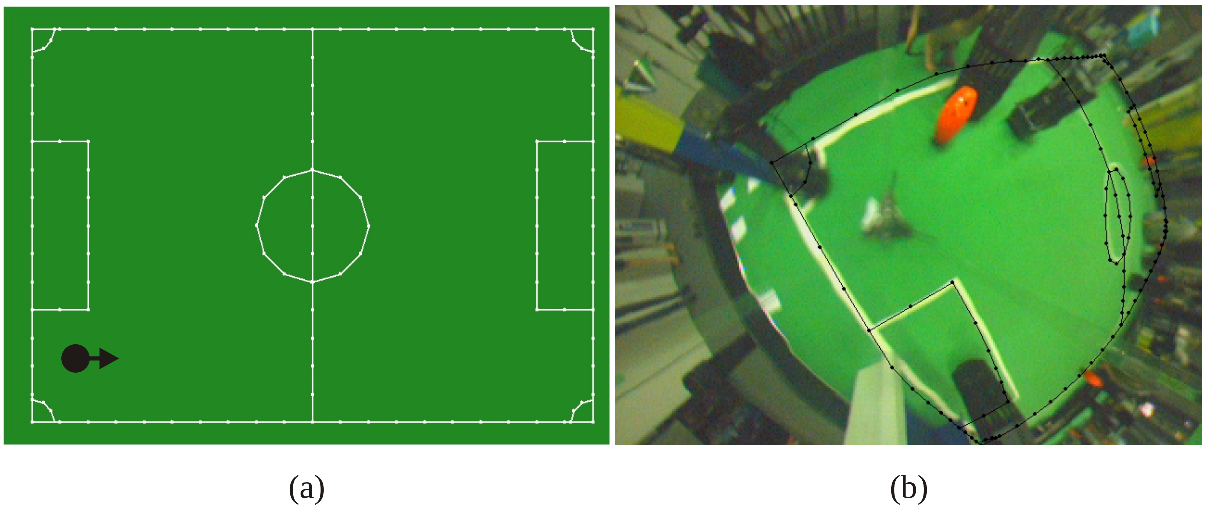
Figure 2.14: Using a model for line detection. (a) The model consists of small line segments whose endpoints are marked by dots. The robot position is indicated by the black disk with the arrow showing the robot's orientation. (b) Knowing the robot's position and having a sufficiently precise distance calibration, the line segments can be projected onto the image and the lines in the neighborhood of these segments can be searched.

spatial selectivity also reduces the number of false candidates. When the robot moves, odometric information can be used to predict the new positions of the lines. As in the system dynamics approach, a disadvantage is that the initial robot position and distance calibration must be known.

Instead of a rigid model as described above, one could adopt the approach of a deformable template as described in [89] where a flexible model consisting of eleven parameters is used to detected the eyes in a face recognition system. The parameters control the position and orientation of the eye, the position and size of the iris and the bounding contour of the eye, which was approximated by two parabolic sections. The deformable template interacts with the image in a dynamic manner in which the parameters are updated by steepest descent.

The philosophy behind using models is to incorporate prior knowledge about the domain of application to accomplish the perception. However, with the long term goal of building flexible, general purpose vision systems, it is obvious that the system itself must learn the models rather than have a fixed number of hard-coded models. A general vision system should be able to perceive even unknown objects.

2.9 Existing Methods for Robot Self-Localization Using the Field lines

The localization problem, that is, determining the robot's pose (position and orientation) relative to its environment, is a key problem in mobile robotics. Its solution plays a fundamental role in various successful mobile robot systems (see e.g. [21, 34, 41, 54, 68, 75, 82] and various chapters in [10, 51]). Knowing the position of the robot is important for navigation and the exchange of information about the environment. In RoboCup, for instance, a robot can inform another robot about the position of the ball if the second robot is not able to see it. However, both robots have to know their respective position on the field.

In general, one can distinguish between *global* and *relative* localization. In *global localization*, the problem is to find the pose of the robot without prior information. In *relative localization*, which is also known as *position tracking*, the position some time Δt ago is known and the task is to update the position due to the movement of the robot within this time interval. In many cases, odometric information of this movement is available and hence, only small errors have to be corrected.

2.9.1 Monte Carlo Localization

For global localization, the robot's observation at a single point in time often is not sufficient to uniquely resolve the robot's position. Therefore, methods that propagate a possibility distribution of the robot's pose have become very popular. In particular the *Monte Carlo Localization*(MCL) [79], which approximates the possibility distribution by using a set of particles, has become the method of choice for global localization. We experimented with MCL in an initial version of our localization system. However, when the feature recognition approach was developed later, we discovered that MCL was no longer necessary. Nevertheless, we shortly describe how we used MCL together with the field lines.

We used 200 particles to represent the probability distribution for the localization of the robot. Each particle has cartesian x, y-coordinates, a heading direction ϕ and represents a hypothesis of the robot's position on the playing field. The probability that the robot is within a certain area is proportional to the density of samples within the area. Initially, all particles are distributed uniformly over the field (see figure 2.15).

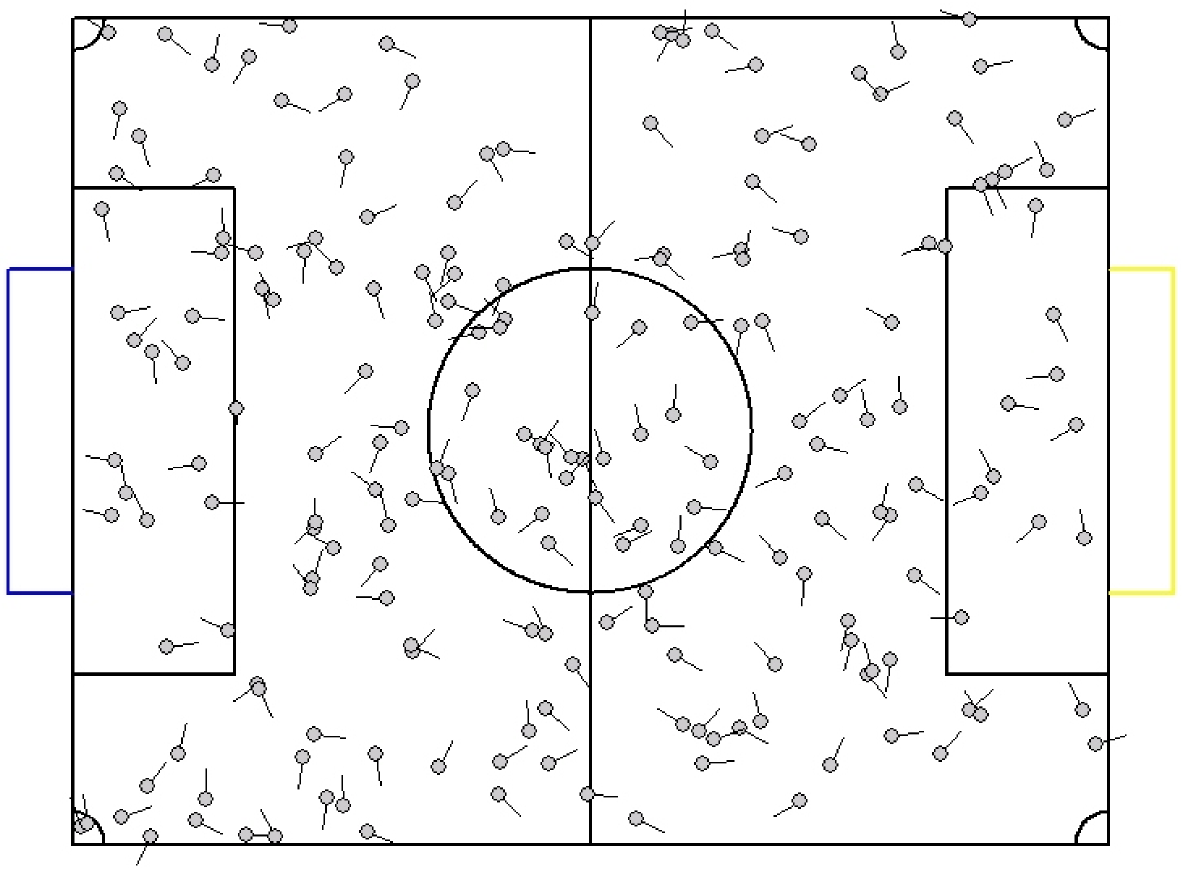

Figure 2.15: Initially the particles are distributed uniformly over the field.

Two kinds of update operations are applied to the sample set, movement updates and sensor updates. In our implementation, movement updates were based on odometric information. When the robot moves within two points in time, odometric information yields a relative translation ΔT and a relative rotation $\Delta\phi$, which approximate the real movement of the robot. Performing a movement update, all particles are translated and rotated according to $\mathbf{\Delta T}$ and $\Delta\phi$ plus additional noise that is modeled by two one-dimensional Gaussians. The first Gaussian G_T, models the noise along the translational direction and G_ϕ models the noise of the rotation. The variances of the Gaussians depend on the length of translation and the amount of rotation. They have been modelled by

$$\sigma_T^2 := 0.005\text{cm}\Delta\mathbf{T} + 4\text{cm}^2 \tag{2.9}$$

$$\sigma_\phi^2 := 0.005\Delta\phi + 0.0005, \tag{2.10}$$

where $\Delta\mathbf{T}$ is measured in centimeters and $\Delta\phi$ in radians. Sensor updates are based on the

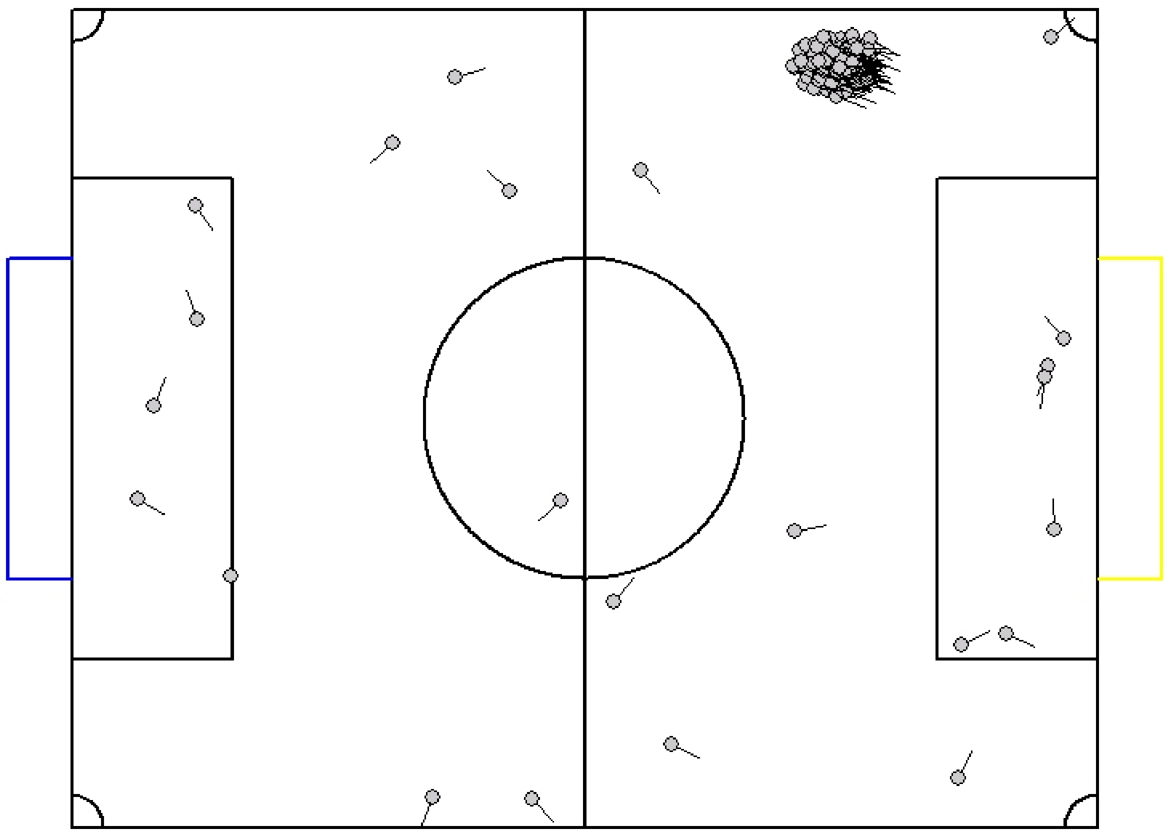

Figure 2.16: Due to movement and sensor updates, the particles begin to cluster. When the robot is localized, a single cluster represents the position of the robot.

line contours. Later, we describe how they are extracted from the images. At this point, it suffices to think of a set of approximately 400 points, each originating from a white field line. Note, that two types of points exist: On the one hand, we have the particles or samples, which represent hypothetical robot positions, on the other hand we have the points, which represent the contours of the field lines that are extracted from the images.

Each sample is given a weight equal to the probability that the line contours have been perceived from the samples pose.

Here, one might think that the computational load is too high to compute the probability for all 200 samples at a real-time video frame rate. However, approximating the probability by a matching distance based on closest point relationships, a lookup table can be pre-computed, which allows an efficient evaluation. This method will be described in more detail later, where it is also used for relative localization (see page 107).

Having weighted the particles, a new set of unweighted particles is generated by randomly sampling from the old set. Here, the probability of a sample being selected is proportional to its weight. By repeating movement and sensor updates, the particles begin to form clusters where the robot's pose is likely to be, and finally a single cluster remains (figure 2.16).

Due to the symmetry of the playing field, two symmetric clusters should always emerge. However, in most cases the number of samples within the two clusters differ slightly due to the random sampling. Since we use relatively few particles (only 200), this introduces a bias for the bigger cluster. Thus, one of the clusters tends to attract more and more particles while the other vanishes. Due to this effect, only one cluster will always remain in practice.

Localization tends to fail if no particle close enough to the robot's position exists. Therefore, a common approach is to take 10 percent of the particles (the ones with the lowest weights) and distribute them uniformly over the field in each step.

We experimented with MCL, also performing a feature recognition in each step, which will be described later. We are able to recognize the center circle and specific corners at the penalty area. Detecting such features gives a direct hint for the robot's position, and we integrated these hints into the Monte Carlo Localization by directly placing particles at the corresponding positions, following the idea of [53].

Later, we discovered that the feature detection allowed us to implement a more efficient global localization method without the need for multiple hypotheses as in MCL. We will describe this method in chapter 4.

Using MCL together with the field lines is not a new idea. In [48], a collection of points on field lines is extracted from the images and MCL is used for localization. However, no attempt has been made to recognize features like the center circle in the field lines. Recognizing such features improves localization, because it allows insertion of particles at the corresponding positions, an idea which is due to [53] and which has been elaborated by [79] in form of the Mixture MCL. The idea of inserting particles is important, because MCL fails if no particle is present at the true position of the robot.

In contrast, MCL together with the recognition of line features has been adopted in [39]. Here, the Hough transform [44] is used for line and circle detection. After having accumulated the evidences, the particles are weighted by predicting the position of the expected features in the Hough spaces and looking up the accumulated values in the parameter space. The approach is interesting because it avoids extracting maxima in the Hough spaces. That is, the features are not extracted explicitly. However, one disadvantage of the approach is that it requires a separate Hough space for each feature, and that a single detected point on a line gives rise to entries in all spaces. Thus, the method is computationally expensive.

In contrast to the above approach, it will be shown later that the field lines can be detected and represented in a way that efficiently allows recognition of different features without using an expensive method like the Hough transform.

2.9.2 Global Localization by Matching Straight Lines

Recently, a localization method based on the field lines but not using MCL has been proposed in [72]. Here, a method that was originally used for localization by laser range scans has been adopted. First, points on field lines are extracted from omni-directional images using the radial scan method (see page 37). The points are represented similarly to laser range scans. However, while a laser range scan has only one point at an angle, the points extracted from the images are organized in different layers. The first layer represents the closest points, the second layer represents the points which follow thereafter, etc. Next, straight line segments are extracted in each layer using an efficient divide and conquer algorithm, and collinear line segments are grouped to form single straight lines. Then, the *LineMatch* algorithm [38] is used to match the lines to a model that also consists of straight lines. The *LineMatch* algorithm essentially tests all combinations recursively, but stops a recursion when two orthogonal lines describe the mapping uniquely.

In this approach, only straight lines have been considered. The center circle has been modelled as a rectangle and aslant lines that could emerge from the circle are excluded by detecting that the angle does not correspond to any of the two main components of line directions. However, if only a few lines are visible, the two main components cannot be determined reliably and the method will fail in such cases.

2.9.3 Relative Localization

Once the position of the robot is known, efficient methods can be used to keep track of its position. The most efficient approach is probably the system dynamics approach that was already described at the beginning of this chapter. In chapter 4, we will sketch how this approach can be implemented for the RoboCup scenario. However, although the approach is efficient, we did not include it in our final system. The reason is that we are performing a feature recognition at 10 frames per second. This feature recognition is of utmost importance for the system for initial pose estimation and recovery from catastrophic localization errors. In RoboCup, such errors can occur, since collisions can happen and robots are sometimes manually placed in other positions.

The feature recognition approach that will be proposed later requires the field contours as input, and we will develop a new method in the next section, which is able to efficiently extract the field lines from the images. This method represents the field lines as chains of points, which is the input format of the later feature recognition approach.

However, even without the recognition of features like the center circle it is possible to use the extracted field lines to correct small errors of the robot's position. For this purpose we implemented a method that is similar to the approach in [55], which was designed for the use of laser range scanners. Here, the task was to recover the relative translation and rotation between two laser range scans, taken at consecutive time points. The relative movement is recovered by minimizing a distance function, which is defined through point-to-point correspondences between the two scans.

In chapter 4, we will propose a new method that is more efficient than the approach of [55]. The increase in efficiency is made possible by exploiting the fact that the playing field of RoboCup is static (the field without the moving objects). It is then possible to pre-compute a force field which is used for registering the field lines to the model. The method will be described in detail later.

2.10 Methods for Feature Detection

In RoboCup, the detection of features or sub-structures such as the center circle are of great value for robust robot self-localization. Once the robot is localized, it is easier to keep track of its position by more efficient methods without trying to recognize features, but for initial localization, features are of immense value. Even when running Monte

Carlo Localization (MCL) [79], features speed up convergence by inserting particles at plausible locations that are indicated by the features [53].

When the distance function, which relates distances in the world to distances in the image, is calibrated badly, uniform methods fail while feature recognition is still possible. Thus, features are an entry point for automatic calibration. Despite of these practical issues, trying to detect features is of intuitive beauty, because it means understanding the data, not just applying a uniform method to a uniform set of data points.

In RoboCup, several approaches to detect features in the field lines have been made. In [72], straight lines are detected, however, no other features, in particular no curved features, are extracted. In [39], straight lines and circles are recognized using the Hough transform [44], but no other features like the quater circles, corners or rectangles are detected. While robust and conceptually nice, the Hough transform is inefficient and for each type of feature, a separate two-dimensional parameter space has to be initialized, evidence has to be accumulated, and the main peeks have to be found. The last step has been optimized in [39] by combining the method with MCL; however, it is still computationally expensive.

Aside from RoboCup, feature detection has been addressed in numerous papers. The simplest approach to feature detection is to directly derive parameters of the feature from the data. For instance, given three non-collinear points, one can easily derive the parameters of a circle containing the points. However, one has to know that the points belong to a circle. If one point is wrong, a false circle will be constructed.

The next class of methods are least-squares fitting methods. Instead of deriving the parameters by the minimum number of required points, they use more points and minimize the overall squared error. Fitting methods exist for lines, circles and ellipses [31, 45], or more general for conic sections. However, as with the first method, it has to be known a priori which points emerge from the feature. Outliers drastically influence the result.

Therefore, attempts for robust detection methods have been made. The most robust methods are probably those based on the Hough transform [44]. Numerous variants for the detection of lines, circles, ellipses and general polygonal shapes exist. Each input item (i.e a point) votes for all possible parameters of the desired feature and the votes are accumulated on a grid. The parameters with the most votes finally determine the feature. This last step is difficult. It requires a decision about the number of votes necessary to consider a feature to be present. Also, if the input data deviates slightly from the perfect feature, accumulation spreads over neighboring cells, and peek detection becomes complex. However, the advantage of Hough transform based methods is that the membership of input data to the feature does not need to be known a priori.

The counterpart to Hough transform methods are methods that probe the parameter space. Instead of beginning with the input data and deriving the parameters in a bottom up fashion, the parameter space is searched in a top down way [64, 13].

Other approaches rely on the initial presence or generation of hypotheses. RANSAC [30] and clustering algorithms such as fuzzy shell clustering (FCS) [23, 24] fall into this category. The advantage of initial hypotheses is that, if there is strong a priori evidence for a hypothesis, outliers can be detected easily by rejecting input data that are too distant. On the other hand, a hypothesis that produces too many outliers can be considered erroneous.

A completely different approach is UpWrite [3, 60], which iteratively builds up small line fragments from points, and higher features like circles and ellipses from the line fragments. In [61], the method has been compared with the Hough transform and comparable robustness has been reported. An approach similar to the UpWrite has been reported in [49] where ellipses emerge from arcs, the arcs from line fragments and the line fragments from points.

Later, we will propose a framework for the detection of many different geometric features within the field lines in the RoboCup domain. We describe how to detect straight lines, corners, T-junctions and the center circle of the playing field. Here, we follow the principle of constructing higher geometric features, such as circles and rectangles from smaller components such as lines, arcs and corners. Typically, different types of higher features are composed of the same kind of lower features. For instance, a rectangle is composed of straight lines and corners, and a T-junction is also composed of two straight lines. This hierarchical organization in which higher-order features share common components, makes the overall recognition process more efficient than approaches that try to detect the individual complex features separately.