Chapter 4

A new Localization Method Using Shape Information

In this section, we develop a new method for robot self-localization in the RoboCup environment using the field lines.

4.1 Three Layers for Robot Self-Localization

Localization is achieved in three layers, which are summarized in table 4.1.

At the bottom layer we have *dead-reckoning*. Here, odometric information supplies information about the wheel rotations, yielding an estimate of the robot's movement. The sensor delay is approximately 8 ms, which is quite small. For small distances, odometric information is very precise. When the robot moves only 1 centimeter, this movement can typically be resolved. However, errors in odometric information accumulate, in particular, when the robot rotates. Even if the rotation is measured with a very small error, this error has a big influence. For instance, when the robot moves straight ahead after a rotation, the initial error in the robot's heading direction will result in a completely wrong position estimation of the robot. Thus, odometric information is fast and precise for small distances, but accumulates to unbounded positional error in the long term.

The second layer, *relative correction*, compensates for this problem. It assumes that an initial estimate of the robots pose is available and it is able to correct small errors using visual information. As visual information we use the field lines as extracted by the previously described region tracking method. When the initial pose estimate is slightly

Layer 1	Odometric information	*by measuring the wheel rotations*
Layer 2	Relative correction	*by matching the field lines to a model*
Layer 3	Feature recognition	*by detecting features like the center circle*

Table 4.1: The three layers of localization

wrong, the extracted field lines do not fit precisely to a model of the lines. The second level calculates how to correct the pose in order to achieve the best fit. The delay of the visual information is typically between 60 and 100 milliseconds. Thus, when the world changes, the visual input will not reflect changes before 60 milliseconds, which is quite long if compared to the odometric delay. Thus, while the first layer is fast but accumulates errors, the second layer is not that reactive, but can correct positional errors.

The first two layers are sufficient to correctly localize the robot for a long time period, provided that an initial correct pose is available. However, when the robot is manually placed in another position (the kidnapped robot problem), when wheel slippage is too large (i.e because of a collision with the ball or another robot) or at the very beginning, when the system is started, this initial position is not available. Therefore, when the initial estimate is wrong, localization fails if solely established on the first two layers. Thus, a third layer exists.

The third layer, *recognition*, performs a feature detection on the extracted line contours. The method is able to detect features, which we refer to as *high-level features*, because recognizing such a feature immediately yields the unique position of the robot in a global coordinate system. Here, "unique" is not completely true, because the line structures on the playing field are symmetric and each high-level feature yields two symmetric positions on the respective sides of the field. However, the correct position can be selected by considering recent valid positions of the robot, or, if these are not available, by considering the goal colors. It is this last layer which makes the proposed localization method robust.

When all the visual tasks including the region tracking, line extraction, and three levels of localization are performed in each frame, 24 frames per second can be processed. However, we discovered that it is not necessary to process all tasks with the same frame rate. We use the following setup: The green regions are tracked with 30 frames per second, because their boundaries are important for ball detection and the ball should be tracked with a high frame rate. This occupies only 2-8 percent of the processing power

(Pentium III 800 MHz). The field lines are extracted only every third frame, resulting in a rate of 10 frames per second. Layers 2 and 3 of the localization also run with this frame rate. Layer 1 uses only odometric information, which is triggered 50 times per second, independent of visual processing.

The remainder of this chapter is organized as follows: Before we start to describe the individual layers, we first describe the general framework for fusing the results of the layers. This fusion process is done by a Kalman filtering approach. It requires the definition of a system state, a dynamic model and a measurement model of the robot. In this context, we also describe specific coordinate transformations that will be used by all three layers. The data fusion is not trivial, since the sensor measurements of the individual layers are subject to different time delays. We explain how these measurement delays are coped with using an explicit modeling of time.

We begin with layer 1, the process of dead-reckoning. Before describing layer 2 and 3, we describe the visual input for these layers: the line contours as extracted by the region tracking algorithm. The line contours are distorted due to the catadioptric mapping and we describe how this distortion is corrected and how the contours are transformed into world coordinates. Then we describe layer 2, *relative correction*, and layer 3, the recognition process.

4.2 The Robot's System State

We specify the robot's position in a cartesian coordinate system whose origin is located at the center of the playing field (see figure 4.1). The cartesian x, y-position together with the orientation ϕ is referred to as the robot's *pose*. The orientation $\phi \in [0, ..., 2\pi[$ is the angle between the x-axis and the heading direction of the robot. Later, when we use a dynamic model of the robot to fuse the results of the three localization layers, three further variables specifying the tanslational (v_x, v_y) and rotational (ω) velocities become important. The robot's pose together with these velocities will be referred to as the robot's system state $\mathbf{x}$.

$$\mathbf{x} = \begin{pmatrix} x & y & \phi & v_x & v_y & \omega \end{pmatrix}^T \tag{4.1}$$

However, when describing the three layers of localization, we only consider the first three components of $\mathbf{x}$ which represent the robot's pose. We will also refer to this vector as the *reduced system state* $\breve{\mathbf{x}}$.

$$\breve{\mathbf{x}} = \begin{pmatrix} x & y & \phi \end{pmatrix}^T \tag{4.2}$$

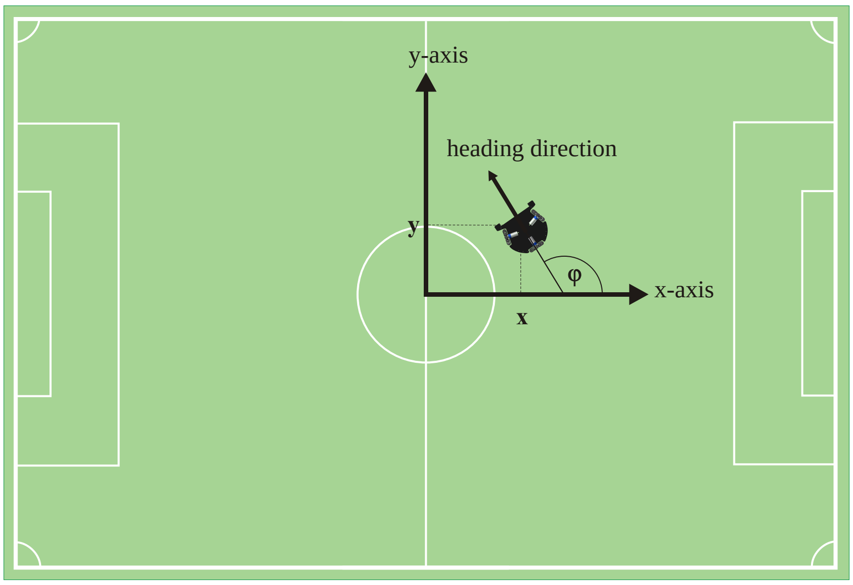


Figure 4.1: The robot's position is represented in a global coordinate system whose origin is at the center of the playing field. The cartesian x, y-position together with the orientation ϕ is referred to as the robot's *pose*. The orientation $\phi \in [0, ..., 2\pi[$ is the angle between the x-axis and the heading direction of the robot.

4.3 Coordinate Systems and Transformations

The vector $\breve{\mathbf{x}}$ is the robot's pose in the global coordinate system. A corresponding local coordinate system can be uniquely derived from this pose whose relationship to the robot's geometry is illustrated in figure 4.2. The local coordinate system of the robot corresponding to $\breve{\mathbf{x}} = (x\ y\ \phi)^T$ is given by the triple $(\mathbf{P}, \mathbf{u}, \mathbf{v})$ where $\mathbf{P} = (x\ y)^T$ is the Cartesian position in the global system, and $\mathbf{u}, \mathbf{v}$, the local x- and y-axis, are given by:

$$\mathbf{u} = \begin{pmatrix} \sin\phi \\ -\cos\phi \end{pmatrix} \tag{4.3}$$

$$\mathbf{v} = \begin{pmatrix} \cos\phi \\ sin\phi \end{pmatrix} \tag{4.4}$$

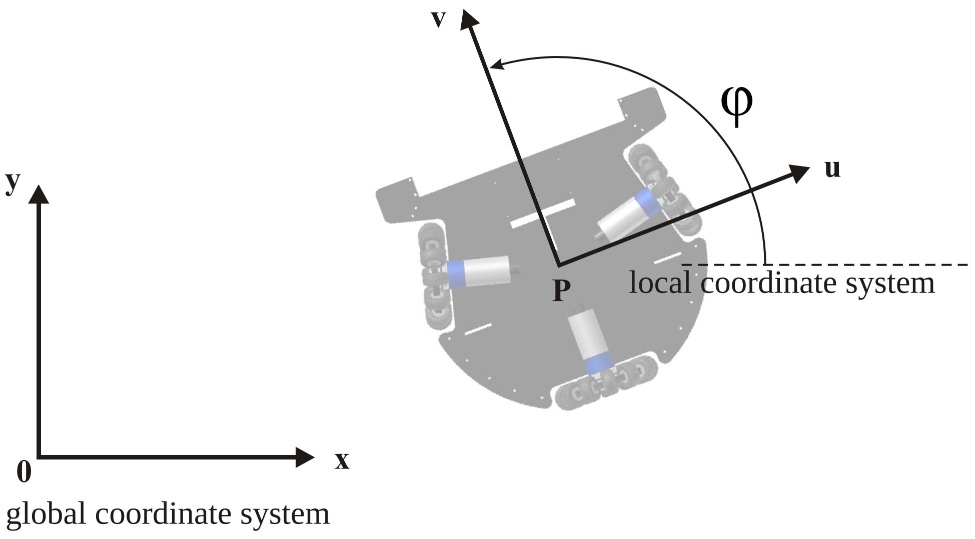


Figure 4.2: The x-axis of the local coordinate system of the robot is directed towards vector **u**, the y-axis towards vector **v**, which is the heading direction of the robot. **P** specifies the Cartesian position of the robot in the global system.

There are four types of entities which can be represented in either coordinate system, the local or the global: *Points*, *vectors*, *poses* and *movements*. Both, points and vectors are specified in the form $\mathbf{p} \in I\!R^2$. Poses and movements are three-dimensional vectors with the third component being an angle. A pose $\breve{\mathbf{p}}$ is used to specify a position together with an orientation and it uniquely defines a local coordinate system with the y-axis directed towards the specified orientation, given as a polar angle.

$$\breve{\mathbf{p}} = \begin{pmatrix} x \\ y \\ \phi \end{pmatrix} \tag{4.5}$$

A movement is the difference between two poses, typically between two poses of the same robot and we refer to a movement $\mathbf{\Delta m} \in I\!R^3$ in the form

$$\mathbf{\Delta m} = \begin{pmatrix} \Delta x \\ \Delta y \\ \Delta \phi \end{pmatrix}. \tag{4.6}$$

However, movements will also occur later in layer 3 for describing how a feature has to be translated and rotated in order to match a corresponding model feature. When the movement is considered as a change of the robot's pose, $\Delta\phi$ is the change of the robot's heading direction. We allow $\Delta\phi$ to have any value in $\mathbb{R}$, that is we do not delimit $\Delta\phi$ to $[-\pi, ...\pi[$.

If an entity is given in the local system, we will denote it with the superscript "r" for "robot". If an entity is specified in the global system, then no subscript is used. Next, we describe how to transform points, vectors, poses and movements from one system to the other. Using homogenous coordinates, transformations for points and vectors can be expressed by matrix multiplications. However, poses and movements which have an angle in the last component cannot be expressed easily by them. Moreover, the matrices would be sparse and it is more efficient to directly implement the transformations without the need for matrix multiplications. For these reasons, we have chosen not to use homogeneous coordinates. Instead, we use a set of transformation functions. In order to specify them correctly we first have to define $P := \mathbb{R} \times \mathbb{R} \times [0, ..., 2\pi[$, the set of robot poses. With $\breve{\mathbf{x}}$, the robot's pose, table 4.2 defines the transformations from the local to the global system.

$\xrightarrow{point}: P \times \mathbb{R}^2 \longrightarrow \mathbb{R}^2$	$\breve{\mathbf{x}} \xrightarrow{point} \mathbf{p}^r = \mathbf{p}$
$\xrightarrow{vector}: P \times \mathbb{R}^2 \longrightarrow \mathbb{R}^2$	$\breve{\mathbf{x}} \xrightarrow{vector} \mathbf{t}^r = \mathbf{t}$
$\xrightarrow{pose}: P \times \mathbb{R}^3 \longrightarrow \mathbb{R}^3$	$\breve{\mathbf{x}} \xrightarrow{pose} \breve{\mathbf{p}}^r = \breve{\mathbf{p}}$
$\xrightarrow{movement}: P \times \mathbb{R}^3 \longrightarrow \mathbb{R}^3$	$\breve{\mathbf{x}} \xrightarrow{movement} \mathbf{\Delta m}^r = \mathbf{\Delta m}$

Table 4.2: Transforming the point $\mathbf{p}^r$, the vector $\mathbf{t}^r$, the pose $\breve{\mathbf{p}}^r$ and the movement $\mathbf{\Delta m}^r$ from the local system at $\breve{\mathbf{x}}$ into the global coordinate system.

$\xleftarrow{point}: P \times \mathbb{R}^2 \longrightarrow \mathbb{R}^2$	$\breve{\mathbf{x}} \xleftarrow{point} \mathbf{p} = \mathbf{p}^r$
$\xleftarrow{vector}: P \times \mathbb{R}^2 \longrightarrow \mathbb{R}^2$	$\breve{\mathbf{x}} \xleftarrow{vector} \mathbf{t} = \mathbf{t}^r$
$\xleftarrow{pose}: P \times \mathbb{R}^3 \longrightarrow \mathbb{R}^3$	$\breve{\mathbf{x}} \xleftarrow{pose} \breve{\mathbf{m}} = \breve{\mathbf{m}}^r$
$\xleftarrow{movement}: P \times \mathbb{R}^3 \longrightarrow \mathbb{R}^3$	$\breve{\mathbf{x}} \xleftarrow{movement} \mathbf{\Delta m} = \mathbf{\Delta m}^r$

Table 4.3: Transforming the point $\mathbf{p}$, the vector $\mathbf{t}$, the pose $\breve{\mathbf{p}}$ and the movement $\mathbf{\Delta m}$ from the global system into the local system at $\breve{\mathbf{x}}$.

With $(\mathbf{P}, \mathbf{u}, \mathbf{v})$ being the local system corresponding to the pose $\breve{\mathbf{x}}$, $\mathbf{p}^r = (p_x^r \, p_y^r)^T$, the mappings of table 4.2 are given as follows:

$$\breve{\mathbf{x}} \xrightarrow{point} \mathbf{p}^r := \mathbf{P} + p^r_x \mathbf{u} + p^r_y \mathbf{v} \tag{4.7}$$

$$\breve{\mathbf{x}} \xrightarrow{vector} \mathbf{t}^r := t^r_x \mathbf{u} + t^r_y \mathbf{v} \tag{4.8}$$

$$\breve{\mathbf{x}} \xrightarrow{pose} \breve{\mathbf{p}}^r := \breve{\mathbf{x}} + \begin{pmatrix} \breve{p}^r_x u_x + \breve{p}^r_x v_x \\ \breve{p}^r_y u_y + \breve{p}^r_y v_y \\ \phi^r \end{pmatrix} \tag{4.9}$$

$$\breve{\mathbf{x}} \xrightarrow{movement} \mathbf{\Delta m}^r := \begin{pmatrix} \Delta m^r_x u_x + \Delta m^r_x v_x \\ \Delta m^r_y u_y + \Delta m^r_y v_y \\ \Delta \phi^r \end{pmatrix} \tag{4.10}$$

Correspondingly, we define transformations for mapping entities from the global to the local system (table 4.3). The mappings $\xrightarrow{point}, \xrightarrow{vector}, \xrightarrow{pose}$ and $\xrightarrow{movement}$ are bijective and the mappings $\xleftarrow{point}, \xleftarrow{vector}, \xleftarrow{pose}$ and $\xleftarrow{movement}$ are the respective inverse functions. They are defined as:[1]

$$\breve{\mathbf{x}} \xleftarrow{point} \mathbf{p} := \begin{pmatrix} (\mathbf{p} - \mathbf{P})^T \mathbf{u} \\ (\mathbf{p} - \mathbf{P})^T \mathbf{v} \end{pmatrix} \tag{4.11}$$

$$\breve{\mathbf{x}} \xleftarrow{vector} \mathbf{t} := \begin{pmatrix} \mathbf{t}^T \mathbf{u} \\ \mathbf{t}^T \mathbf{v} \end{pmatrix} \tag{4.12}$$

$$\breve{\mathbf{x}} \xleftarrow{pose} \breve{\mathbf{p}} := \breve{\mathbf{x}} - \begin{pmatrix} (\breve{p}_x \, \breve{p}_y) \mathbf{u} \\ (\breve{p}_x \, \breve{p}_y) \mathbf{v} \\ \phi \end{pmatrix} \tag{4.13}$$

$$\breve{\mathbf{x}} \xleftarrow{movement} \mathbf{\Delta m} := \begin{pmatrix} (\Delta m_x \, \Delta m_y) \mathbf{u} \\ (\Delta m_x \, \Delta m_y) \mathbf{v} \\ \Delta \phi \end{pmatrix} \tag{4.14}$$

[1]Note that $\binom{a_x}{a_y}^T \binom{b_x}{b_y} = (a_x \; a_y) \binom{b_x}{b_y} = a_x b_x + a_y b_y$

4.4 Relationship between Wheel Rotations and the Robot's Movement

Figure 4.3 depicts the local coordinate system of the robot and the geometric relationships that will be used in order to derive equations that relate the velocities of the individual wheels to the translational and rotational velocity of the robot. First of all, we consider

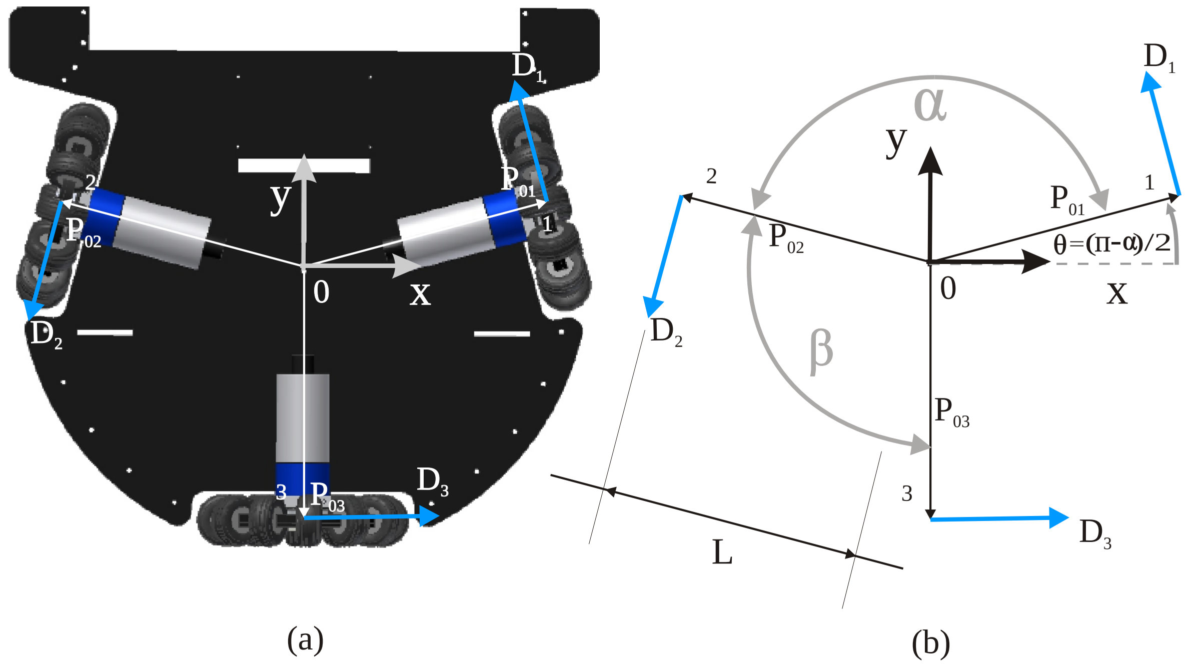


Figure 4.3: In (a) the robot is viewed from above and the symbols that are used in the text are illustrated. To avoid confusion, not all symbols are drawn in (a) but the geometry is repeated in (b) with additional symbols.

the contact points $\mathbf{P}_{0i}$, $i = 1, 2, 3$, of the wheels, specified in the local system of the robot. In order to easily calculate these points, we define

$$\mathbf{R}(\psi) := \begin{pmatrix} \cos\psi & -\sin\psi \\ \sin\psi & \cos\psi \end{pmatrix}, \tag{4.15}$$

to be the rotation matrix which rotates a vector about the angle ψ. Then the contact points are:

$$\mathbf{P}_{0i} = L\mathbf{R}(\gamma_i)\begin{pmatrix}1\\0\end{pmatrix} \tag{4.16}$$

$$\tag{4.17}$$

with $\theta = \frac{(\pi-\alpha)}{2}$ and the definitions

$$\gamma_1 := \theta \tag{4.18}$$

$$\gamma_2 := \theta + \alpha \tag{4.19}$$

$$\gamma_3 := \theta + \alpha + \beta. \tag{4.20}$$

Note, that the contact points are specified in the local coordinate system of the robot. Thus, $\mathbf{P}_{0i}$ also expresses the direction of the respective axis of the wheels. Consecutively, the normalized directional vectors of the wheels are:

$$\mathbf{D}_i \quad = \quad \mathbf{R}(\frac{\pi}{2} + \gamma_i)\begin{pmatrix}1\\0\end{pmatrix} \tag{4.21}$$

Within the global coordinate system the velocity vectors of the the contact points are

$$\mathbf{v}_i \quad = \quad \begin{pmatrix}\dot{x}\\\dot{y}\end{pmatrix} + \dot{\mathbf{R}}(\phi - \frac{\pi}{2})\mathbf{P}_{0i} \tag{4.22}$$

where $\binom{\dot{x}}{\dot{y}}$ is the translational velocity and ϕ is the heading direction of the robot. Note, that $\dot{x}$, $\dot{y}$ and ϕ are part of the system state. Then the individual wheel velocities towards the active directions $\mathbf{D}_i$ are

$$v_i = \mathbf{v}_i^T\mathbf{R}(\phi - \frac{\pi}{2})\mathbf{D}_i \tag{4.23}$$

Substituting equation (4.22) into equation (4.23) results in

$$v_i = \left(\begin{pmatrix}\dot{x}\\\dot{y}\end{pmatrix} + \dot{\mathbf{R}}(\phi - \frac{\pi}{2})\mathbf{P}^r_{0i}\right)^T \mathbf{R}(\phi - \frac{\pi}{2})\mathbf{D}_i = \tag{4.24}$$

$$\begin{pmatrix}\dot{x}\\\dot{y}\end{pmatrix}^T \mathbf{R}(\phi - \frac{\pi}{2})\mathbf{D_i} + \underbrace{\mathbf{P}_{0i}{}^T\dot{\mathbf{R}}(\phi - \frac{\pi}{2})^T\mathbf{R}(\phi - \frac{\pi}{2})\mathbf{D}_i}_{=:v_{rot}}$$

The second term

$$v_{rot} = \mathbf{P}_{0i}{}^T \underbrace{\dot{\mathbf{R}}(\phi - \frac{\pi}{2})^T \mathbf{R}(\phi - \frac{\pi}{2})}_{=:\mathbf{M}} \mathbf{D}_i \tag{4.25}$$

can be simplified. Defining $\tau := \phi - \frac{\pi}{2}$ matrix $\mathbf{M}$ evolves to

$$\begin{aligned}
\mathbf{M} = &\quad \dot{\mathbf{R}}(\phi - \tfrac{\pi}{2})^T \mathbf{R}(\phi - \tfrac{\pi}{2}) &= \\
= &\quad \dot{\phi} \begin{pmatrix} -\sin\tau & -\cos\tau \\ \cos\tau & -\sin\tau \end{pmatrix}^T \begin{pmatrix} \cos\tau & -\sin\tau \\ \sin\tau & \cos\tau \end{pmatrix} &= \\
= &\quad \dot{\phi} \begin{pmatrix} -\sin\tau & \cos\tau \\ -\cos\tau & -\sin\tau \end{pmatrix} \begin{pmatrix} \cos\tau & -\sin\tau \\ \sin\tau & \cos\tau \end{pmatrix} &= \\
= &\quad \dot{\phi} \begin{pmatrix} 0 & \cos^2\tau + \sin^2\tau \\ cos^2\tau - \sin^2\tau & 0 \end{pmatrix} &= \\
= &\quad \dot{\phi} \begin{pmatrix} 0 & 1 \\ -1 & 0 \end{pmatrix}.
\end{aligned} \tag{4.26}$$

Substituting this result back into equation 4.25 we have:

$$\begin{aligned}
v_{rot} = &\quad \dot{\phi}\mathbf{P}_{0i}{}^T \begin{pmatrix} 0 & 1 \\ -1 & 0 \end{pmatrix} \mathbf{D}_i &= \\
= &\quad \dot{\phi}L \left(\mathbf{R}(\gamma_i)\tbinom{1}{0}\right)^T \begin{pmatrix} 0 & 1 \\ -1 & 0 \end{pmatrix} \mathbf{R}(\tfrac{\pi}{2} + \gamma_i)\tbinom{1}{0} &= \\
= &\quad \dot{\phi}L \tbinom{\cos\gamma_i}{sin\gamma_i}^T \begin{pmatrix} 0 & 1 \\ -1 & 0 \end{pmatrix} \tbinom{\cos\frac{\pi}{2}+\gamma_i}{\sin\frac{\pi}{2}+\gamma_i} &= \\
= &\quad \dot{\phi}L \tbinom{\cos\gamma_i}{sin\gamma_i}^T \tbinom{\sin\frac{\pi}{2}+\gamma_i}{-\cos\frac{\pi}{2}+\gamma_i} = \dot{\phi}L(\cos\gamma_i \; sin\gamma_i)\tbinom{\cos\gamma_i}{\sin\gamma_i} &= \dot{\phi}L
\end{aligned} \tag{4.27}$$

Thus, equation 4.23 evolves to

$$\begin{aligned}
v_i = \begin{pmatrix}\dot{x}\\ \dot{y}\end{pmatrix}^T \mathbf{R}(\phi - \frac{\pi}{2})\mathbf{D_i} + L\dot{\phi} = \\
= \begin{pmatrix}\dot{x}\\ \dot{y}\end{pmatrix}^T \mathbf{R}(\phi - \frac{\pi}{2})\mathbf{R}(\frac{\pi}{2} + \gamma_i)\begin{pmatrix}1\\ 0\end{pmatrix} + L\dot{\phi} = \\
= \begin{pmatrix}\dot{x}\\ \dot{y}\end{pmatrix}^T \mathbf{R}(\phi + \gamma_i)\begin{pmatrix}1\\ 0\end{pmatrix} + L\dot{\phi} = \\
= \begin{pmatrix}\dot{x}\\ \dot{y}\end{pmatrix}^T \begin{pmatrix}\cos\phi + \gamma_i\\ \sin\phi + \gamma_i\end{pmatrix} + L\dot{\phi} = \\
= \cos(\phi + \gamma_i)\dot{x} + \sin(\phi + \gamma_i)\dot{y} + L\dot{\phi}
\end{aligned} \tag{4.28}$$

The drive velocities are thus non-linear functions of the translational velocity and the angular velocity of the robot.

$$v_i = \cos(\phi + \gamma_i)\dot{x} + \sin(\phi + \gamma_i)\dot{y} + L\dot{\phi} \tag{4.29}$$

Summarizing v_1, v_2 and v_3 in a vector, this relationship can be written in matrix form:

$$\begin{pmatrix}v_1\\ v_2\\ v_3\end{pmatrix} = \begin{pmatrix}\cos(\phi + \gamma_1) & \sin(\phi + \gamma_1) & L\\ \cos(\phi + \gamma_2) & \sin(\phi + \gamma_2) & L\\ \cos(\phi + \gamma_3) & \sin(\phi + \gamma_3) & L\end{pmatrix}\begin{pmatrix}\dot{x}\\ \dot{y}\\ \dot{\phi}\end{pmatrix} \tag{4.30}$$

Substituting γ_1, γ_2 and γ_3 yields

$$\begin{pmatrix}v_1\\ v_2\\ v_3\end{pmatrix} = \begin{pmatrix}\cos(\phi + \theta) & \sin(\phi + \theta) & L\\ \cos(\phi + \theta + \alpha) & \sin(\phi + \theta + \alpha) & L\\ \cos(\phi + \theta + \alpha + \beta) & \sin(\phi + \theta + \alpha + \beta) & L\end{pmatrix}\begin{pmatrix}\dot{x}\\ \dot{y}\\ \dot{\phi}\end{pmatrix}. \tag{4.31}$$

Since $\theta = \frac{\pi-\alpha}{2}$ we have

$$\begin{pmatrix} v_1 \\ v_2 \\ v_3 \end{pmatrix} = \begin{pmatrix} \cos(\phi + \frac{\pi-\alpha}{2}) & \sin(\phi + \frac{\pi-\alpha}{2}) & L \\ \cos(\phi^* + \frac{\pi-\alpha}{2} + \alpha) & \sin(\phi + \frac{\pi-\alpha}{2} + \alpha) & L \\ \cos(\phi^* + \frac{\pi-\alpha}{2} + \alpha + \beta) & \sin(\phi + \frac{\pi-\alpha}{2} + \alpha + \beta) & L \end{pmatrix} \begin{pmatrix} \dot{x} \\ \dot{y} \\ \dot{\phi} \end{pmatrix} = \tag{4.32}$$

$$= \underbrace{\begin{pmatrix} \cos(\phi + \frac{\pi-\alpha}{2}) & \sin(\phi + \frac{\pi-\alpha}{2}) & L \\ \cos(\phi + \frac{\pi+\alpha}{2}) & \sin(\phi + \frac{\pi+\alpha}{2}) & L \\ \cos(\phi + \frac{\pi+\alpha}{2} + \beta) & \sin(\phi + \frac{\pi+\alpha}{2} + \beta) & L \end{pmatrix}}_{=:\mathbf{W}(\phi)} \begin{pmatrix} \dot{x} \\ \dot{y} \\ \dot{\phi} \end{pmatrix} \tag{4.33}$$

Thus, we obtain the relationship

$$\begin{pmatrix} v_1 \\ v_2 \\ v_3 \end{pmatrix} = \mathbf{W}(\phi) \begin{pmatrix} \dot{x} \\ \dot{y} \\ \dot{\phi} \end{pmatrix} \tag{4.34}$$

and vice versa,

$$\begin{pmatrix} \dot{x} \\ \dot{y} \\ \dot{\phi} \end{pmatrix} = \mathbf{W}^{-1}(\phi) \begin{pmatrix} v_1 \\ v_2 \\ v_3 \end{pmatrix} . \tag{4.35}$$

Note, that v_1, v_2 and v_3 are absolute values that specify the velocities of the respective contact points in the direction of the corresponding wheel, specified in [cm/s]. Dividing these velocities by the effective wheel radius r_e, we obtain the angular velocities ω_1, ω_2 and ω_3 of the respective axes:

$$\omega_i = \frac{v_i}{r_e} \tag{4.36}$$

Solving this equation for v_i and substituting v_1, v_2 and v_3 in equation 4.34 we obtain

$$\begin{pmatrix} \omega_1 \\ \omega_2 \\ \omega_3 \end{pmatrix} = \frac{1}{r_e} \mathbf{W}(\phi) \begin{pmatrix} \dot{x} \\ \dot{y} \\ \dot{\phi} \end{pmatrix} . \tag{4.37}$$

When considering odometric information later, this relationship will play an important role (see page 96).

4.5 The Dynamic Model

The process model for a system describes how the system states change as a function of time and is usually written as a first-order non-linear vector differential equation or state model of the form

$$\dot{\mathbf{x}}(t) = \mathbf{f}(\mathbf{x}(t), \mathbf{u}(t), t) + \mathbf{q}(t) \tag{4.38}$$

where $\mathbf{x}(t) \in I\!R^n$ is a vector of the states of interest at time t, $\mathbf{u}(t) \in I\!R^r$ is a known control input, $\mathbf{f}(\cdot,\cdot,\cdot)$ is a model of the rate of change of system state as a function of time, and $\mathbf{q}(t)$ is a random vector describing both dynamic driving noise and uncertainties in the state model itself.

We represent the state $\mathbf{x}(t)$ at time t of the robot by its cartesian x, y-position (units in [m]), its orientation ϕ [rad], its translational velocities v_x, v_y [m/s] and its rotational velocity w [rad/s].

$$\mathbf{x}(t) = \begin{pmatrix} x(t) \\ y(t) \\ \phi(t) \\ v_x(t) \\ v_y(t) \\ \omega(t) \end{pmatrix} \tag{4.39}$$

Then the differential equations of the model are

$$\begin{aligned} \dot{x}(t) &= v_x(t) + n_x(t) \\ \dot{y}(t) &= v_y(t) + n_y(t) \\ \dot{\phi}(t) &= \omega(t) + n_\phi(t) \\ \dot{v_x}(t) &= a_x(t) + n_{v_x}(t) \\ \dot{v_y}(t) &= a_y(t) + n_{v_y}(t) \\ \dot{\omega} &= a_\omega(t) + n_\omega(t). \end{aligned} \tag{4.40}$$

Here, n_x, n_y, n_ϕ, n_{v_x}, n_{v_y} and n_ω are assumed to be zero-mean, temporally uncorrelated Gaussian process noise errors with variance σ_x^2, σ_y^2, σ_ϕ^2, $\sigma_{v_x}^2$, $\sigma_{v_y}^2$ and σ_ω^2, respectively.

The terms $a_x(t)$, $a_y(t)$ and $a_\omega(t)$ are the accelerations that result from the control $\mathbf{u}(t)$. They will be derived in the following.

Low-level motor control is done by a special electronic system which is embedded in the robot. Communication with the main computer, which processes the computer vision and behavior calculation, is established via a serial interface. The electronic subsystem gets a command for a desired translational and rotational velocity and automatically controls the power of the three motors in order to reach the desired values. The delay from sending the control input to the electronics until the control input has an effect on the motor speeds is approximately eight microseconds.

We will denote the control commands at time t by

$$\mathbf{u}(t) = \begin{pmatrix} u_{v_x}(t) \\ u_{v_y}(t) \\ u_{\omega}(t) \end{pmatrix} \tag{4.41}$$

where $u_{v_x}(t)$, $u_{v_y}(t)$ and $u_{\omega}(t)$ are normalized command-values (v/v_{max}) without physical units, lying within the interval $[-1.0, ..., 1.0]$. The values specify the desired translational and rotational velocities at time t. A value of 1.0 corresponds to the maximum reachable speed for the respective component. The values $u_{v_x}(t)$, $u_{v_y}(t)$ are specified from the perspective of the robot. The y-direction corresponds to the robot's heading and the x-direction corresponds to its right-direction. For instance, to let the robot move with full speed towards its heading direction, $\mathbf{u}(t)$ should be set to

$$\mathbf{u}(t) = \begin{pmatrix} 0 \\ 1 \\ 0 \end{pmatrix}. \tag{4.42}$$

To let the robot slowly rotate at a fixed position to the left, $\mathbf{u}(t)$ should be set to

$$\mathbf{u}(t) = \begin{pmatrix} 0 \\ 0 \\ 0.1 \end{pmatrix}. \tag{4.43}$$

The control electronics that receives the commands implements a PID-controller. Since the 90W motors with the 12:1 gear produce a high torque compared to the relatively low weight of the robot($\approx$ 10kg), we do not consider the precise dynamics of the controller, but assume a simplified closed feedback loop model of its behavior in the following: The torque $T_i(t)$ of the ith ($i = 1, 2, 3$) direct current (DC) motor that arises from the control input $\mathbf{u}(t)$ depends on the current speed of the motor shaft. A reasonably accurate model that captures this relationship is

$$T_i(t) = \bar{\alpha}U(t) - \bar{\beta}\omega_i \tag{4.44}$$

where $U(t)$ [V] is the voltage applied to the motor, and $\omega_i(t)$ [rad/s] is the angular velocity of the shaft of motor i. The constants $\bar{\alpha}$ [Nm/V] and $\bar{\beta}$ [Nm rad/s] characterize the motor. The salient feature of this model is that the amount of torque available for acceleration is a function of the speed of the motor. When neglecting wheel slippage, the force $f_i(t)$ generated by the respective motor driven wheel is simply

$$f_i(t) = \alpha U(t) - \beta v_i(t) \tag{4.45}$$

where $v_i(t)$ [m/s]is the ground-velocity of the wheel mounted at the shaft of motor i. The constants α [N/V] and β [kg/s][2] can readily be determined from $\bar{\alpha}$, $\bar{\beta}$ and the geometry of the robot. The translational acceleration of wheel i is

$$a_i(t) = \frac{f_i(t)}{m} \tag{4.46}$$

where m is the mass that has to be accelerated. Now, we use the matrix $\mathbf{W}(\phi)$ defined in equation 4.33 on page 82 to project the accelerations of the wheels to the resulting acceleration of the robot:

$$\begin{pmatrix} a_x(t) \\ a_y(t) \\ a_\omega(t) \end{pmatrix} := \mathbf{W}(\phi)^{-1} \begin{pmatrix} a_1(t) \\ a_2(t) \\ a_3(t) \end{pmatrix} \tag{4.47}$$

4.6 Using a Kalman Filter to Fuse the Three Layers

We use a Kalman-filtering approach to fuse the resulting positions of the different layers. For an introduction to the filter see for instance [58][11].

We begin with a very general and simplified description of the filter and successively remove the simplifications.

[2]Note, that $1N = 1\frac{\text{kg}m}{s^2}$

A discrete Kalman filter tries to estimate the state $\mathbf{x} \in I\!R^n$ of a discrete-time controlled process that is governed by the linear stochastic difference equation

$$\mathbf{x_{k+1}} = \mathbf{A}\mathbf{x}_k + \mathbf{B}\mathbf{u}_k + \mathbf{q}_k. \tag{4.48}$$

The state is observed by a measurement vector $\mathbf{z} \in I\!R^m$ that is linearly coupled to the system state:

$$\mathbf{z}_k = \mathbf{C}_k\mathbf{x}_k + \mathbf{r}_k \tag{4.49}$$

The $n \times n$ matrix $\mathbf{A}$ relates the state at time step k to the state at step $k+1$. The $n \times l$ matrix $\mathbf{B}$ relates the control input $\mathbf{u} \in I\!R^l$ to the state $\mathbf{x}$. The $m \times n$ matrix $\mathbf{C}$ in the measurement equation relates the state to the measurement $\mathbf{z}_k$.

The process noise is represented by the random variable $\mathbf{q}_k$ which is normal distributed with zero mean and covariance matrix $\mathbf{Q}$. Accordingly, $\mathbf{r_k}$ models the measurement noise with covariance matrix $\mathbf{R}$.

In our system we represent the state $\mathbf{x}$ of the robot by its Cartesian x, y-position, its orientation ϕ, its translational velocities v_x, v_y and its rotational velocity $\omega = \dot{\phi}$.

$$\mathbf{x} = \begin{pmatrix} x & y & \phi & v_x & v_y & \omega \end{pmatrix}^T \tag{4.50}$$

The measurement $\mathbf{z}$ consists of the measurements from all three layers of localization.

$$\mathbf{z} := \left(\ \mathbf{z}_{odo} \, \mathbf{z}_{rel\ vis} \, \mathbf{z}_{rec} \ \right)^T \tag{4.51}$$

Here $\mathbf{z}_{odo}$, $\mathbf{z}_{rel\ vis}$ and $\mathbf{z}_{rec}$ are the respective measurement vectors of the three layers. The syntax used in equation 4.51 should indicate that the compound measurement vector $\mathbf{z}$ consists of the components of the three vectors, rather than of the vectors themselves. The measurement vector $\mathbf{z}_{odo}$ of layer 1 consists of the angles $\Delta\alpha_1$, $\Delta\alpha_2$ and $\Delta\alpha_3$ specifying the wheel rotations between two successive time steps. These angles are directly measured by the tick counters of the motors, which provide 768 impulses per wheel revolution.

$$\mathbf{z}_{odo} = (\Delta\alpha_1 \, \Delta\alpha_2 \, \Delta\alpha_3)^T \tag{4.52}$$

The measurement vectors of layer 2 and 3 are defined in a way, reflecting the direct measurement of the robot's pose.

$$\mathbf{z}_{rel\,vis} = (x_2 \; y_2 \; \phi_2)^T \tag{4.53}$$

$$\mathbf{z}_{rec} = (x_3 \; y_3 \; \phi_3)^T \tag{4.54}$$

Although layer 2 supplies only a relative movement, the measurement of an absolute pose can be simulated by adding this relative information to the most recent estimated pose. Layer 3 directly measures the robot's pose by recognizing high-level features that allow the unique inference of the robot's pose.

The measurement vector of each layer has to be related to the system state $\mathbf{x}$. Typically, not all components of the system state are required to define the measurement models for the individual layers. For instance, the odometric measurement vector only depends on the last three components of the system state, the translational and rotational velocities of the robot. In contrast, the measurements of layer 2 and 3 depend only on the first three components of the system state, namely the robot's pose. For layer 1, the relationship between the system state and measurement vector is not linear. A detailed description will follow on page 96, where we begin to describe the layers separately. There, the Jacobian matrix $\mathbf{C}_{odo}$ will be derived, linearizing the measurement model around the current estimate of the robot's pose:

$$\mathbf{z}_{odo} = \mathbf{C}_{odo} \begin{pmatrix} \dot{x} \\ \dot{y} \\ \dot{\phi} \end{pmatrix} + \mathbf{R}_{odo} \tag{4.55}$$

Here $\mathbf{R}_{odo}$ is the covariance matrix of the odometric measurements. For layer 2 and 3 the measurement model is linear and it can be expressed by the following equations:

$$\mathbf{z}_{rel\,vis} = \mathbf{C}_{rel\,vis} \begin{pmatrix} x \\ y \\ \phi \end{pmatrix} + \mathbf{R}_{rel\,vis} \tag{4.56}$$

$$\mathbf{z}_{rec} = \mathbf{C}_{rec} \begin{pmatrix} x \\ y \\ \phi \end{pmatrix} + \mathbf{R}_{rec} \tag{4.57}$$

Here, both $\mathbf{C}_{rel\,vis}$ and $\mathbf{C}_{rec}$ are the identity, since the layers are designed to directly measure the robot's pose. $\mathbf{R}_{rel\,vis}$ and $\mathbf{R}_{rec}$ are the corresponding measurement covariance matrices. Summarizing the measurement equations of all three layers, we have

$$\begin{pmatrix} \Delta\alpha_1 \\ \Delta\alpha_2 \\ \Delta\alpha_3 \\ x_2 \\ y_2 \\ \phi_2 \\ x_3 \\ y_3 \\ \phi_3 \end{pmatrix}_k = \underbrace{\begin{pmatrix} 0 & 0 & 0 & & & \\ 0 & 0 & 0 & & \mathbf{C}_{odo} & \\ 0 & 0 & 0 & & & \\ & & & 0 & 0 & 0 \\ & \mathbf{C}_{rel\,vis} & & 0 & 0 & 0 \\ & & & 0 & 0 & 0 \\ & & & 0 & 0 & 0 \\ & \mathbf{C}_{rec} & & 0 & 0 & 0 \\ & & & 0 & 0 & 0 \end{pmatrix}}_{\mathbf{C}_k} \begin{pmatrix} x \\ y \\ \phi \\ v_x \\ v_y \\ \omega \end{pmatrix}_k + \mathbf{r}_k. \tag{4.58}$$

Assuming that the measurements of the individual layers are independent from one another, the compound measurement matrix $\mathbf{r}_k$ is

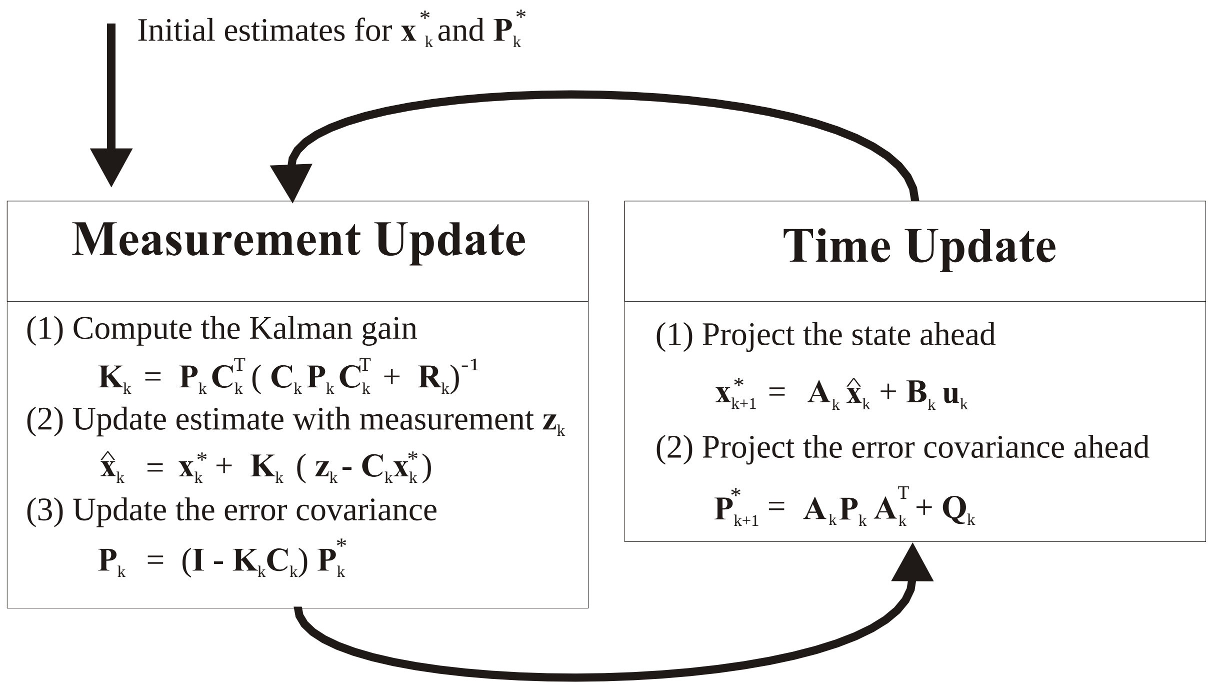
$$\mathbf{r}_k = \begin{pmatrix} \mathbf{R}_{odo} & 0 & 0 \\ 0 & \mathbf{R}_{rel\,vis} & 0 \\ 0 & 0 & \mathbf{R}_{rec} \end{pmatrix}. \tag{4.59}$$

Note, that the assumption of independent measurements is not completely valid, because the measurements of both layer 2 and layer 3 are based on the same field contours.

The measurement covariance matrices $\mathbf{R}_{odo}$, $\mathbf{R}_{rel\,vis}$ and $\mathbf{R}_{rec}$ are not static, but depend on the system state and other influences. Odometry, for instance, has larger variances when the robot is moving fast or when it is accelerating. When using features in the second layer to obtain a relative visual correction of the robot's pose, the variances depend i.e on the distance and on the orientation of the features with respect to the robot. However, the dynamic adaption of the measurement matrices is not dealt with in this thesis.

The Kalman filter iteratively processes two steps, the *measurement update* and the *time update*. The measurement update combines the last estimate of the system state with the current measurement and produces a new estimate of the state. The time update uses the dynamic model to predict the state for the next time step. The calculations that have to be carried out are illustrated in figure 4.4.

Here, it is assumed that the measurements of the three layers have the same time delay. This simplification will be removed later.



Initial estimates for $\mathbf{x}^*_k$ and $\mathbf{P}^*_k$
Measurement Update
(1) Compute the Kalman gain
$\mathbf{K}_k = \mathbf{P}_k \mathbf{C}_k^T (\mathbf{C}_k \mathbf{P}_k \mathbf{C}_k^T + \mathbf{R}_k)^{-1}$
(2) Update estimate with measurement $\mathbf{z}_k$
$\hat{\mathbf{x}}_k = \mathbf{x}^*_k + \mathbf{K}_k (\mathbf{z}_k - \mathbf{C}_k \mathbf{x}^*_k)$
(3) Update the error covariance
$\mathbf{P}_k = (\mathbf{I} - \mathbf{K}_k \mathbf{C}_k) \mathbf{P}^*_k$
Time Update
(1) Project the state ahead
$\mathbf{x}^*_{k+1} = \mathbf{A}_k \hat{\mathbf{x}}_k + \mathbf{B}_k \mathbf{u}_k$
(2) Project the error covariance ahead
$\mathbf{P}^*_{k+1} = \mathbf{A}_k \mathbf{P}_k \mathbf{A}_k^T + \mathbf{Q}_k$

Figure 4.4: The Kalman filter iteratively processes two steps, the *measurement update* and the *time update*. The measurement update combines the last estimate of the system state with the current measurement and produces a new estimate of the state. The time update uses the dynamic model to predict the state for the next time step.

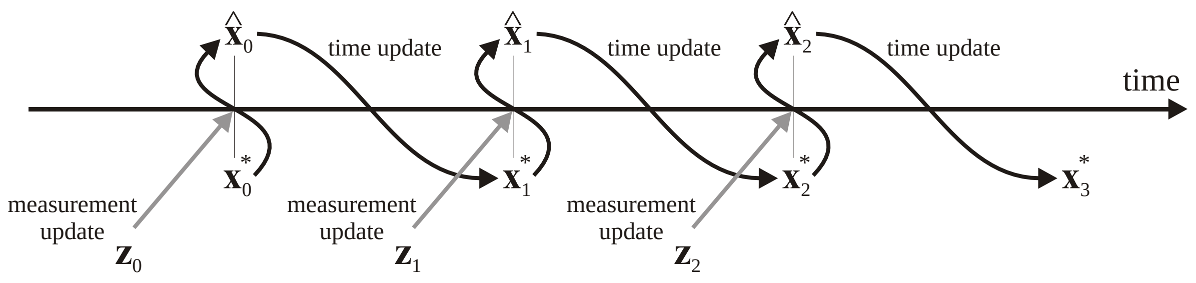


Figure 4.5: This figure shows the flow of computation while executing the filter steps of figure 4.4 over several time steps.

4.7 Fusing Delayed Measurements

In the above description of using a Kalman filter in order to fuse the measurements of the three layers a simplification was made. It was assumed that the measurements have the same time delays. Of course, this is not true. Odometric information typically enters the system 8 ms after the corresponding physical event (wheel rotation). In contrast, images have a typical delay of approximately 60 ms. Consider what happens when not taking account for this time shift. Assume that just the last incoming data is fused and that the robot is standing still on the playing field at the beginning. Assume further that the robot is detecting the field lines and tries to match the lines onto an internal model in order to correct for small positional errors (layer 2). Then, when the robot starts rotating, odometric information will reflect the rotation after 8 ms. However, at that time the images still reflect the state with the robot not moving because of the larger time delay. The images will not show a rotation before 60 ms after the robot started rotating. Hence, it is incorrect to fuse just the last incoming data. One has to take account for the correspondence of the different signals in time. The above consideration is illustrated in figure 4.6. For the sake of simplicity, the observed signal is considered to be continuous and fusion of only the first two layers (odometry and relative visual correction) are considered.

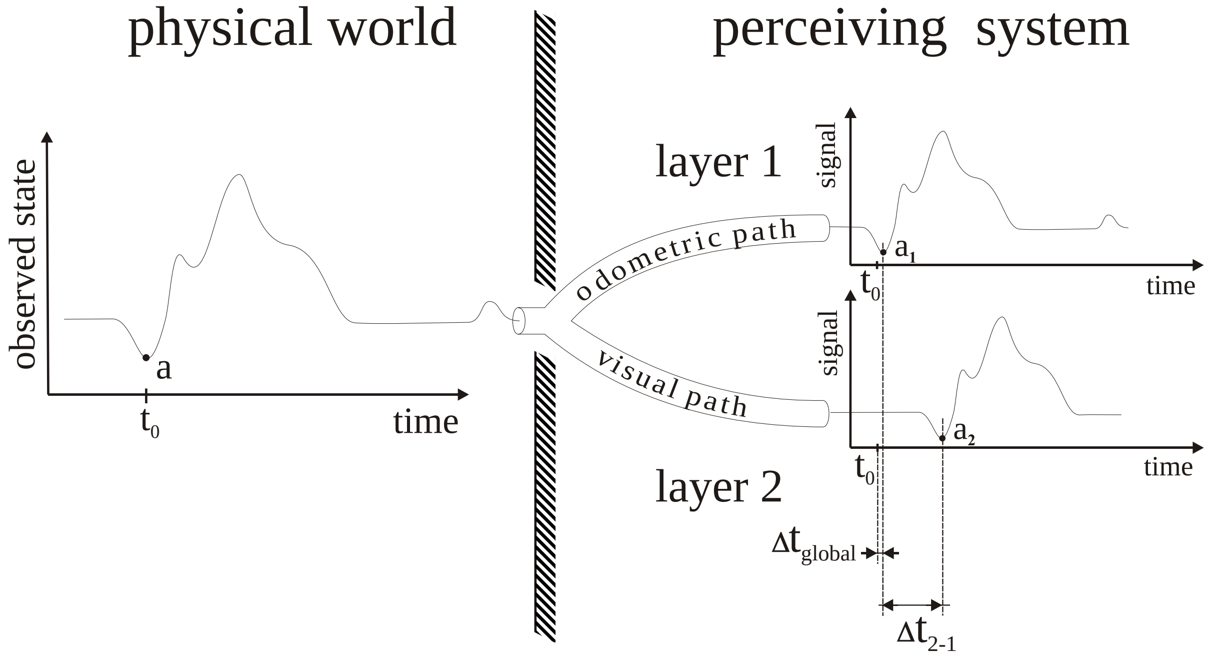


Figure 4.6: A physical state (i.e. the orientation of the robot) is observed by two different sensor systems, odometry (layer 1) and relative visual correction (layer 2). A physical event (t_0, a) is observed with different time delays. The corresponding event in layer 1, $(t_0 + \Delta t_{global})$, has a delay of Δt_{global}. In layer 2 the corresponding event is $(t_0 + \Delta t_{global} + \Delta t_{2-1})$, that is the delay is $\Delta t_{global} + \Delta t_{2-1}$. Here Δt_{global} is the delay of the most reactive sensor (layer 1) and Δt_{2-1} denotes the time shift between layer 1 and 2. Here, only two layers are depicted for the sake of simplicity, however the scheme extends to all three layers. The higher a layer the greater the delay. The time shift between two layers, i.e Δt_{2-1}, can be determined by the system by matching the signals to each other. However, it is not possible to determine Δt_{global} without another reference sensor. In principle, such a reference sensor is just another layer (layer 0) with a time delay Δt_{global} that is so small that it can be neglected. For fusing the signals of the different layers correctly, it is sufficient to determine the relative time shifts between the layers. However, when incorporating the motion model, Δt_{global} also needs to be known. In our case Δt_{global} is the odometric delay which is approximately 8 milliseconds.

4.7.1 Splitting the Kalman Cycle

When considering the most recent data entering the system through the different sensor layers, there exists odometric information for which no corresponding visual information is yet available. Accordingly, there are situations, where layer 3 has not yet detected a feature, but there is already relative visual information (layer 2) available. To keep the description easy, let us concentrate only on the first two layers. The scheme can then be adapted easily to all three layers. We split the Kalman cycle into two phases, phase 0 and phase 1. (Considering all three layers, the process would have been split into three phases). Each phase consists of a time and a measurement update. While the dynamic model is the same for both phases, each phase has its own measurement model. During phase 0, corresponding visual and odometric information is fused. During phase 1, the most recent odometric information is used, for which no corresponding visual information is yet available (see figure 4.7). The measurement model for phase 0 is:

$$\mathbf{z}_{phase\,0,k} = \begin{pmatrix} \Delta\alpha_1 \\ \Delta\alpha_2 \\ \Delta\alpha_3 \\ x_2 \\ y_2 \\ \phi_2 \end{pmatrix}_{phase\,0,k} = \begin{pmatrix} 0 & 0 & 0 & & & \\ 0 & 0 & 0 & & \mathbf{C}_{odo} & \\ 0 & 0 & 0 & & & \\ & & & 0 & 0 & 0 \\ & \mathbf{C}_{rel\;vis} & & 0 & 0 & 0 \\ & & & 0 & 0 & 0 \end{pmatrix} \begin{pmatrix} x \\ y \\ \phi \\ v_x \\ v_y \\ \omega \end{pmatrix}_k + \mathbf{r}_{phase\,0,k}, \tag{4.60}$$

and the measurement model for phase 1 is

$$\mathbf{z}_{phase1,k} = \begin{pmatrix} \Delta\alpha_1 \\ \Delta\alpha_2 \\ \Delta\alpha_3 \end{pmatrix}_{phase\,1,k} = \begin{pmatrix} 0 & 0 & 0 & & \\ 0 & 0 & 0 & \mathbf{C}_{odo} & \\ 0 & 0 & 0 & & \end{pmatrix} \begin{pmatrix} x \\ y \\ \phi \\ v_x \\ v_y \\ \omega \end{pmatrix}_{phase\,1,k} + \mathbf{r}_{phase\,1,k}. \tag{4.61}$$

Figure 4.8 shows the overall flow of computation.

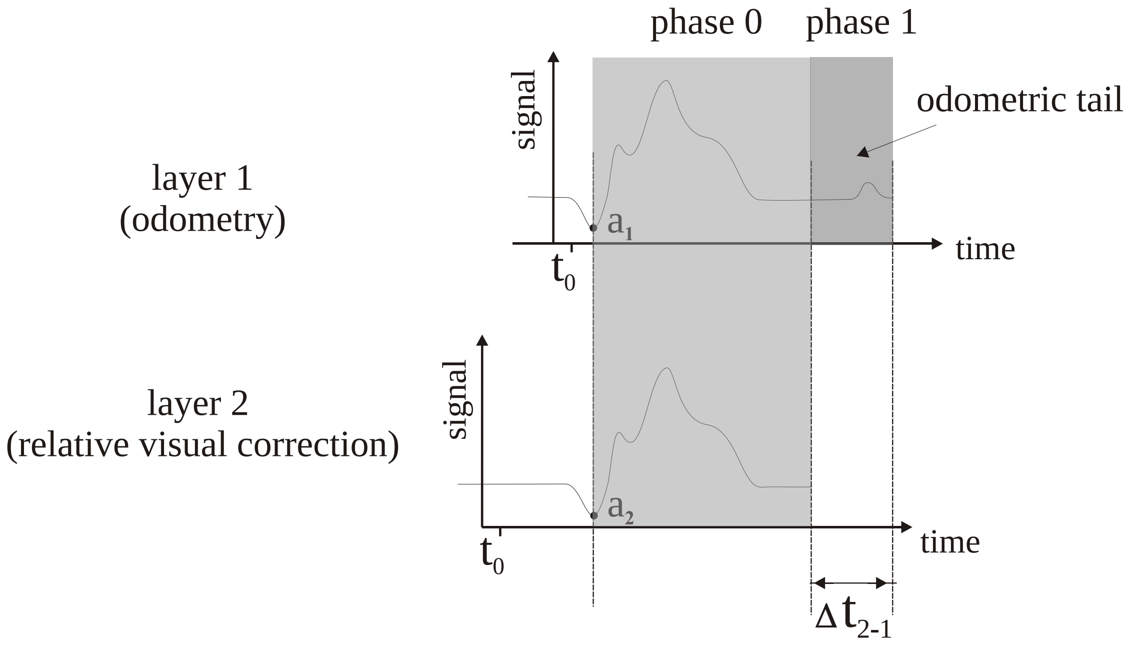


Figure 4.7: When considering the most recent information gathered by the different sensor systems, there exists odometric data for which no corresponding visual information is yet available. This is due to the fact that odometric information has a shorter delay (8 ms) compared to visual information (60 ms). Therefore, when only considering the first two layers, the fusion process is divided into two phases. During phase 0 corresponding odometric and visual information are fused, while during phase 1, the most recent odometric data (the *odometric tail*), for which there is not yet correlating visual data, is used to update the estimate of the system state of the robot. For both phases different measurement equations are used and each phase uses its own filter matrix $\mathbf{K}_i$, $i = 0, 1$.

4.7.2 Explicit Representation of Time

When dealing with measurement delays, one major problem has to be solved: The correspondence of different signals in time observing the same system state has to be resolved. Assume for a moment, that we have two discrete signals $..., a_{t-3}, a_{t-2}, a_{t-1}, a_t$ and $..., b_{t-3}, b_{t-2}, b_{t-1}, b_t$, with a_t and b_t being the signal values at time step t. Assume further, that the signals are sampled with the same frequency. Then, if we know the relative time shift Δt between the signals, with b being the signal with the higher measurement delay, then b_t corresponds to $a_{t-\Delta t}$. Under these simplified conditions, it would suffice, to store the incoming data in two separate queues and to use shifted indices to address corresponding data.

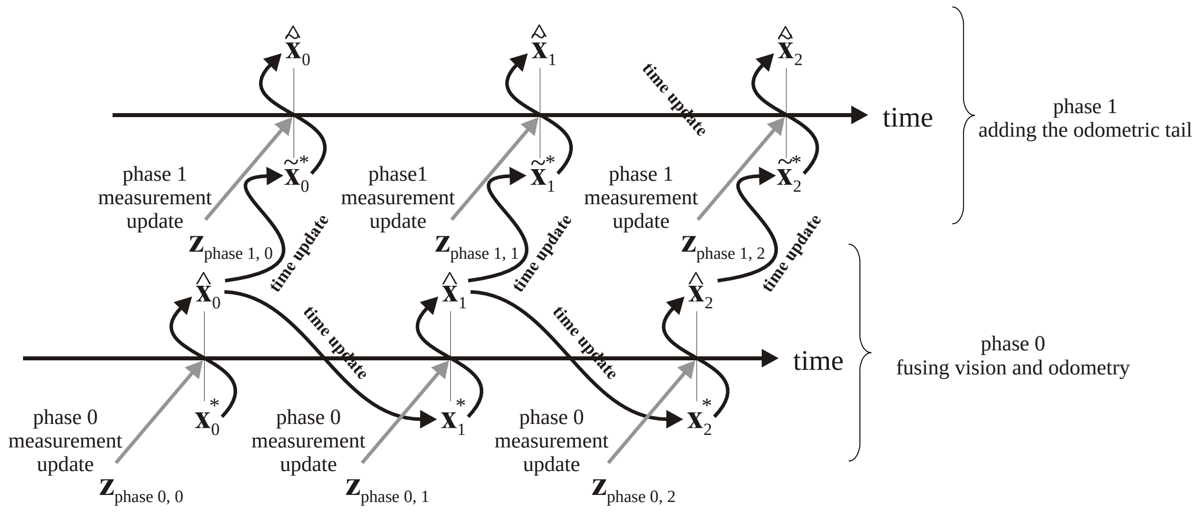


Figure 4.8: The bottom part of this figure corresponds to figure 4.5 where no measurement delays are considered. Beginning with the last estimate $\hat{\mathbf{x}}_k$ of the system state, a phase 0 time update propagates the motion model for the time interval in which both odometric and relative visual information is available (yielding $\mathbf{x}^*_k$), then a phase 0 measurement update is performed, which fuses the corresponding odometric and visual information (yielding $\hat{\mathbf{x}}^*_k$). A phase 1 time update is performed in parallel, propagating the motion model for the time interval in which only odometric information is available (yielding $\tilde{\mathbf{x}}^*_k$). Finally, a phase 1 measurement update is processed, which integrates the odometric tail and produces the final estimate $\hat{\hat{x}}_k$ of the system state. This system state is used to control trajectory execution (after having applied a further prediction step).

However, reality is more complicated. First of all, the different signals typically are not sampled with the same frequency. Layer 1, odometric information, provides data at a rate of 50 samples per second, while layer 2 and 3 yield only 10 measurements per second. Moreover, it typically happens that the sampling intervals within one signal vary. While this is not true for odometric information, whose sampling intervals are very precise, the time gaps between two successive image frames are not always the same. This can be due to the operating system, the video acquisition drivers, or the varying computational load while processing the images. In the extreme, the processing is so expensive, that a frame has to be dropped. Thus, using a simple shift of indices to take account for the measurement delays is not adequate.

Instead, time has to be represented explicitly. With recent processors (i.e. the Pentium III and higher), a so-called *performance counter* is available, a 64-bit counter that is incremented with each clock cycle. With such a counter, time can be measured with a precision below a microsecond (10^{-6} s). When the signals of the three different layers enter the system, each data item is provided with such a 64 bit time stamp, and the data is stored in a queue. Each layer has its own queue, which we refer to as a *data channel* in the following. The data channels of the different layers have a length of approximately 100 items. The odometric data channel has 500 items due to the higher sampling frequency. That is, each channel stores the last 500 or 100 incoming data items, together with their time stamps. Note, that the time stamps reflect the time when the signal enters the system, and not the time of the corresponding physical event.

Now assume that two successive images are grabbed with time stamps t_1 and t_2. Now, we know the relative time shift Δt between odometric and visual information, which is 52 milliseconds in our system. We will describe later, how this time shift can be determined automatically by the system. In order to find the odometric data which correspond to the time interval of the two images, we consider the interval $t_1 - \Delta t$ and $t_2 - \Delta t$. We use each shifted time stamp to find the index i of the data item in the odometric data channel that most closely matches by its time stamp. This search can be efficiently implemented (in $O(\log n)$) by binary search, since the data in each channel are ordered by the time stamps. In this way, we find the corresponding odometric data and we fuse the data as described in the previous sections.

To find the relative time shift Δt between odometry and vision, one can simply observe a landmark, start the robot rotating and to stop the rotation after a small time period. Then both odometry and vision will reflect the change of the angle and the time shift can be recovered by determining how the data in the different channels have to be shifted in order to produce the highest correlation. In our system we use the ball lying still on the field as a landmark. It typically suffices to determine the time shift once, however, one could consider techniques which are able to adapt to a changing time shift online.

4.8 Layer 1: Odometric Information

The robot's motors are equipped with tick counters generating 64 impulses per revolution. Since each motor has a 12:1 gear, $n_{ticks} = 768$ impulses are generated per wheel revolution. The counters are equipped with a sign specifying the direction of rotation. The ticks of each wheel are summed up by the control electronics and periodically (50 times per second) transmitted to the computer. Let $\lambda_{k,i}$, $i = 1, 2, 3$, denote these accumulated ticks of motor i at time t_k. By taking the difference of two successive accumulated tick values, the relative ticks between two time steps can be computed. That is, we have the relative tick vector:

$$\boldsymbol{\Delta}\lambda_k = \begin{pmatrix} \Delta\lambda_{k,1} \\ \Delta\lambda_{k,2} \\ \Delta\lambda_{k,3} \end{pmatrix} = \begin{pmatrix} \lambda_{k,1} \\ \lambda_{k,2} \\ \lambda_{k,3} \end{pmatrix} - \begin{pmatrix} \lambda_{k-1,1} \\ \lambda_{k-1,2} \\ \lambda_{k-1,3} \end{pmatrix} \tag{4.62}$$

We can compute the corresponding wheel rotations from the relative tick vector $\boldsymbol{\Delta}\lambda_k$ corresponding to the time interval $\Delta t = k - t_{k-1}$:

$$\begin{pmatrix} \Delta\alpha_{k,1} \\ \Delta\alpha_{k,2} \\ \Delta\alpha_{k,3} \end{pmatrix} = \frac{2\pi}{n_{ticks}} \begin{pmatrix} \Delta\lambda_{k,1} \\ \Delta\lambda_{k,2} \\ \Delta\lambda_{k,3} \end{pmatrix} \tag{4.63}$$

This vector $(\Delta\alpha_{k,1}, \Delta\alpha_{k,2}, \Delta\alpha_{k,3})^T$ is the measurement vector of layer 1 at time step k. In section 4.4 on page 82 equation 4.37 was derived, describing the relationship between the robot's translational and rotational velocities, and the angular velocities of the wheels $(\omega_1, \omega_2, \omega_3)^T$. This relationship is

$$\begin{pmatrix} \omega_1(t) \\ \omega_2(t) \\ \omega_3(t) \end{pmatrix} = \frac{1}{r_e}\mathbf{W}(\phi(t)) \begin{pmatrix} \dot{x}(t) \\ \dot{y}(t) \\ \dot{\phi}(t) \end{pmatrix}. \tag{4.64}$$

When integrating the angular wheel velocities over the time interval $[t_{k-1}, ..., t_k]$, we obtain the relationship

$$\begin{pmatrix} \Delta\alpha_{k,1} \\ \Delta\alpha_{k,2} \\ \Delta\alpha_{k,3} \end{pmatrix} = \mathbf{g}_{odo}(\mathbf{x}(t_{k-1}), ..., \mathbf{x}(k), r_e) := \int_{t=t_{k-1}}^{k} \frac{1}{r_e}\mathbf{W}(\phi(t)) \begin{pmatrix} \dot{x}(t) \\ \dot{y}(t) \\ \dot{\phi}(t) \end{pmatrix} dt \tag{4.65}$$

Linearizing $\mathbf{g}_{odo}$ around the current estimate $\mathbf{x}^*$ of the system state, we obtain the approximation

$$\begin{pmatrix} \Delta\alpha_1 \\ \Delta\alpha_2 \\ \Delta\alpha_3 \end{pmatrix} \approx \frac{1}{r_e}\mathbf{W}(\phi^*) \begin{pmatrix} \dot{x} \\ \dot{y} \\ \dot{\phi} \end{pmatrix} \Delta t. \tag{4.66}$$

Thus, the matrix $\mathbf{C}_{odo}$, which was already used in equation 4.60 is given by

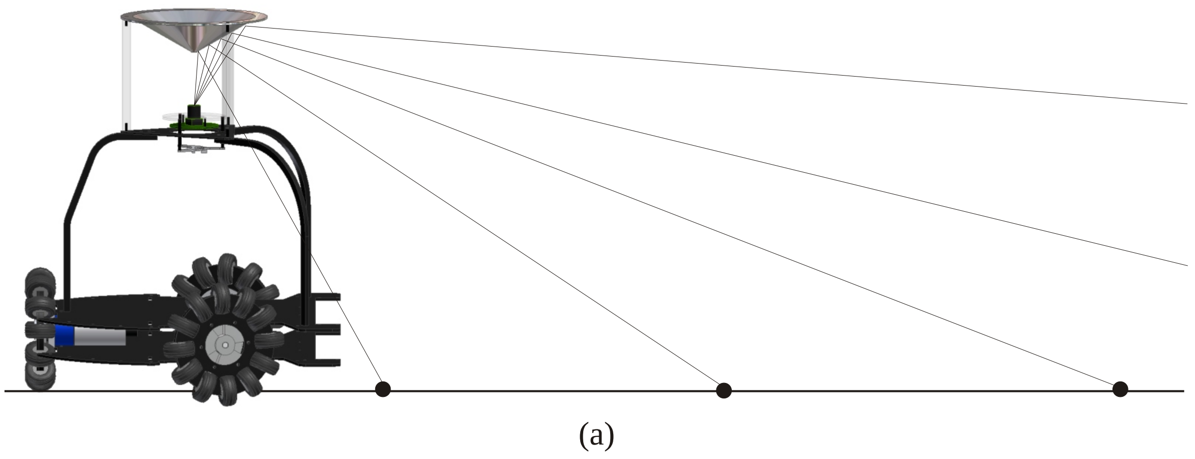
$$\mathbf{C}_{odo} = \frac{1}{r_e}\mathbf{W}(\phi^*)\Delta t. \tag{4.67}$$

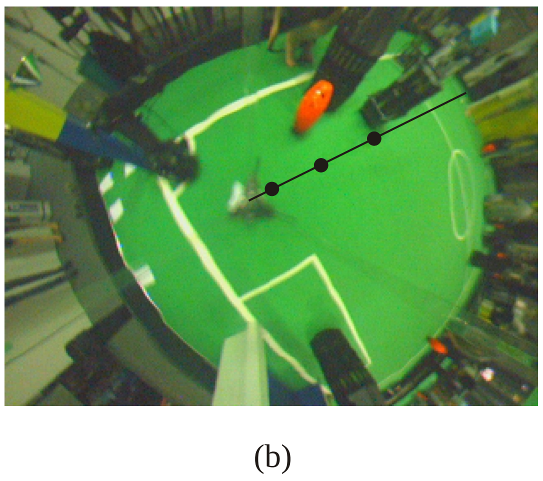
4.9 The Observation Model

In layer 1 we solely considered odometric information. The next two layers will use visual information. In order to describe these layers we have to know the relationship between the robot's system state and the location where things can be expected in the image. To establish this relationship is the purpose of the following section. After having described the relationship we will continue with the description of layer 2 and 3.

4.9.1 The Omni-Directional Vision System

Standard cameras have a limited visual field of about 30 - 60 degrees. To enhance the field of view, the idea of using a mirror in combination with a conventional imaging system to obtain *omni-directional images* has been proposed by Rees in a U.S. patent in 1970 [67]. The general approach of combining a mirror with a lens is referred to as *catadioptric* image formation. The term is composed of *catoptrics*, the science of reflecting, and *dioptrics*, the science of refracting surfaces. Different types of setups have been examined. Yagi and Kawato used a conic mirror [87]. Hong and others used a spherical mirror [43]. Yamazawa and others used a hyperboloidal mirror [88] and finally, Nayar and Baker [63] developed a setup using a parabola mirror and a telecentric lens. The latter two approaches generate images taken by a single center of projection, which is particularly important when aiming to transform the omni-directional images into perspective images.





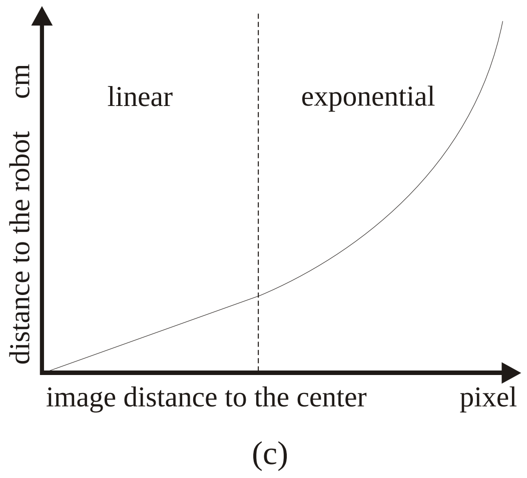


Figure 4.9: The optical mapping. In figure (b) three points are marked by small black dots. Each of these points corresponds to a ray, which is reflected by the mirror (figure (a)) and intersects with the ground plane, yielding corresponding points on the floor. Thus, each distance from the mirror's center to a pixel in the image corresponds to a real-world distance from the robot. This relationship is described by the distance function (figure (c)).

4.9.2 The Distance Function

For our soccer robots we have designed a specific mirror. Here, the design principle was not the goal to have a single center of projection, but a desired *distance function*, the function which relates distances in the image to distances in the world. If a convex, rotationally symmetric mirror is placed directly above the camera, then the center of the mirror is mapped to the center of the image. A pixel that is at a distance d to the center in the image corresponds to a ray that has an angle α to the viewing direction of the camera. The farther a pixel is away from the center, the greater is the angle of the corresponding ray. For analysis, it is easier to reverse the direction of the light, thinking of a ray as coming out of the camera and tracing the ray back to its origin. Adapting this notion, the rays leave the camera and are reflected by the mirror. After reflection, rays corresponding to outer pixels have a lower inclination towards the ground plane and the distance from the robot where they intersect with the ground plane increases (see figure 4.9a). The *distance function* describes this relationship. It maps a pixel distance to a world distance. The pixel distance is measured from the location of the pixel to the location of the mirror's center in the image (figure 4.9b). We designed our mirrors to yield a distance function which consists of a linear and an exponential portion as illustrated in figure 4.9c.

4.9.3 Predicting the Location of Objects in the Image

Given the pose of the robot on the playing field, we can predict the location of objects in the image. Before we describe the projection of arbitrary 3D points we first consider the special case that the point is on the two-dimensional playing field. Then we will reduce the general 3D case to the two-dimensional mapping.

4.9.4 Transformation of Two-Dimensional Points on the Field

Consider a point on the field, for instance a point $\mathbf{a}$, being part of the field lines as illustrated in figure 4.10. In the following we will describe how this point is mapped to the image. The mapping takes place in three stages, transforming the point into different two-dimensional cartesian coordinate systems.

The origin of the global world coordinate system is at the center of the playing field as illustrated in figure 4.10. All existing objects are initially modelled in this system, including the field lines. The robot's local coordinate system is given by its reduced system state $\breve{\mathbf{x}}$. Then the point $\mathbf{a}$ in the global system is transformed into the local

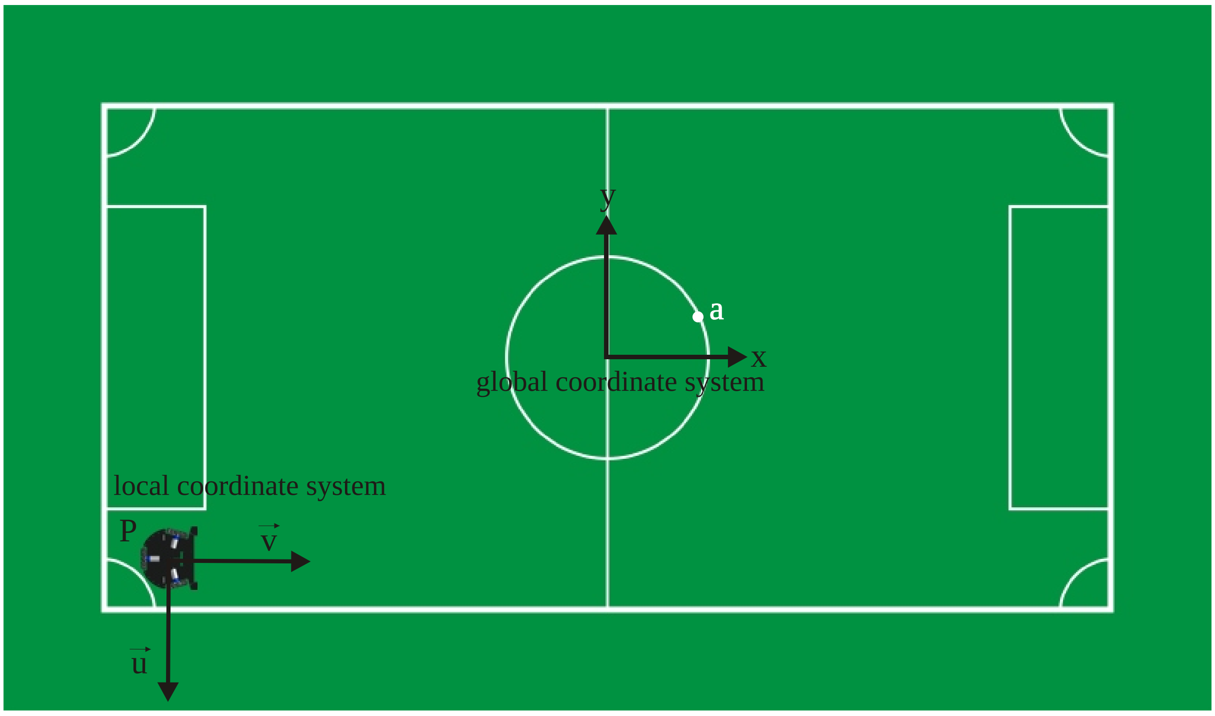


Figure 4.10: The world coordinate systems. The global system is located in the center of the playing field. The local system is located below the center of the robot's camera and mirror. Its y-axis is directed towards the robot's heading direction, its x-axis to the right.

system by the following equation[3]:

$$\mathbf{a^r} = \breve{\mathbf{x}} \xrightarrow{point} \mathbf{a} \tag{4.68}$$

The catadioptric mapping preserves angles between directions seen from the robot's position. Consider two different points in the local world coordinate system. These points form an angle α with the origin of the system, the robot's position. When projecting these points to the image, the points form the same angle with respect to the center of the mirror in the image. The reason is that the whole projection process of a light ray entering the CCD array of the camera takes place in a plane orthogonal to the ground.

Essentially, the image directly corresponds to the local world coordinate system after a pointwise application of the distance function. Here, applying the distance function means to measure the length of the point-vector $\mathbf{a}^r$, to transform that length by the distance

[3]The transformation $\xrightarrow{point}$ is defined on page 76

function and to create a new point-vector $\mathbf{a}^*$ pointing to the same direction but scaled to the transformed length:

$$\mathbf{a}^* = \frac{\mathbf{a}^r}{||\mathbf{a}^r||} d(||\mathbf{a}^r||) \tag{4.69}$$

Here $d(\cdot)$ denotes the distance function. However, the image appears rotated, because the camera has been mounted on the robot in a way that the diagonal of the CCD array is directed towards the robot's heading direction. The reason is that the diagonal direction offers a greater range of sight and we wanted the robot's heading direction to benefit from this advantage.

Therefore, after having applied the distance function to the point $\mathbf{a}^r$ the point is represented in a coordinate system which appears to be rotated in the image. We refer to this system as the local image coordinate system and its origin $\mathbf{C}$ is located at the center of the mirror in the image. In the image, the y-axis (the vector $\mathbf{v}^*$) of this system points in the direction that corresponds to the heading direction of the robot (see figure 4.11). Finally, after projecting the point out of the local image coordinate system, the point is represented in the global image coordinate system which is located at the top left of the image, with the coordinates specifying the pixel position.

$$\mathbf{a}^{\sim} = \mathbf{C} + a_x^* \mathbf{u}^* + a_y^* \mathbf{v}^* \tag{4.70}$$

4.9.5 Transformation of Arbitrary 3D Points

In order to transform arbitrary 3D points, we observe that a 3D point B is mapped to the same image pixel as the ground point a which is on the same ray being reflected into the camera. To calculate the corresponding point b on the floor we just have to determine the ray and intersect it with the ground plane. However, to determine the ray we would have to calculate the point R where the ray is reflected on the mirror. Although the precise calculation is possible, involving the shape of the mirror and the location of the focal point of the camera, it can be simplified by choosing an approximate reference point R which is located at the center of the mirror according to figure 4.12. For catadioptric systems with a single center of projection the calculation is precise if R is placed at the center of projection, for other setups it is just an approximation. However, especially for

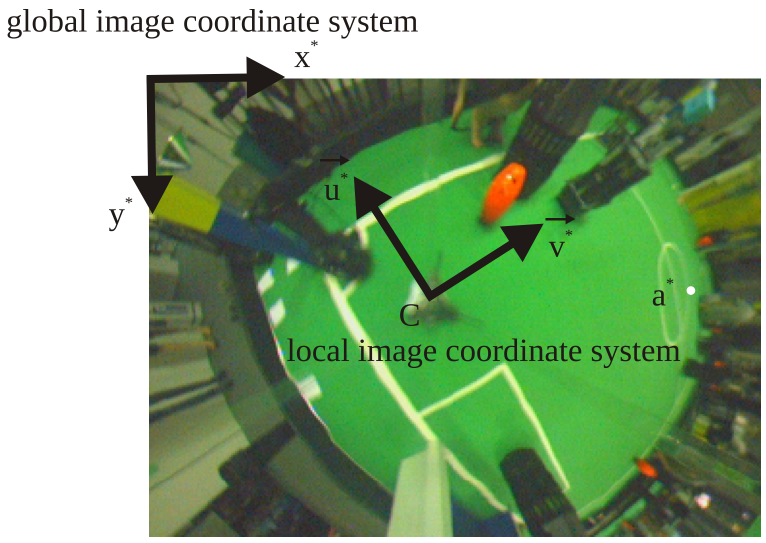


Figure 4.11: The image coordinate systems. The origin of the global system is located at the top left corner of the image. The local system is located at the center of the mirror in the image. Its y-axis (vector $\mathbf{v}^*$) is directed towards the robot's heading direction.

distant objects the error is small.

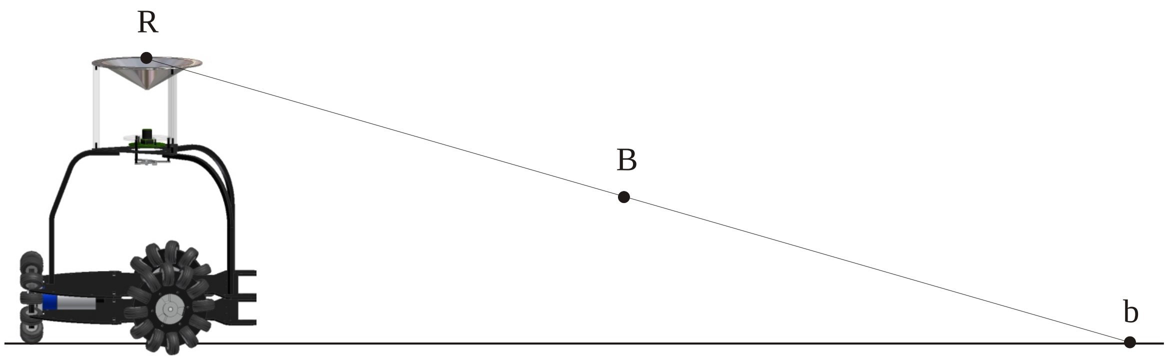


Figure 4.12: In order to project the 3D point B to the image we can calculate a corresponding point b on the ground and apply the two-dimensional mapping.

4.10 Transforming the Contours into World Space

Independent of the method that is used to localize the robot, the lines, as extracted from the images, are distorted by the mirror. We account for this distortion by transforming each point from image space to world space. This step makes use of the distance function as described on page 98. Figure 4.13 shows the distorted line contours as they are extracted by the region tracking algorithm. Given the robot's pose $\breve{\mathbf{x}}$, the line can be transformed

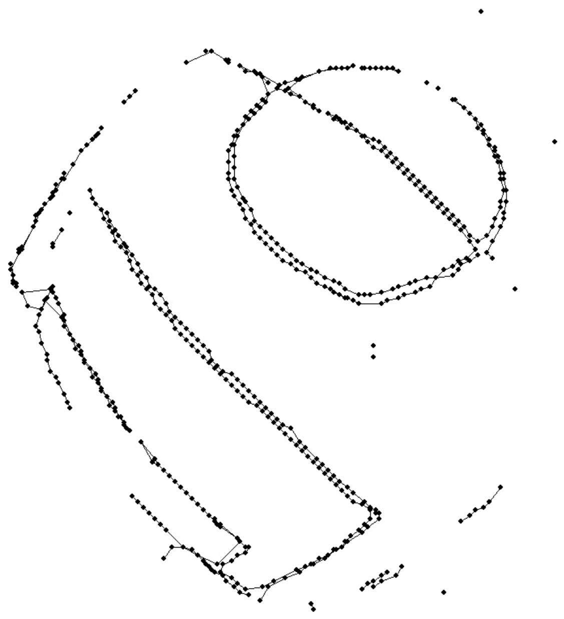


Figure 4.13: The distorted line contours as extracted by the region tracking algorithm.

out of the corresponding local coordinate system. If the robot's pose is incorrect, the lines will not fit to a model, as illustrated in figure 4.14. In contrast, when the robot's pose is correct, the lines closely match to the model (see figure 4.15). In order to verify whether the lines fit to the model or not, we have to establish a distance function. Before we set up this function, we first describe how the field lines are represented in the model.

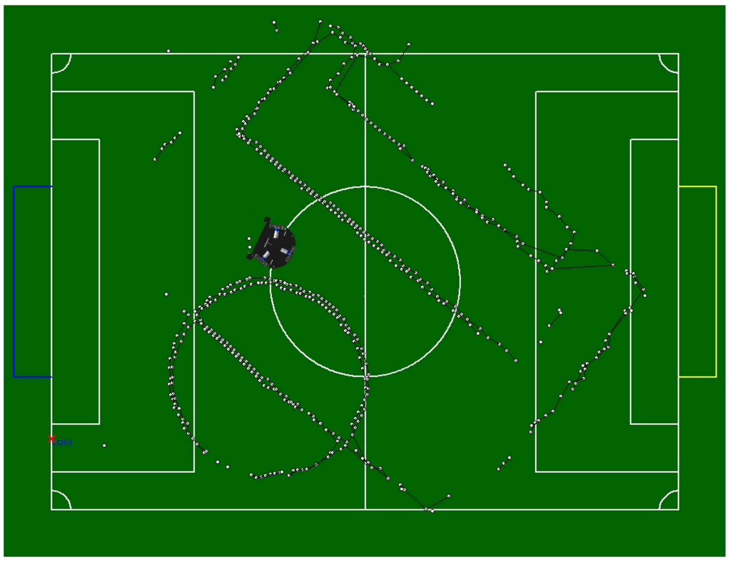


Figure 4.14: The distortion has been removed by applying the distance function and the contours have been transformed out of the local coordinate system. In this example, the estimate of the robot's pose is incorrect and as a result the line contours do not fit to the model.

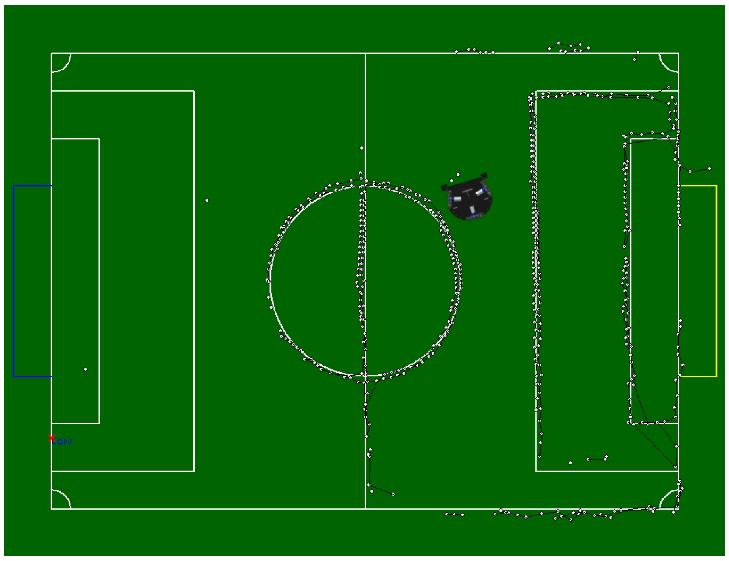


Figure 4.15: When the estimate of the robot's pose is correct, the image lines fit to a model of the field lines.

4.11 Modelling the Field Lines

In order to define a matching distance, we have to define a model of the lines. The field lines consist of curved and straight line segments (see figure 4.16).

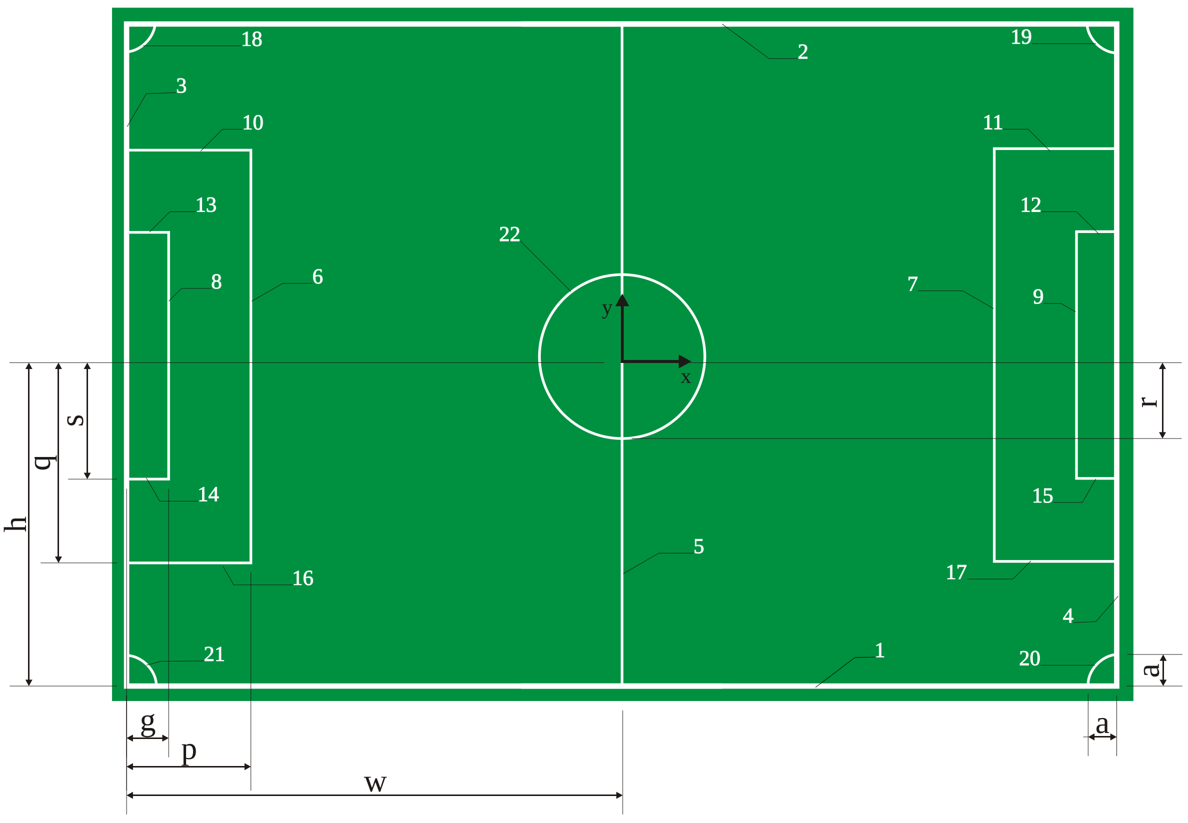


Figure 4.16: The model of the field lines consists of 17 straight line segments and five circular arcs, one of which is the center circle.

Curved lines can be found in form of the center circle and the quater circles at the corners. We represent straight line segments by the triple

$$(\mathbf{A}, \mathbf{v}, l), \tag{4.71}$$

where $\mathbf{A}$ is the two-dimensional start point of the line segment, $\mathbf{v}$ is the normalized direction of the line segment and l represents its length. The circular elements are defined by

$$(\mathbf{M}, r, \phi_0, \phi_1), \tag{4.72}$$

where $\mathbf{M}$ is the center point, r is the radius and ϕ_0, ϕ_1 define the enclosing angles of the respective circular arc. In this way, the center circle can be represented by setting $\phi_0 = 0, \phi_1 = 2\pi$ and the quater circles can be defined by setting the difference of the angles to $\frac{\pi}{2}$.

Then the whole model consists of 17 straight line segments and five circular arcs which are listed in table 4.4.

Nr.	line segment $(\mathbf{A}, \mathbf{v}, l)$	Nr.	line segment $(\mathbf{A}, \mathbf{v}, l)$
1	$(\binom{-w}{-h}, \binom{1}{0}, 2w)$	12	$(\binom{w}{s}, \binom{-1}{0}, g)$
2	$(\binom{w}{h}, \binom{-1}{0}, 2w)$	13	$(\binom{-w+g}{s}, \binom{-1}{0}, g)$
3	$(\binom{-w}{h}, \binom{0}{-1}, 2h)$	14	$(\binom{-w}{-s}, \binom{1}{0}, g)$
4	$(\binom{w}{-h}, \binom{0}{1}, 2h)$	15	$(\binom{w-g}{-s}, \binom{1}{0}, g)$
5	$(\binom{0}{-h}, \binom{0}{1}, 2h)$	16	$(\binom{-w}{-q}, \binom{1}{0}, p)$
6	$(\binom{-w+p}{-q}, \binom{0}{1}, 2q)$	17	$(\binom{w-p}{-q}, \binom{1}{0}, p)$
7	$(\binom{w-p}{q}, \binom{0}{-1}, 2q)$		circular arc $(\mathbf{M}, r, \phi_0, \phi_1)$
8	$(\binom{w-g}{-s}, \binom{0}{1}, 2s)$	18	$(\binom{-w}{h}, a, \frac{3}{2}\pi, 2\pi)$
9	$(\binom{-w+g}{s}, \binom{0}{-1}, 2s)$	19	$(\binom{w}{h}, a, \frac{2}{2}\pi, \frac{3}{2}\pi)$
10	$(\binom{-w+p}{q}, \binom{-1}{0}, p)$	20	$(\binom{w}{-h}, a, \frac{1}{2}\pi, \pi)$
11	$(\binom{w}{q}, \binom{-1}{0}, p)$	21	$(\binom{-w}{-h}, a, 0, \frac{1}{2}\pi)$
		22	$(\binom{0}{0}, r, 0, 2\pi)$

Table 4.4: The elements of the line model. For the world championships in Lisboa 2004 the constants are: $w = 600$ cm, $h = 300$ cm, $p = s = 150$ cm, $g = 50$ cm, $q = 250$ cm, $r = 100$ cm and $a = 40$ cm.

4.12 Layer 2: Relative Visual Localization

Relative localization assumes that an initial estimate of the robot's pose is available and keeps track of the pose while the robot moves.

Assume that initially the robot has been determined to be at pose $\mathbf{A}$. Then, within a small time interval the robot moves to pose $\mathbf{P}$. However, we do not know this new position. Instead, we have an estimate $\mathbf{P_{odo}}$ of the position due to odometric information. In the long term, odometric information accumulates to unbounded positional error. Thus, we have to correct $\mathbf{P_{odo}}$ with visual information.

In the following, we consider two different methods that achieve this correction. The first method is new, and we refer to it as the "MATRIX"-method. It takes a cloud of points representing the field lines, transforms the points into the global coordinate system corresponding to $\mathbf{P_{odo}}$ and calculates a small correcting movement $\Delta\mathbf{m}$ which makes the points fit better to the model of the lines. Here, the idea behind the method is that the points and the lines attract each other, resulting in an overall force and momentum which iteratively makes the cloud adjust towards the line model. A force field is pre-calculated in order to achieve real-time computation.

The second method is an already known-method, which is referred to as the *system dynamics approach.* It is due to Dickmanns [26] and Wünsche [85, 86] and was already described in chapter 2. Although the method is more efficient when considered isolated, we have not adapted the method in our overall approach, since the third layer for localization, which performs the feature recognition, requires the same input as is used for the MATRIX method. However, we sketch how to adapt the system dynamics approach, since it is elegant, extremely efficient and a prospective base for further research.

4.12.1 MATRIX: A Force Field Pattern Approach

Matching Distance Based on Closest Point Relationships

Having a hypothesis for the robot's pose on the playing field, it is possible to transform the perceived line contours into the global coordinate system. In order to evaluate the quality of a hypothesis, we want to define a matching distance function between the perceived lines and a model of the lines. Defining this function is difficult, because the function not only needs to be appropriate, but also calculable in real-time.

An intuitive way to define a matching distance function is to determine for each point of the perceived line contours the closest point of the model and to calculate the average of the corresponding distances. In order to be more robust against outliers, we sum up a constant cost $c_{outlier}$ for distances which exceed a threshold $t_{outlier}$ (in our implementation $c_{outlier} = t_{outlier} = 50$ cm). With $\mathbf{A}$ being the set of line contour points given in the local coordinate system of the robot and the robot considered at pose $\breve{\mathbf{x}}$, the distance function is defined as:

$$d(\breve{x}, \mathbf{A}) := \frac{1}{n}((\sum_{p^r \in A_{valid}} |\breve{\mathbf{x}} \xrightarrow{point} p^r - p^*|) + |A_{outlier}| c_{outlier}). \tag{4.73}$$

Here $n := |\mathbf{A}|$ denotes the total number of points and $\mathbf{p}^*$ denotes the closest model point of perceived line point $\mathbf{p}^r \in \mathbf{A}$ subject to the estimated pose $\breve{\mathbf{x}}$ of the robot. The sets A_{valid} and $A_{outlier}$ divide $\mathbf{A}$ into valid points and outliers. A similar definition of a matching distance function can be found in [55], where a distance function between two laser range scans has been defined.

The computationally most expensive part in 4.73 is the determination of the point correspondence. Given a point $\mathbf{p} := \breve{\mathbf{x}} \xrightarrow{point} p^r$ we have to iterate through all elements (straight lines and circular arcs) of the line model, calculate the respective closest point, and finally take the point $\mathbf{p}^*$ with the overall smallest distance. For a straight line segment $(\mathbf{A}, \mathbf{v}, l)$ the respective closest point $\mathbf{p}^*$ can be calculated as

$$s := (\mathbf{p} - \mathbf{A})\mathbf{v} \tag{4.74}$$

$$t := \begin{cases} 0 & if \quad s < 0 \\ l & if \quad s > l \\ s & if \quad 0 <= s <= l \end{cases} \tag{4.75}$$

$$\mathbf{p}^* = \mathbf{A} + t\mathbf{v}. \tag{4.76}$$

For a circular arc $(\mathbf{M}, r, \phi_0, \phi_1)$, the calculation is

$$\mathbf{v} := (\mathbf{p} - \mathbf{M}) \tag{4.77}$$

$$\mathbf{p}^* = \begin{cases} \mathbf{M} + r\frac{\mathbf{v}}{|\mathbf{v}|} \text{if the polar angle of } \mathbf{v} \text{ is enclosed by } \phi_0 \text{ and } \phi_1 \\ \text{the closest endpoint of the arc otherwise} \end{cases} \tag{4.78}$$

These calculations are illustrated in figure 4.17.

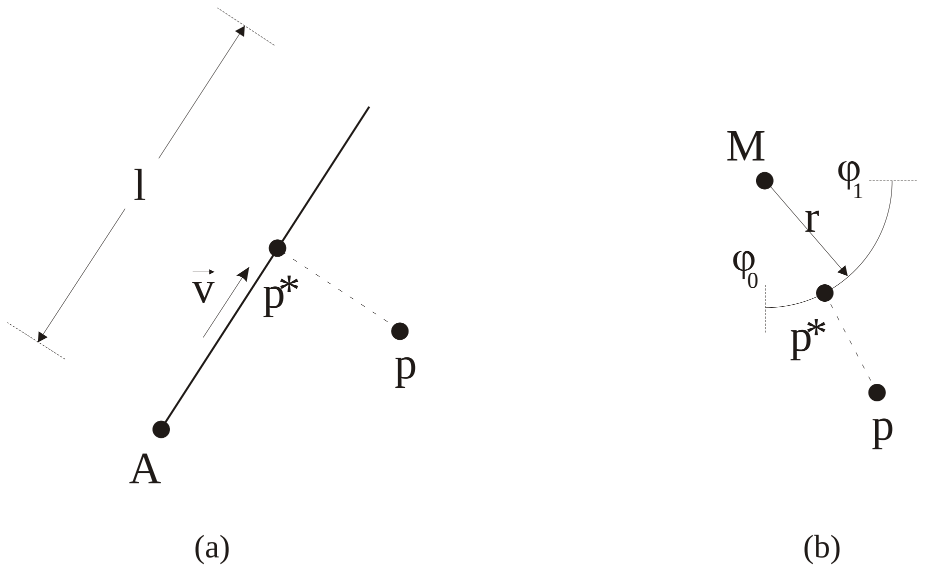


Figure 4.17: Calculating the closest point $\mathbf{p}^*$ of a point $\mathbf{p}$ for (a) a straight line segment $(\mathbf{A}, \mathbf{v}, l)$ and (b) a circular arc $(\mathbf{M}, r, \phi_0, \phi_1)$

The Quality Matrix

On average the perceived line contours consist of approximately 400 points and since the line model has 22 elements, about 8800 correspondence evaluations have to be carried out if we want to examine a hypothetical pose $\breve{\mathbf{x}}$. Since the line model is always the same, we can optimize the calculation by precomputing the correspondence relationships. We discretize the area of the playing field into small cells (the *MATRIX*) and precompute the distance to the closest point for the center of each cell. When evaluating the matching distance function $d(\breve{x}, \mathbf{A})$, we let each point $\mathbf{p} := \breve{\mathbf{x}} \xrightarrow{point} p^r$ index its corresponding cell and lookup the distance $\delta_{p^r} := |\breve{\mathbf{x}} \xrightarrow{point} p^r - p^*|$. When pre-computing the grid, we verify whether or not the calculated distances exceed the outlier threshold t and store $c_{outlier}$ to the respective cells in these cases. Thus, the distance function evolves to

$$d(\breve{\mathbf{x}}, \mathbf{A}) := \frac{1}{n} \sum_{p^r \in A} \delta_{p^r}. \tag{4.79}$$

This pre-computed *quality matrix* is illustrated in figure 4.18.

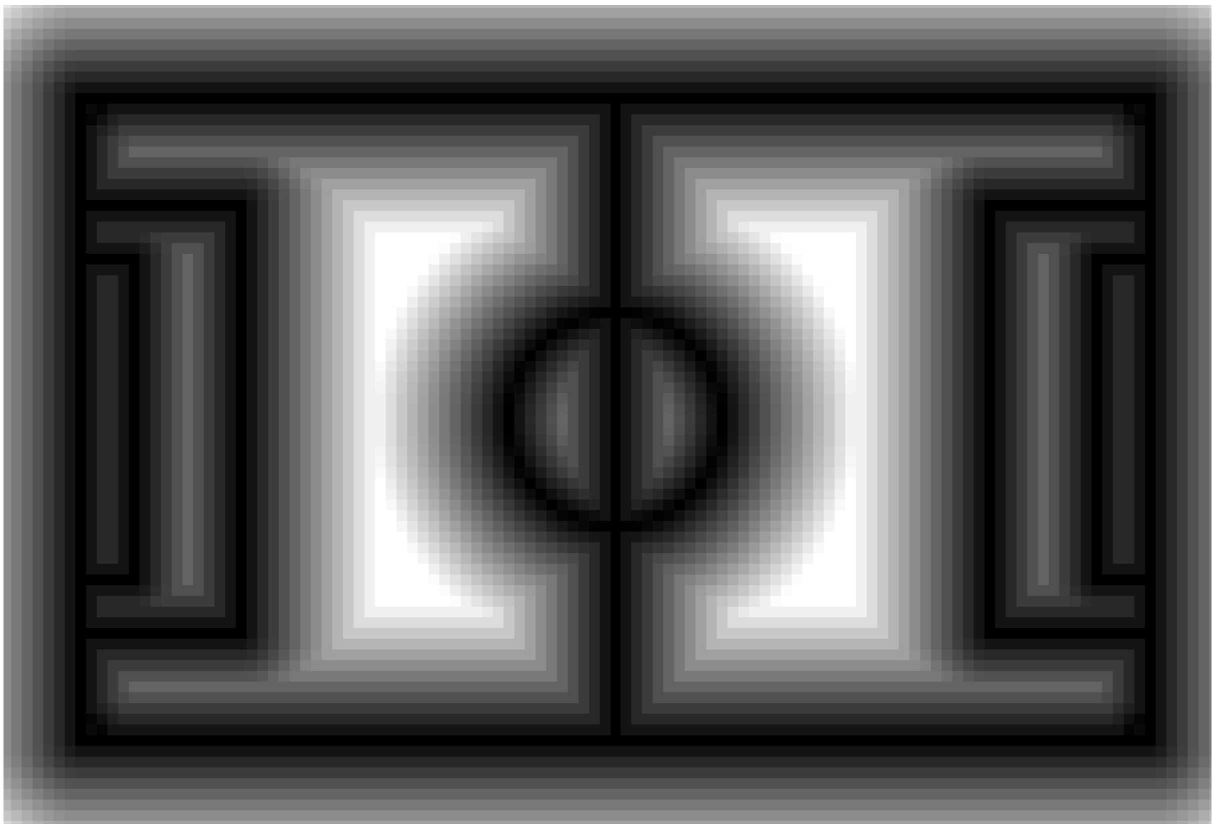


Figure 4.18: Each cell in the *quality matrix* stores the distance to the closest line point. High intensities correspond to large distances. The quality matrix is used to efficiently calculate the distance function $d(\breve{\mathbf{x}}, \mathbf{A})$, expressing how good the point set $\mathbf{A}$ fits to the field lines subject to the robot's pose $\mathbf{x}$.

Relative Correction by Particles

By means of the distance function $d(\breve{\mathbf{x}}, \mathbf{A})$ we can efficiently verify the hypothesis for the robot's pose $\breve{\mathbf{x}}$. In particular, when we have an initial estimate $\mathbf{P_{odo}}$, we can distribute a small number of particles (20 in our experiments) in the neighborhood of $\mathbf{P_{odo}}$. The distribution of the particles should account for the system state of the robot, including its velocities. That is, the variance of the particles should be highest in the direction of highest uncertainty. This can be achieved by sampling from a Gaussian distribution with the covariance matrix $\mathbf{P}$, the error covariance matrix of the system state $\mathbf{x}$. An example with the robot moving forward is illustrated in figure 4.19. Then the distance function $d(\breve{\mathbf{x}}_i, \mathbf{A})$ is evaluated for each sample $\breve{\mathbf{x}}_i$ and the best sample $\breve{\mathbf{x}}_{i_{best}}$ is chosen as the correct

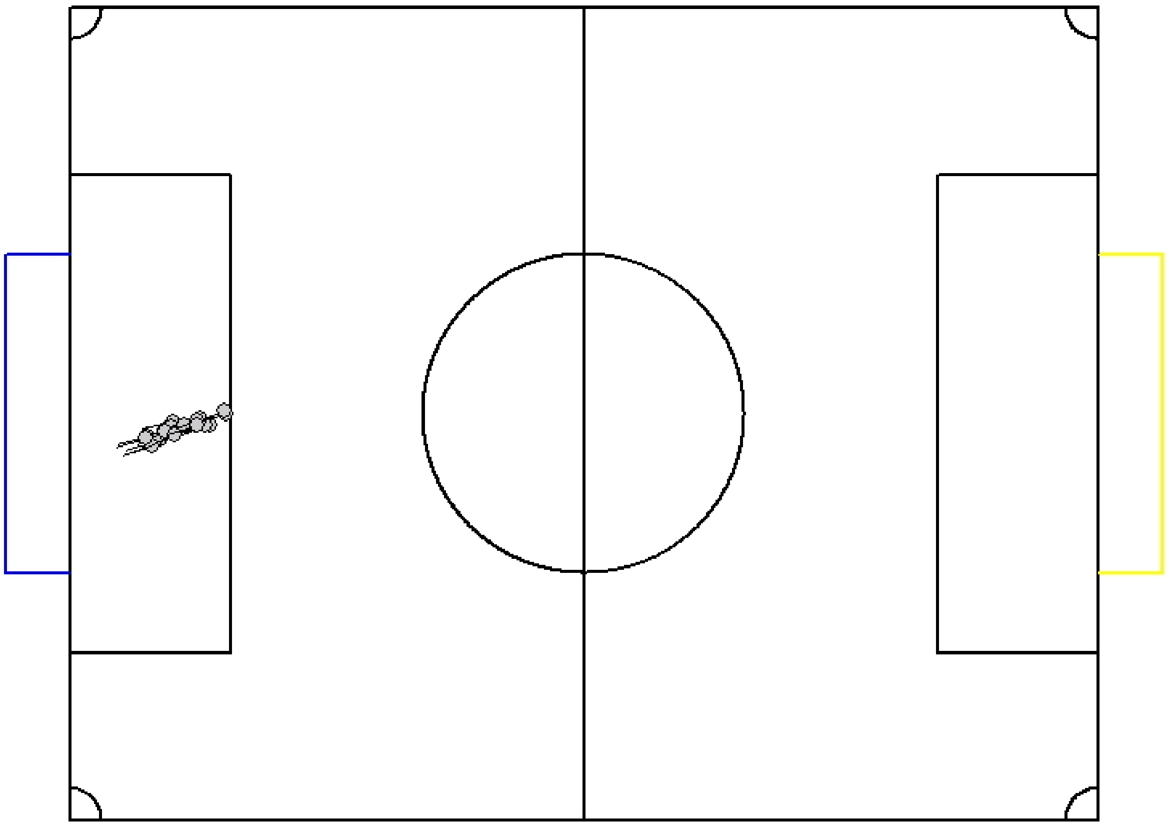


Figure 4.19: During relative localization 20 particles are distributed according to the robot's motion state around the estimated position of the robot.

position. That is

$$\mathbf{z}_{relative\,vision} := \breve{\mathbf{x}}_{i_{best}}. \tag{4.80}$$

Relative Correction by a Force Field

Instead of using samples, one can also simulate an attractive force and a momentum between the line model and the point set $\mathbf{A}$. This results in a gradient descent algorithm similar to the well-known *Iterative Closest Point* (ICP) algorithm [9], starting at $\mathbf{P_{odo}}$ and iteratively adjusting the position until the line contours converge towards a best fit to the model. Instead of calculating the forces online, we precalculate a force field in order to achieve real-time computation. Given a point $\mathbf{p}$ in the global coordinate system, we define the force vector function $\mathbf{f}(\mathbf{p})$ to be a weighted sum of the vectors $\mathbf{v_k}$ reaching from $\mathbf{p}$ to the closest point $\mathbf{p}_k^*$ on element k of the line model (straight line segments and circular arcs):

$$\mathbf{f}(\mathbf{p}) := \sum_{k=1}^{22} w_k \frac{\mathbf{v}_k}{||\mathbf{v}_k||} \min(||\mathbf{v}_k||, f_{max}), \tag{4.81}$$

Here, we delimit the force contributions to $f_{max} = 150$, with

$$\mathbf{v}_k := \mathbf{p}_k^* - \mathbf{p}. \tag{4.82}$$

The weights w_k exponentially decrease with an increasing distance of the corresponding points $\mathbf{p}_k^*$:

$$w_k := \tau e^{-\eta ||\mathbf{v}_k||}. \tag{4.83}$$

Since we do not use physical units, the constants and units in which lengths are expressed depend on each other. Here the constants are specified for lengths expressed in centimeters. The constant η defines the decline of the force field ($\eta = 0.1$ in our implementation) and τ is a normalizing constant which is

$$\tau = \sum_{i=1}^{22} e^{-\eta ||\mathbf{v}_i||}. \tag{4.84}$$

Hence, the weights w_k, $k = 1, 2, ..., 22$ sum up to one. Given an initial pose $\breve{\mathbf{x}}_k$ and a point set $\mathbf{A} = \{\mathbf{p}_0^r, \mathbf{p}_1^r, ..., \mathbf{p}_n^r\}$ representing points on the field lines, which are specified in the local coordinate system of the robot, we calculate a translational force and a momentum, which in turn result in a relative translation and rotation producing an improved estimate $\breve{\mathbf{x}}_{k+1}$ of the robot's pose. The overall force vector $\mathbf{F}(\breve{\mathbf{x}}_k, \mathbf{A})$ acting on the point cloud $\mathbf{A}$ and subject to an initial estimate of the robot's pose $\breve{\mathbf{x}}_k$ is computed by

$$\mathbf{F}(\breve{\mathbf{x}}_k, \mathbf{A}) = \frac{1}{n} \sum_{i=1}^{n} \mathbf{f}(\breve{\mathbf{x}}_k \xrightarrow{point} \mathbf{p}_i^r) \tag{4.85}$$

Pre-computing the forces for all possible positions on a discretization of the playing field yields a force-field, which is illustrated in figure 4.20.

The momentum $m(\mathbf{p}, \mathbf{g})$ induced by the force $\mathbf{f}(\mathbf{p})$ on a single point $\mathbf{p}$ onto the center of gravity $\mathbf{g}$ of point set $\mathbf{A}$ is calculated by

$$m(\mathbf{p}, \mathbf{g}) = \mathbf{f}(\mathbf{p}) \mathbf{R}(\pi/2)(\mathbf{g} - \mathbf{p}). \tag{4.86}$$

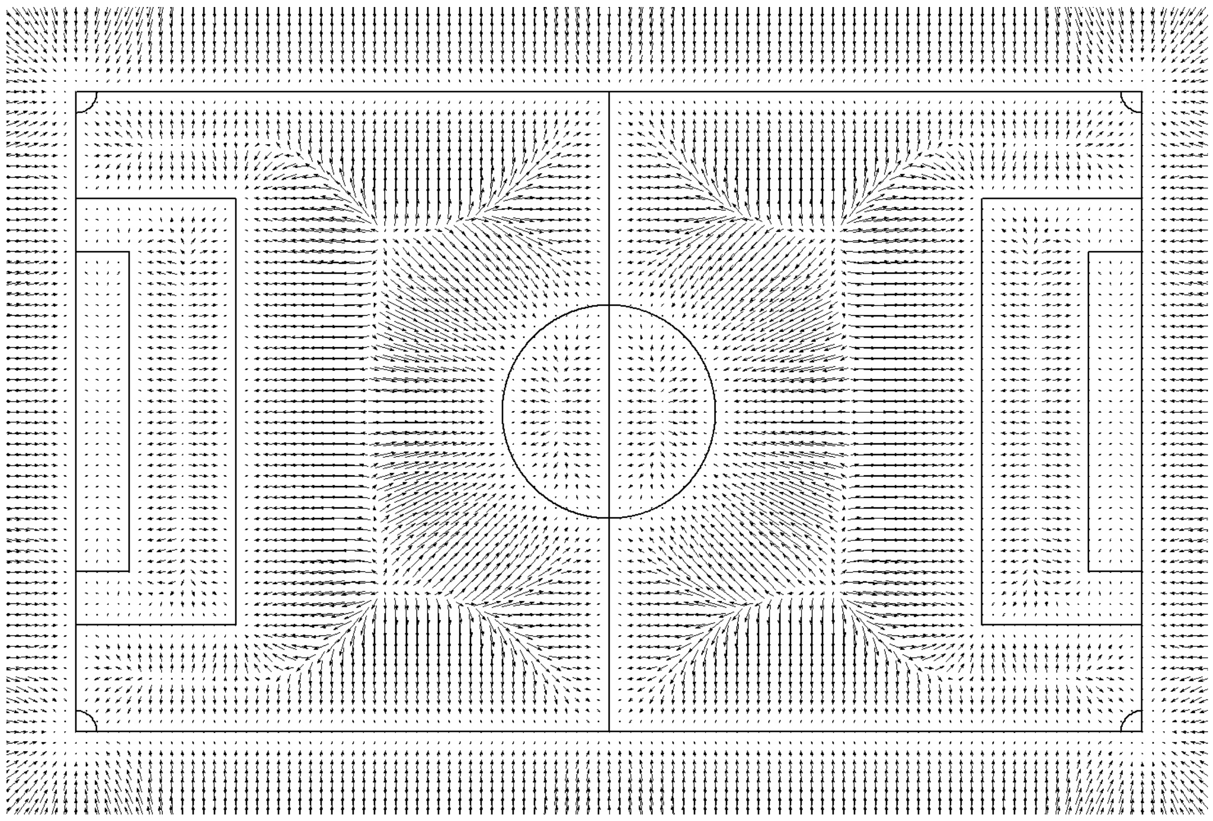


Figure 4.20: The arrows indicate the forces acting on a point at the corresponding position. The length of the arrows indicates the strength of the forces. The force field is used to iteratively calculate a translation and rotation of a point cloud and a corresponding robot position in order to make the points fit to the model of the field lines.

Here, $\mathbf{R}(\pi/2)$ is the rotation matrix which rotates a vector 90 degrees (definition on page 78). Then the overall momentum $M(\breve{\mathbf{x}}, \mathbf{A}, \mathbf{g})$ acting on g with respect to the robot's pose $\breve{\mathbf{x}}$ is

$$M(\breve{\mathbf{x}}, \mathbf{A}, \mathbf{g}) = \frac{1}{n} \sum_{i=1}^{n} m(\breve{\mathbf{x}} \xrightarrow{point} \mathbf{p}_i^r, \mathbf{g}). \tag{4.87}$$

With $\breve{\mathbf{x}}_k = \begin{pmatrix} x_k \\ y_k \\ \phi_k \end{pmatrix}$ we compute $\breve{\mathbf{x}}_{k+1}$ by

$$\begin{pmatrix} x_{k+1} \\ y_{k+1} \end{pmatrix} = \begin{pmatrix} x_k \\ y_k \end{pmatrix} + \gamma \mathbf{F}(\breve{\mathbf{x}}_k, \mathbf{A}) \tag{4.88}$$

$$\phi_{k+1} = \phi_k + \gamma M(\breve{\mathbf{x}}_k, \mathbf{A}, \mathbf{g}). \tag{4.89}$$

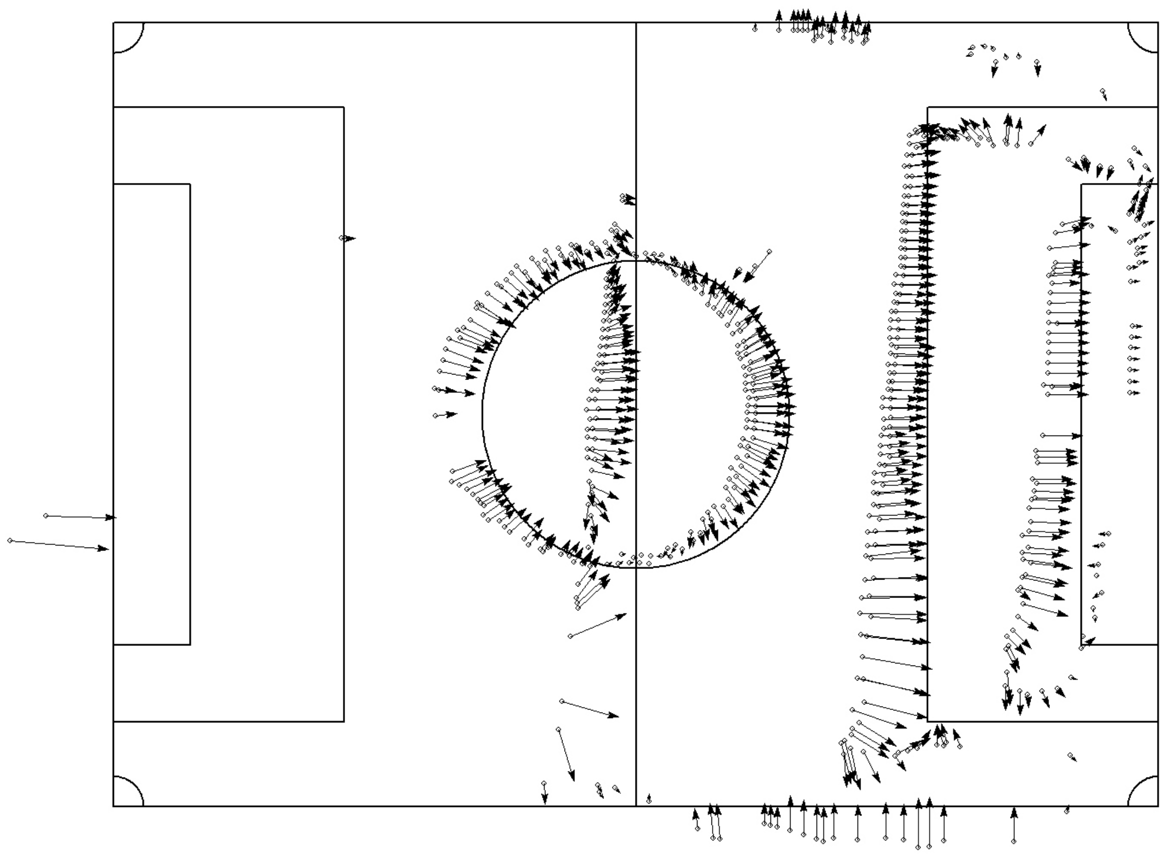


Figure 4.21: This figure shows the force vectors acting on the points. Each force vector is a weighted sum of a list of vectors. The number of vectors in each list is equal to the number of straight and curved line segments in the model of the field lines and each vector points to the closest point of the respective line segment, measured from the current point. The force vectors for all possible locations can be pre-computed, since the model of the field lines is static. Then, given a point, it suffices to look up the force vector in the lookup table. Here, the lookup table is a two-dimensional discretization of the playing field. After discarding outliers, an overall translational force can be computed by a weighted sum of the forces.

Here $\gamma = 0.1$ is a constant, specifying the step width for the iterations. Both figure 4.21 and 4.22 are based on the same exemplary field contours, assuming that they were detected in an image and that they were transformed into the global world coordinate system. The figures show the forces and momenta that would arise, respectively.

In our overall setup, the line contours are extracted every third frame, yielding a frequency of ten with a frame rate of 30 frames per second. With the robot's maximum speed

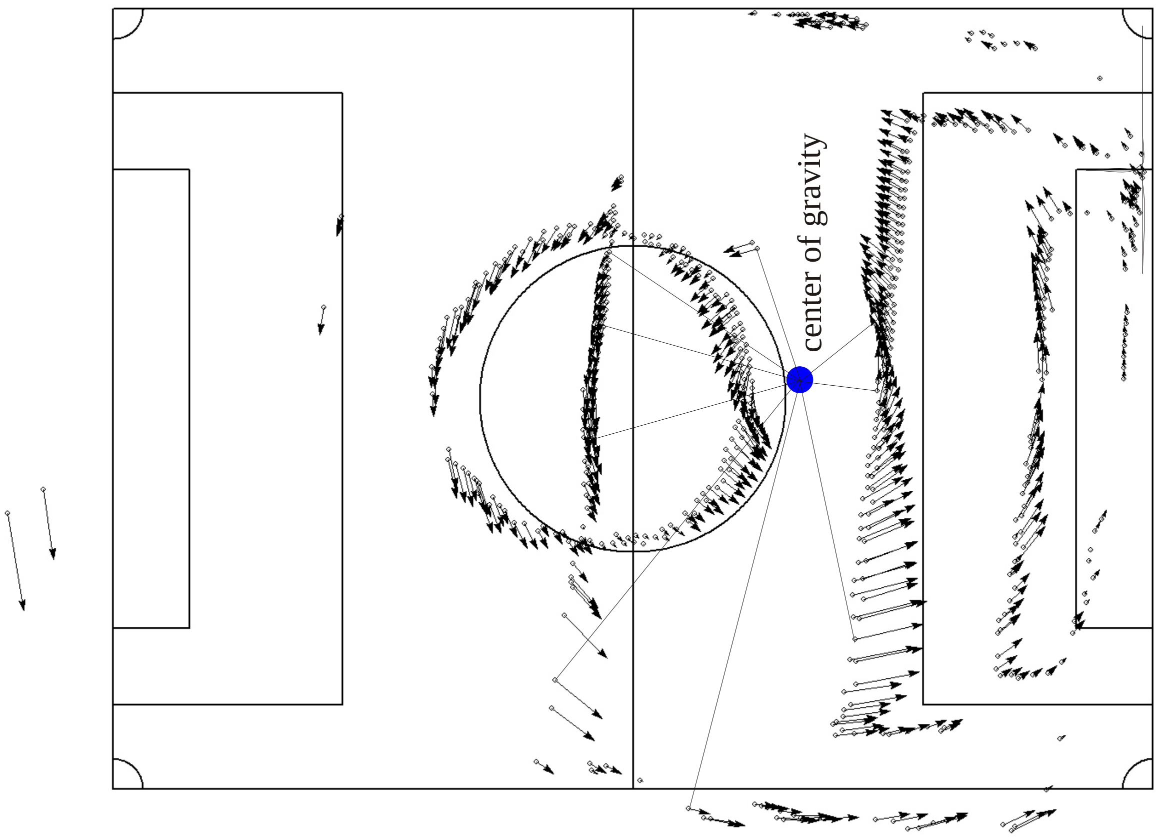


Figure 4.22: Similar to figure 4.21 the forces on the points are considered. However, for calculating a rotational momentum, only the component of the individual force vectors, which are perpendicular to a line connecting the center of gravity and the current point are shown. These components are aggregated to an overall momentum as described in the text.

of approximately 350 cm/s the robot can move a maximum distance of approximately 35 cm. Furthermore, we have a good initial estimate of this movement by odometry. Assuming an odometric error of at most 10%, the error that has to be corrected for can be bounded by approximately 3.5 cm. This is the reason why even one iteration is sufficient to correct for these errors. Thus, using the force field pattern approach for relative visual correction, the final measurement of layer 2 is:

$$\mathbf{z}_{relative\ vision} := \breve{\mathbf{x}}_{k+1}. \tag{4.90}$$

Considering Fusion

Relative localization is processed after image analysis and therefore runs at the frame rate of the video stream. Here, the difficulty arises in using the odometric information that corresponds to the time interval of the respective visual information. This is not trivial because odometry and vision data are subject to different time delays. For instance, vision has a typical delay of 150 milliseconds, while odometry has a delay of approximately 8 milliseconds. For instance, when the robot is standing still and then starts rotating, the rotation will be observed by odometry after 8 milliseconds, but the visual processing will not detect the rotation until 150 milliseconds after the rotation. Here, we can distinguish two different times, the *mental time* at which an event is detected by the processing system, and the *physical time* at which the event happened in the physical world.

When capturing odometric and visual information, we measure the times when the information is processed by making use of the processor's *performance counter*, which allows a time resolution below one microsecond ($10^{-6}s$). These measured times are the *mental times*. To obtain *physical times* we have to subtract the respective time delays of the preprocessing, which includes both hardware and software processing.

We store the incoming odometric information together with the corresponding physical time stamps in a queue. Thus, we always have the data of the last 2-3 seconds and we can access the data by physical times. Within the video callback function we also take a time measurement and subtract the 150-millisecond time delay. Then, for two successive images, we have two physical times and we can search for the corresponding odometry information in the data buffer. Since the data is sorted by times within the data buffer, we can use binary search to efficiently find the respective indices of the time interval and we obtain the corresponding odometric information.

The precise handling of time is not that important for global localization because here, we're only interested in a rough position of the robot. However, relative localization works on a very fine scale in time and space and the resulting position's precision is of great importance for estimating position and speed not only of the robot itself, but also of other objects that are perceived with relative coordinates from the robot's perspective.

Usually, relative localization is active 99 percent of the time and global localization only 1 percent.

4.12.2 Adapting the System Dynamics Approach

This section describes how the system dynamics approach can be applied instead of the MATRIX method to correct for small errors in the estimate of the robot's system state. Based on the model of the field lines, the dynamic model and the observation model, the idea is to predict the appearance and position of the field lines at a few locations in the images and to use only a few detectors to correct the estimate of the system state. In the following, we will describe the kinds of features, the detectors and how the estimate of the system state is corrected. Two different approaches of defining appropriate

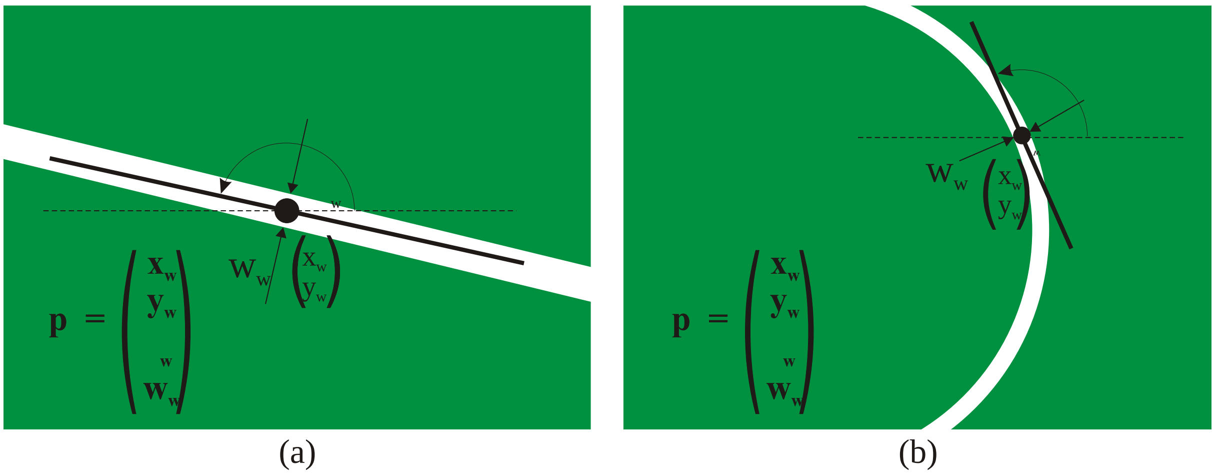


Figure 4.23: In the first approach, we consider a feature to be a point on a field line together with the orientation of a tangent to the field line at the respective point. The two examples (a) and (b) show that the corresponding field line can be straight or curved. The width of the field line at the point is also a component of the corresponding feature vector. In the text, we refer to such a feature as a *tangent point* $\mathbf{p} := (x_w\ y_w\ \phi_w\ w_w)^T$.

features are considered, *tangent points* (see figure 4.23) and *point triplets* (see figure 4.27). The first version, tangent points, results in a complicated mechanism to measure the components of the feature in the image. Also, the resulting mathematical formulation becomes complicated. Thus, the definition of the features is modified later, yielding the point triplet features. Both versions will be described here, because the second approach can most easily be understood when considering the first approach at the beginning. Furthermore, by defining different features, the flexibility of the overall approach becomes apparent.

The basic idea is to consider the orientation and position of the field lines at a few

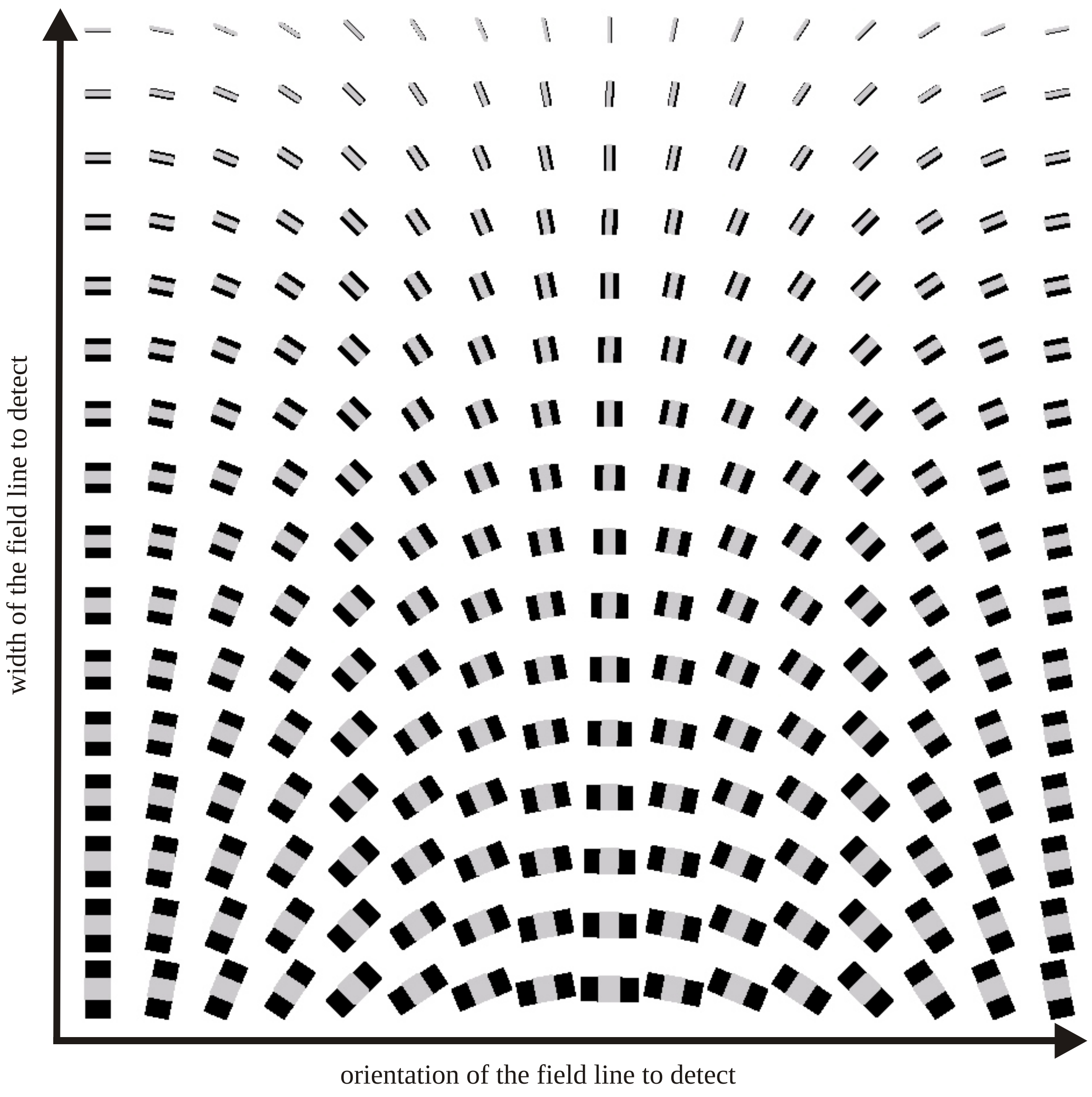

width of the field line to detect
orientation of the field line to detect

Figure 4.24: A set of detectors with 16 different orientations and widths is used for the field line detection. The response of the detector is the average intensity of pixels below the light areas of the detector minus the average intensity of pixels below the dark areas.

locations on the playing field. By observing how these fixation points behave in the image, the estimates of the system state are calculated. Thus, in an initial approach, we define a feature that will later be referred to as a *tangent point*. Of course, a point does not have a tangent. Rather, with *tangent point* we consider a point on a curved or straight field line together with the tangent direction of the corresponding line in the model taken at the respective point. Additionally, we include the width of the corresponding field line as a component of the feature-vector. We denote a tangent point with $\mathbf{p} := (x_w, y_w, \phi_w, w_w)^T$ where x_w and y_w specify its position in the global world coordinate system, ϕ_w is the polar angle of the tangent direction and w_w denotes the width of the line. Here, the subscript "w" emphasizes that the values are specified in world coordinates. This definition is illustrated in figure 4.23. Note that we always know the coupling between a *tangent point* and the curved or straight line, whose tangent at that point is considered. This coupling is important when later calculating the Jacobian matrix $\mathbf{C}$, which relates changes in the system state to changes of features in the images.

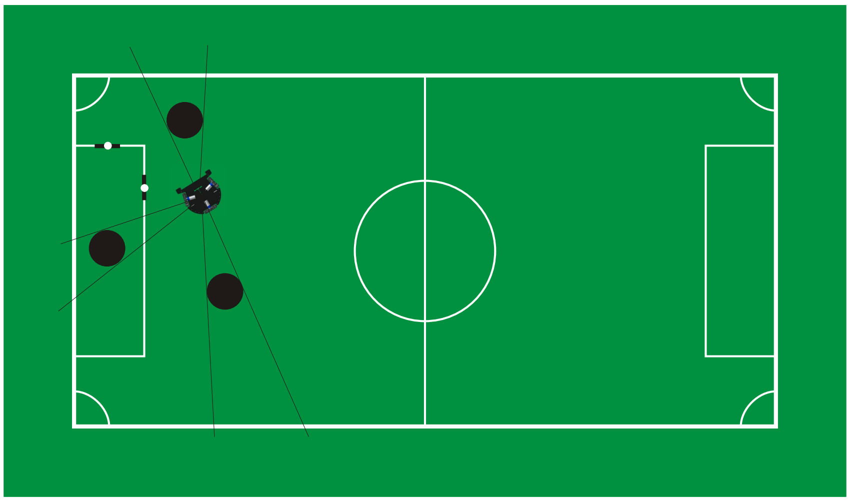


Figure 4.25: Given the position of the robot and the obstacles, two visible tangent points are determined. The points are depicted by small, white disks and the corresponding orientations by small, black line fragments. The cones marked by thin black lines show the parts of the field lines that are occluded by obstacles.

Using the observation model, we can predict the position of a tangent point in the image, but we also want to predict the orientation and width of the corresponding line. For the tangent direction, we consider two points in the world coordinate system that are slightly spaced to each side of the tangent point, map these points onto the image and calculate the angle of their difference vector. Similarly, to predict the width of the line in the image, we consider two points around the tangent point, but this time perpendicularly located at both sides of the field line, enclosing the field line in the middle. We denote this mapping with

$$\mathbf{y}^* = g(\mathbf{p}, \mathbf{x}^*(\mathbf{k}), \mathbf{k_c}), \tag{4.91}$$

where $\mathbf{x}^*(k)$ is the estimated system state at time step k and $\mathbf{k_c}$ are known factors of the optical mapping (i.e. specifying the mirror's center in the image). The resulting vector $\mathbf{y}^* = (x^*,\ y^*,\ \phi^*,\ w^*)^T$ consists of four components describing the predicted position, orientation and width of the projected tangent point in the image.

Our goal is to measure the discrepancy between the predicted feature $\mathbf{y}^*$ and the real feature $\mathbf{y}$. Since we expect the difference to be small, we use $\mathbf{y}^*$ as a starting point for the detection process. That is, we have an initial estimate of the position, orientation and width of the tangent point in the image, and we choose an appropriate detector from a set of pre-computed filters defined for different orientations and widths of the expected line (see figure 4.24).

With the above definition of the tangent points, considering a single point in the image is insufficient to measure the feature's true values $\mathbf{y}$ in the image. At least one, better two auxiliary points have to be used. Figure 4.25 shows the robot close to the penalty area and two tangent points that are used in figure 4.26 to measure the discrepancy between the predicted and real appearance of the feature.

However, the need for auxiliary points in the image to measure the discrepancy in the orientation strongly suggests that the idea of the *tangent point* is misrepresented. Although this representation has the advantage of a small feature-vector - thus, resulting in small matrices later - it is more straight-forward to consider the auxiliary points from the very beginning, that is at the level of the world model.

Hence, we discard the idea of tangent points and instead define a new feature that we refer to as a *point triplet* in the following. A point triplet is simply a set of three points on a straight or curved line, equally spaced at a small distance ($\approx$ 20cm). Figure 4.27 illustrates two cases for straight and curved lines. The corresponding feature vector $\mathbf{p}$ is defined as

$$\mathbf{p} := (x_{w1},\ y_{w1},\ x_{w2},\ y_{w2},\ x_{w3},\ y_{w3})^T, \tag{4.92}$$

with x_{wi}, y_{wi} as the cartesian coordinates of the ith point ($i = 1, 2, 3$) specified in the global world coordinate system. The feature vector $\mathbf{p}$ was already defined for the tangent points. However, discarding this approach, we redefine this vector. Based on the current estimate of the system state, the position of the point triplet in the image can be predicted.

$$\mathbf{y}^*(k) := g(\mathbf{p}, \mathbf{x}^*(k), \mathbf{k_c}). \tag{4.93}$$

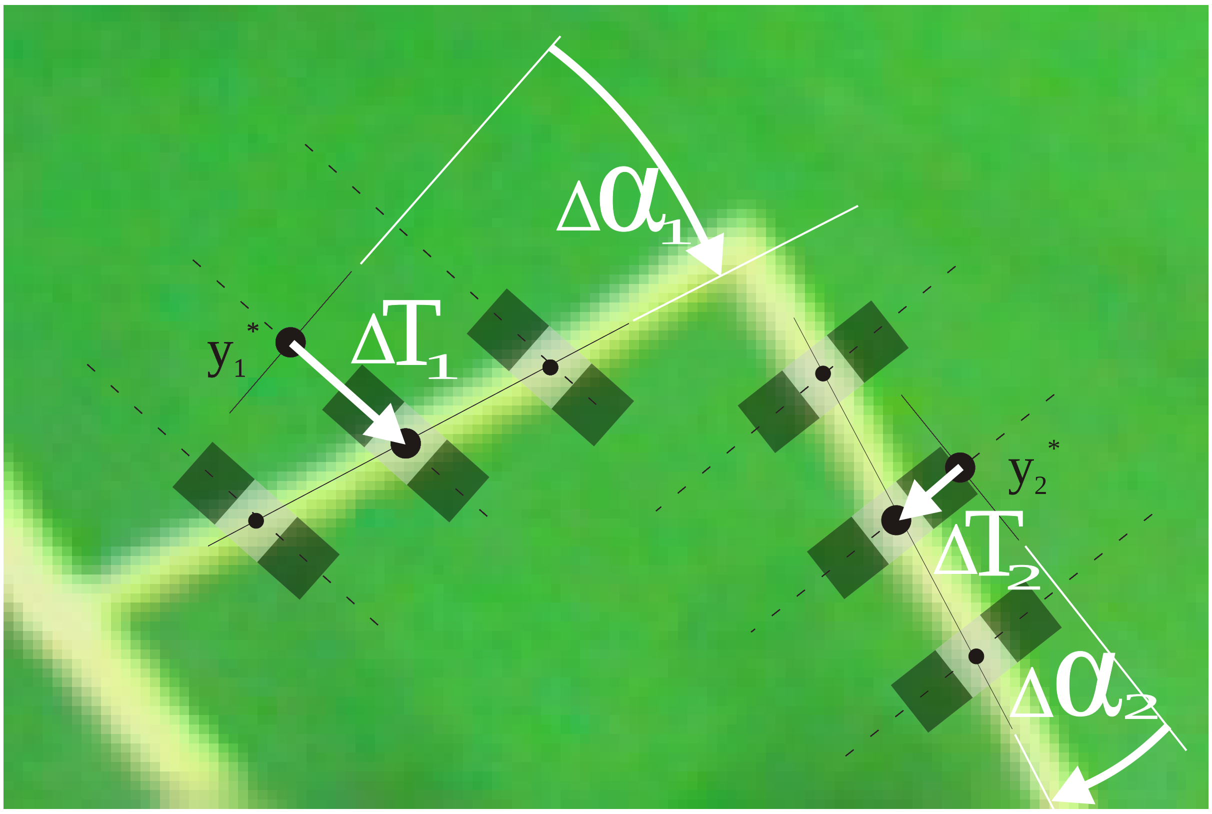


Figure 4.26: This figure shows how the discrepancy between predicted and real measurements is determined using tangent points features. It shows that auxiliary points are required to measure the orientation. This is one reason, why the idea of the tangent points is discarded and replaced by point triplet features later. The figure shows a closeup of an image at the corner of the penalty area. The estimate of the robot's position is slightly wrong. Thus, the predicted feature locations $\mathbf{y}_1^*$ and $\mathbf{y}_2^*$ do not precisely cover the pixels corresponding to the respective field lines. Here, $\mathbf{y}_1^*$ and $\mathbf{y}_2^*$ have three components including the predicted x, y-position plus the predicted orientation. To measure the true location of each feature, three line detectors are applied along straight lines. The lines are oriented perpendicular to the predicted orientation of the corresponding field line. Each of the detectors then finds the position of maximum response, which is on a pixel of the field line. Fitting a straight line to the three detected points then yields the measured orientation of the field line. Later, when we discard tangent points and use point triplets instead, the measurements will solely consist of pixel coordinates, not including angular components.

Now, $\mathbf{y}^*(k) = (x_1^*,\ y_1^*,\ x_2^*,\ y_2^*,\ x_3^*,\ y_3^*)^T$ contains the predicted pixel coordinates of the projected points in the image. Also, we predict the orientation and width of the field line at the respective points and select an appropriate detector. The detectors yield the

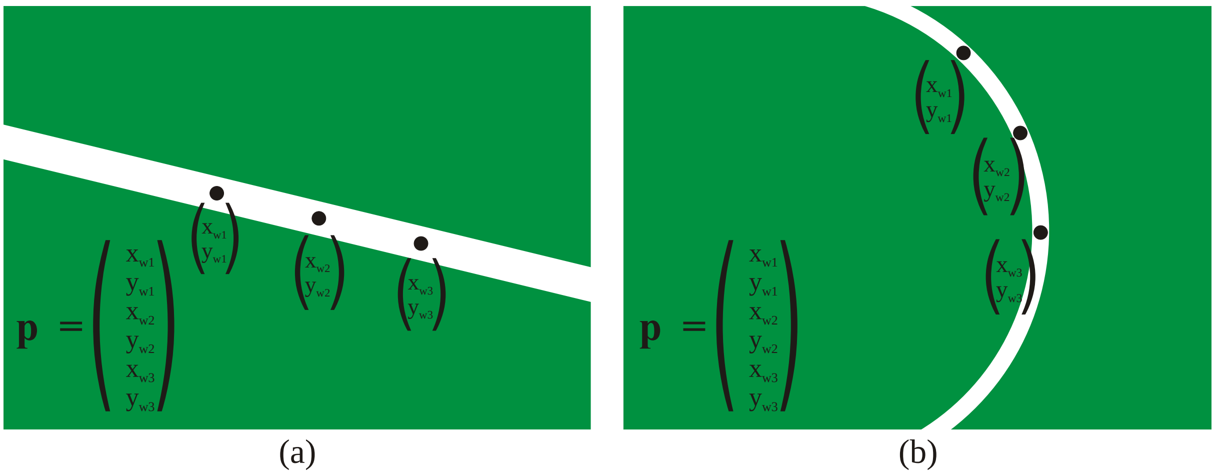


Figure 4.27: In a second approach, we define a *point triplet* to be a feature consisting of three points on a field line equally spaced at a small distance (between 20 and 80cm). The two examples (a) and (b) show that the corresponding field line can be straight or curved. The resulting feature vector has the simple form $\mathbf{p} := (x_{w1},\, y_{w1},\, x_{w2},\, y_{w2},\, x_{w3},\, y_{w3})^T$.

positions of maximal responses, which are summarized in $\mathbf{y}(k) = (x_1\, y_1\, x_2\, y_2\, x_3\, y_3)^T$. This step is illustrated in figure 4.28. Next, we consider the discrepancy between predicted and measured feature values.

$$\Delta\mathbf{y}(k) = \mathbf{y}(k) - \mathbf{y}^*(k) \tag{4.94}$$

There is a coupling between a change $\Delta\mathbf{x}(k)$ of the system state and the measurement discrepancy $\Delta\mathbf{y}(k)$. However, this mapping is non-linear because of the distortion of the image and the angular components in the system state. Note that this non-linearity is typical, not only for omni-directional vision systems, but also for perspective vision. Hence, we linearize the mapping at the predicted estimate $\mathbf{x}^*(k)$ of the system state.

$$\Delta\mathbf{y}(k) = \mathbf{C}(k)\Delta\mathbf{x}(k) \tag{4.95}$$

Here,

$$\Delta\mathbf{x}(k) = \mathbf{x}(k) - \mathbf{x}^*(k), \tag{4.96}$$

and $\mathbf{C}(k)$ is the Jacobian at time step k, the derivation of the vector function g (see equation 4.93) with respect to the system state $\mathbf{x}$. We will describe later how it is calculated. At this point, we consider $\mathbf{y}(k)$ to contain only one point triplet. In this case, the mapping described by matrix $\mathbf{C}(k)$ is typically not bijective, that is, $\mathbf{C}(k)$ cannot be

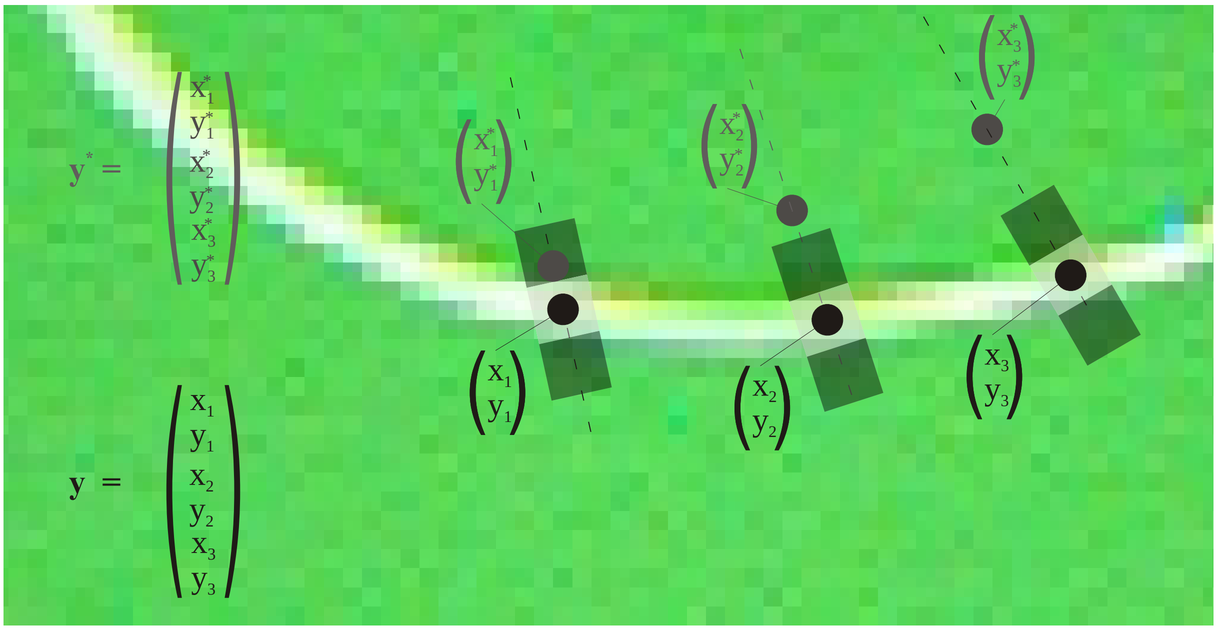


Figure 4.28: The position of a point triplet in the image $\mathbf{y}^* = (x_1^*,\ y_1^*,\ x_2^*,\ y_2^*,\ x_3^*,\ y_3^*)^T$ is predicted based on the current estimate of the system state. A local detector is applied along lines orthogonal to the predicted orientation of the field lines at the respective points. Based on the prediction of the field line widths and orientations, the appropriate detector is selected out of a set of pre-computed detectors that vary in width and orientation. The detectors then measure the true position of the point features in the image $\mathbf{y} = (x_1,\ y_1,\ x_2,\ y_2,\ x_3,\ y_3)^T$.

inverted. Hence, it is not possible to calculate the system state by replacing $\Delta\mathbf{x}(k)$ in equation 4.95 with equation 4.96 and solving for $\mathbf{x}(k)$:

$$\mathbf{x}(k) = \mathbf{x}^*(k) + \mathbf{C}(k)^{-1}\Delta\mathbf{y}(k). \tag{4.97}$$

Moreover, such a calculation would contradict the overall approach, since we want to include the dynamic and measurement models to interpret the discrepancy $\Delta\mathbf{y}(k)$. Indeed, we will collect more than one point triplet in the vector $\mathbf{y}(k)$ later, and $\mathbf{C}(k)$ will be nonsingular in many cases. But even then, we do not use equation 4.97 to calculate the system state. Instead, we apply a Kalman filter:

$$\mathbf{x}^\wedge(k) = \mathbf{x}^*(k) + \mathbf{K}(k)\Delta\mathbf{y}(k). \tag{4.98}$$

Here, $\mathbf{K}(k)$ is the so-called gain matrix, which is calculated by

$$\mathbf{K} = \mathbf{P}^*\mathbf{C}^T(\mathbf{C}\mathbf{P}^*\mathbf{C}^T + \mathbf{R})^{-1}. \tag{4.99}$$

Here, $\mathbf{P}^*$ is the current error covariance matrix. The matrix is updated by

$$\mathbf{P} = \mathbf{P}^* - \mathbf{K}\mathbf{C}\mathbf{P}^*. \tag{4.100}$$

The number of rows and columns of the matrix $\mathbf{P}$ 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 $\mathbf{y}$. The matrix $\mathbf{R}$ contains the covariances between the individual quantities of the measurement vector $\mathbf{y}$. These covariances depend on the currently selected features. A feature, which is close to the robot gives rise to smaller entries than a distant feature. The reason is, that closer objects have a higher resolution in the image due to the catadioptric image formation.

The Jacobian $\mathbf{C}(k)$ plays a fundamental role in the overall approach. The components are the partial derivatives of the vector-function g, describing the optical mapping, with respect to the components of the system state $\mathbf{x}$, taken at $\mathbf{x}^*$.

Remember, that the system state was defined as

$$\mathbf{x} = \begin{pmatrix} x, & y, & \phi, & v_x, & v_y, & \omega \end{pmatrix}^T, \tag{4.101}$$

with x,y,ϕ representing the robots position and orientation (the robot's *pose* or *reduced system state*) and v_x, v_y, ω specifying the translational and rotational velocities (see page 73). The appearance of features in the image not only depends on the robot's pose and other objects, but also on its translational and rotational velocities. For instance, when the robot rotates very fast and the exposure time of the camera is short, i.e. because of dim light, then features like the field lines tend to smear along the rotational direction. However, in the following, we neglect these effects. It suffices to consider the partial derivatives with respect to $\check{\mathbf{x}}$, the reduced system state. When considering only one point triplet $\mathbf{p}(k) = (x_{w1}\, y_{w1}\, x_{w2}\, y_{w2}\, x_{w3}\, y_{w3})^T$ and setting

$$(x_1\, y_1\, x_2\, y_2\, x_3\, y_3)^T := \mathbf{y}(k) = g(\mathbf{p}, \mathbf{x}(k), \mathbf{k_c}) \tag{4.102}$$

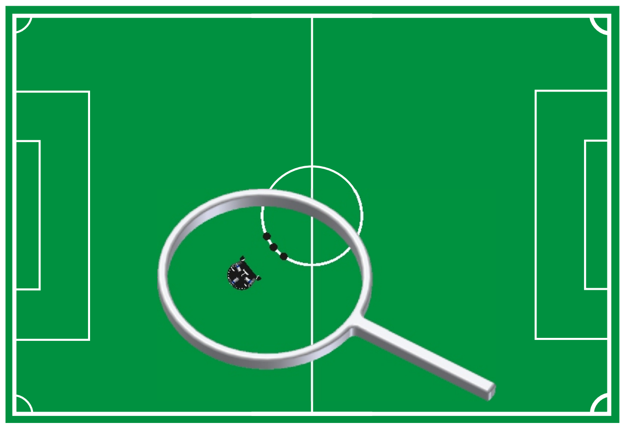
the Jacobian has the form:

$$\mathbf{C}(k) := \begin{pmatrix} \frac{\delta x_1}{\delta x} & \frac{\delta x_1}{\delta y} & \frac{\delta x_1}{\delta \phi} \\ \frac{\delta y_1}{\delta x} & \frac{\delta y_1}{\delta y} & \frac{\delta y_1}{\delta \phi} \\ \frac{\delta x_2}{\delta x} & \frac{\delta x_2}{\delta y} & \frac{\delta x_2}{\delta \phi} \\ \frac{\delta y_2}{\delta x} & \frac{\delta y_2}{\delta y} & \frac{\delta y_2}{\delta \phi} \\ \frac{\delta x_3}{\delta x} & \frac{\delta x_3}{\delta y} & \frac{\delta x_3}{\delta \phi} \\ \frac{\delta y_3}{\delta x} & \frac{\delta y_3}{\delta y} & \frac{\delta y_3}{\delta \phi} \end{pmatrix} \tag{4.103}$$

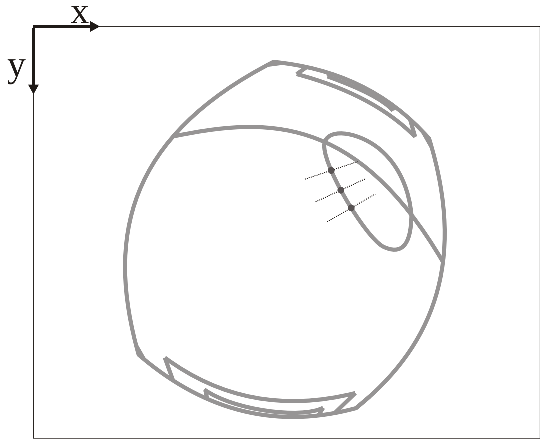
Here, we omitted the variable k in the components of the matrix for the sake of readability, but of course, the matrix is not static, instead depending on the time step k.

Because of the distortion by the robot's mirror, the matrix $\mathbf{C}(k)$ cannot be computed analytically. Instead, it has to be computed numerically in each time step k by combining knowledge of the model of the field lines, the current estimate of the system state and the detection process.

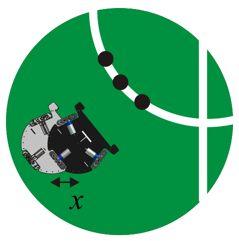
The differential components of the Jacobian $\mathbf{C}(k)$ at time step k are approximated by finite differences. For that approximation, it is predicted how the measurements of the detectors evolve in the image when individual components of the system state are slightly modified. That is, small, finite differences Δx, Δy and $\Delta \phi$ are added to the components of the current estimate $\mathbf{x}^*(k)$ of the system state and the resulting geometric constellation of the field lines in the image, before and after the modification, is investigated. Predicting the image position of the currently used features, the question is, what the corresponding detectors would measure if the state changed as supposed by the finite differences. Since the measurements of the detectors are determined by applying them along straight scan lines and returning the position of maximum response, a prediction of the measurement can be made by calculating the intersection points between hypothetical scan lines and the predicted field lines according to the modified system state. Relating the measurements to the finite differences in the system state yields the approximate components of the Jacobian matrix $\mathbf{C}$. This calculation is illustrated in figure 4.29.



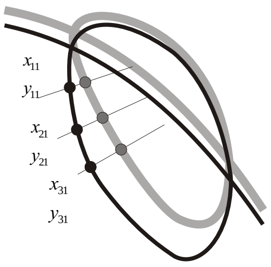
(a)



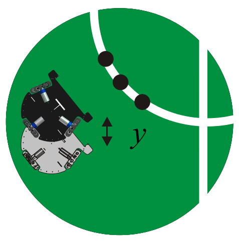
(b)



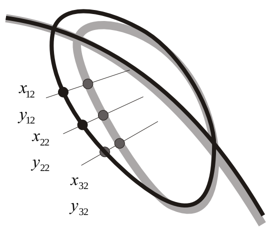
(c1)



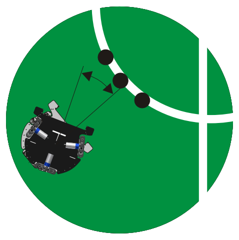
(c2)



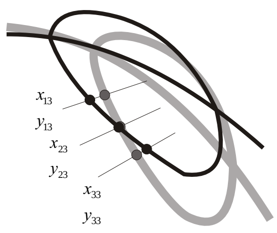
(d1)



(d2)



(e1)



(e2)

$$C(k) \quad \begin{array}{ccc} \frac{x_1}{x} & \frac{x_1}{y} & \frac{x_1}{} \\ \frac{y_1}{x} & \frac{y_1}{y} & \frac{y_1}{} \\ \frac{x_2}{x} & \frac{x_2}{y} & \frac{x_2}{} \\ \frac{y_2}{x} & \frac{y_2}{y} & \frac{y_2}{} \\ \frac{x_3}{x} & \frac{x_3}{y} & \frac{x_3}{} \\ \frac{y_3}{x} & \frac{y_3}{y} & \frac{y_3}{} \end{array} \qquad \begin{array}{ccc} \frac{x_{11}}{x} & \frac{x_{12}}{y} & \frac{x_{13}}{} \\ \frac{y_{11}}{x} & \frac{y_{12}}{y} & \frac{y_{13}}{} \\ \frac{x_{21}}{x} & \frac{x_{22}}{y} & \frac{x_{23}}{} \\ \frac{y_{21}}{x} & \frac{y_{22}}{y} & \frac{y_{23}}{} \\ \frac{x_{31}}{x} & \frac{x_{32}}{y} & \frac{x_{33}}{} \\ \frac{y_{31}}{x} & \frac{y_{32}}{y} & \frac{y_{33}}{} \end{array}$$

(f)

Figure 4.29: The Jacobian **C** is calculated numerically by finite differences. Figure (b) shows the field lines as they would appear to the robot when it is located as in (a). In (c1), (d1) and (e1), the robot's current pose is slightly modified in one of its components $(x, y, \phi)^T$. The original pose is compared to each of the modified poses; in particular, the corresponding predictions of the field lines in the image are compared. Assuming a single point triplet, the response of the detector is predicted by calculating where the detector's scan lines would intersect the geometry in the image that corresponds to the modified poses. In figure (c2), (d2) and (e2) the gray contours reflect the field lines that correspond to the robot's original pose. Each is the same closeup of (b). The black lines reflect the field lines corresponding to the modified poses, and the straight scan lines and black dots illustrate the expected detector measurements. To calculate the intersection points, only the local geometry around the point triplet has to be used. Thus, figures (c-e) show cut-outs. Relating the expected detector measurements to the modification of the robot's pose yields an approximation of the entries into the Jacobian matrix (f).

Up to this point, only one point triplet was used to correct the current estimate of the system state. Now, we will use several point triplets and collect all their components in a single feature vector.

The point triplets used will not remain the same over time. Rather, a selection algorithm will dynamically choose different features. But how many and which features should be used?

A feature selection process must carry out the following three steps. First, visible point triplets have to be determined. For that, the portions of the field lines that are not occluded by obstacles have to be determined. Second, point triplets on the field that can be easily recognized with feature detectors have to be selected. They should not be close to obstacles, corners or junctions of the field lines to avoid detection problems. Also, they should not be too distant. Third, from the set of potential point triplets, the combination most suited to correct the current estimate of the system state must be determined.

The feature selection process must consider the obstacles to avoid the selection of point triplets that will be occluded in the image. Obstacles can also be tracked with the system dynamics approach. However, in the following, we simply assume that we know these obstacle positions.

When observing the playing field from above, the left and right side of an obstacle form a cone together with the position of the robot. Field lines within this cone and behind the obstacle are occluded in the omni-directional images (see figure 4.30). Thus, if the intersection points between the field lines and the sides of the cones are calculated, a list of visible curved and straight line segments can be computed as illustrated in figure 4.30. On each visible line segment, one or several potential point triplets can be considered. However, the line segments should be long enough to allow a point triplet to be placed in some distance w from the end points of the segment. Otherwise, they should be discarded to avoid detection problems due to imprecisions in the current estimate of the system state. If a line segment is long enough, several point triplets can be placed on it, with the constraint that they are spaced at a distance of at least $d = 10$cm.

It is important to understand that a single point triplet on a straight line is not able to recover the component of a movement that is parallel to the line. This is the well-known aperture problem. However, by selecting constellations of point triplets that are not parallel, this problem can be overcome.

After having selected a set $\mathbf{H}$ of l potential point triplets, the best k (i.e. k=2) should

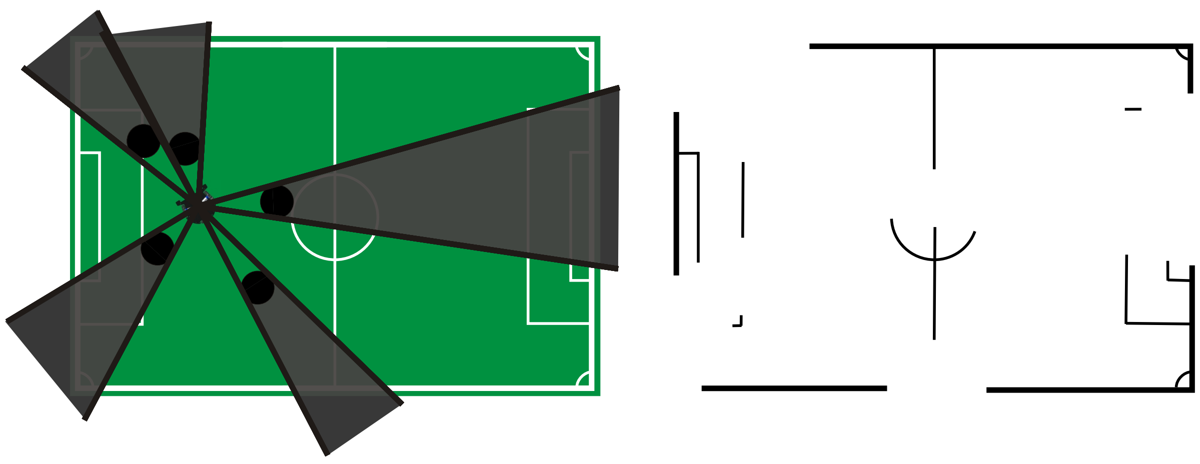


Figure 4.30: The shaded areas in (a) show sections occluded by obstacles. The visible parts of the field lines (b) can be determined by calculating the intersection points between field lines and rays starting at the robot's position and passing through each side of the obstacles.

be selected. Here, the best point triplets are those that allow the best estimate of the system state, that is, with the "smallest" resulting error covariance matrix $\mathbf{P}$. The error covariance matrix $\mathbf{P}$ is recursively computed from the old matrix $\mathbf{P}^*$ by:

$$\mathbf{P} = \mathbf{P}^* - \mathbf{KCP}^* \tag{4.104}$$

Wünsche developed a criterion for the efficiency of a feature combination in his doctoral thesis [86]. Underlying a Gauss-Markov estimator, the error covariance matrix is calculated by

$$\mathbf{P} = (\mathbf{C}_N^T \mathbf{C}_N)^{-1}, \tag{4.105}$$

where

$$\mathbf{C}_N = \sqrt{\mathbf{R}}^{-1} \mathbf{C}_R \mathbf{S}. \tag{4.106}$$

Here, $\mathbf{C}_R$ is the Jacobian matrix based on the reduced system state, and $\mathbf{S}$ is a scaling matrix that is necessary to relate errors in the individual components of the system state to each other. Wünsche defined

$$J = |\mathbf{C}_N^T \mathbf{C}_N| = |\mathbf{P}^{-1}| \tag{4.107}$$

as a criterion to decide for a feature combination. The feature combination that maximizes J should be selected. In this way, the determinant of $\mathbf{P}$ is minimized, favoring small diagonal elements in $\mathbf{P}$, which represent the error variances of the system state components. At the same time, uncorrelated errors are favored. Using a Kalman filter to calculate the error covariance matrix $\mathbf{P}$, the definition

$$J := |\mathbf{P}^{-1}| \tag{4.108}$$

is the analog to equation 4.107. However, the problem is that the determinant does not consider the different units of the components of $\mathbf{P}$. For instance, a variance of 1.0 radians in the orientation of the robot is much more severe than a variance of 1.0 centimeters in the cartesian position. Thus, the definition should be adjusted to

$$J := |\mathbf{S}^2\mathbf{P}^{-1}|, \tag{4.109}$$

with $\mathbf{S}$ being a scaling matrix that relates the components to each other. Selecting k features from the set $\mathbf{H}$ of l potential features, $\binom{l}{k}$ combinations are possible. In many applications the number of potential features is restricted. With $k = 2$ and $l = 20$ this would result in $\frac{20 \cdot 19}{2} = 190$ required evaluations of the criterion J. Selecting 4 from 10 features produces 61 combinations, which is a treatable computational load. However, when k and l increase, the combinatorial possibilities explode. For instance, when selecting 6 from 50 possible features, approximately $15 \cdot 10^6$ combinations are possible.

In order to reduce the computational load, Wünsche's idea was, not to compute the best combination of features in each step, but instead to try to improve the current feature combination by exchanging single features. In this way, each of the currently used k features must be considered to be exchanged with one of the $k-l$ unused features. That is, the criterion J must be evaluated only $k(k-l)+1$ times. In the example, where 6 from 50 features have to be selected, only $50 \cdot 44 + 1 = 2201$ combinations have to be considered in each step. Compared to the $15 \cdot 10^6$ combinations that had to be evaluated before, this corresponds to a reduction of the computational load of two orders of magnitude.

4.13 Layer 3: Feature Recognition

In the third layer, we run a feature detection process that allows us to find sub-structures like corners or the center circle, which yield strong hints for the robot's position. A similar approach has been done in [39], where straight lines and circles are detected for localization. Our approach differs from the previous method in the way we detect the features. While the approach in [39] is based on the Hough transform, which is computationally very expensive, we detect the features using a *constructive approach*. That is, we try to iteratively compose features from smaller parts. Our method is based on the representation of the field lines as obtained by the new region tracking algorithm described on page 66. Here, the field lines are already organized as chains of points.

4.13.1 Representation of the Line Contours

We use our region tracking algorithm to extract the field lines from the images. We determine the boundary of all green regions in the images and we search for green-white-green transitions perpendicular to the boundary curves. After having extracted the lines, they are represented by the pair (P, C) where $P = p_0, ..., p_{n-1}$ is a set of n points with cartesian x,y-coordinates and $C = c_0, ..., c_{l-1}$ supplies connectivity information that partitions P into l point sequences. Here, each $c_i = (s_i, e_i)$ is a tuple of indices determining the start and end point in P that belong to the corresponding point sequence. That is, point sequence i consists of the points $p_{s_i}, ..., p_{e_i}$. By manipulating the connectivity information C, point sequences can efficiently be split or merged.

The line contours as extracted from the images are distorted due to the mirror in the omni-directional vision system. In the following, we assume that the distortion has been compensated (see page 98). However, even without removing the distortion correctly, we are still able to detect most of the features. Figure 4.31 can provide an impression of the initial data. In many cases, there are long point sequences that correspond precisely to the shape of the field lines. However, there are also outliers, missing lines, and small line fragments due to occlusion and detection errors.

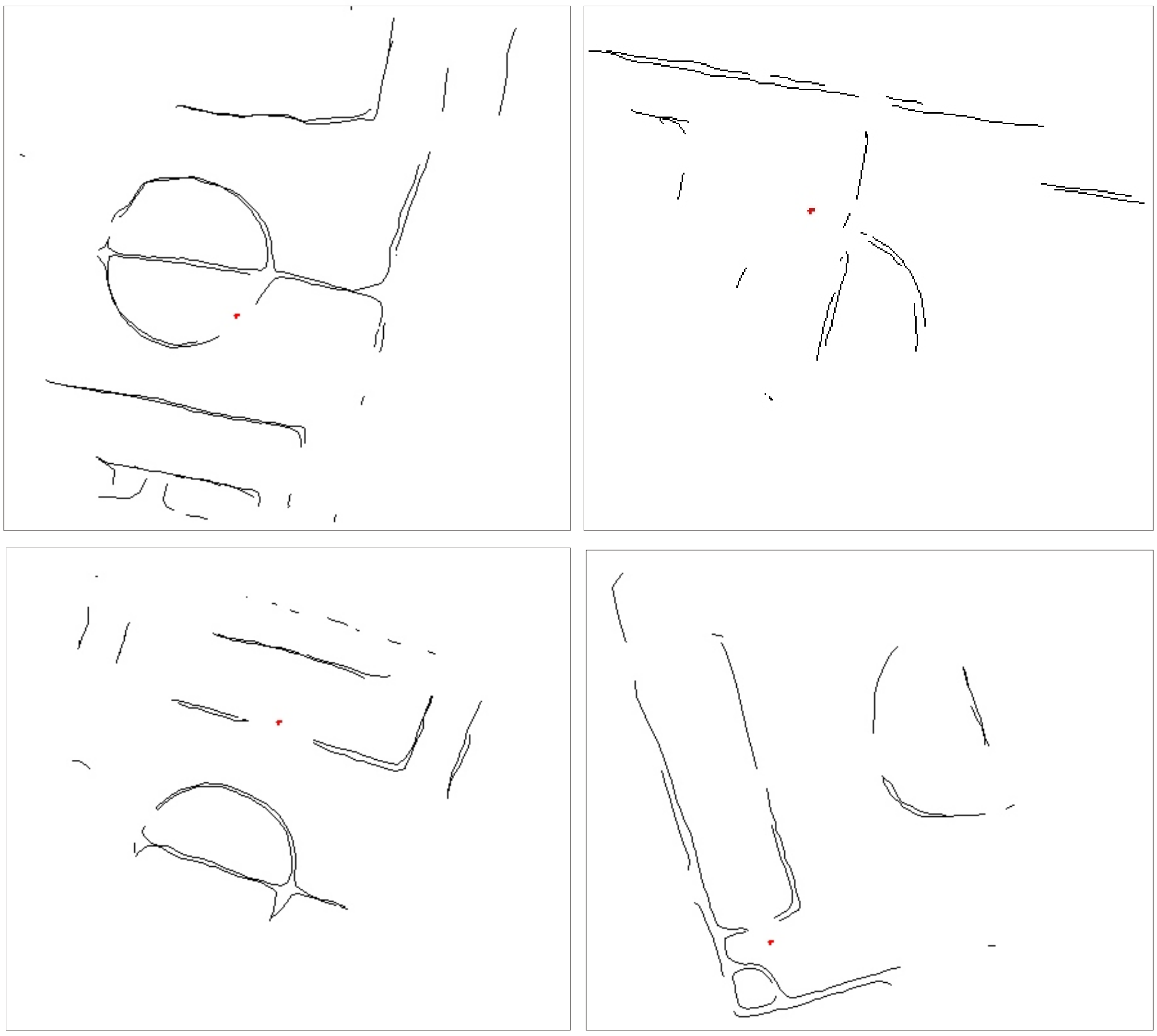


Figure 4.31: Four examples of extracted line contours. Some long sequences of points correspond precisely to the shape of the field lines. But there are also outliers, missing lines, and small line fragments due to occlusion and detection errors.

4.13.2 Quality and Quantity of Features

In the following sections, we will concentrate on the recognition of specific features which we refer to as *high-level features*. The idea of hierarchically ordering the features stems from the fact that there exist features, such as the center circle, which yield a unique clue of the robot's position, while others give rise to a whole set of possible positions. Figure 4.32 provides a first idea of different feature types that might be detected in the line contours.

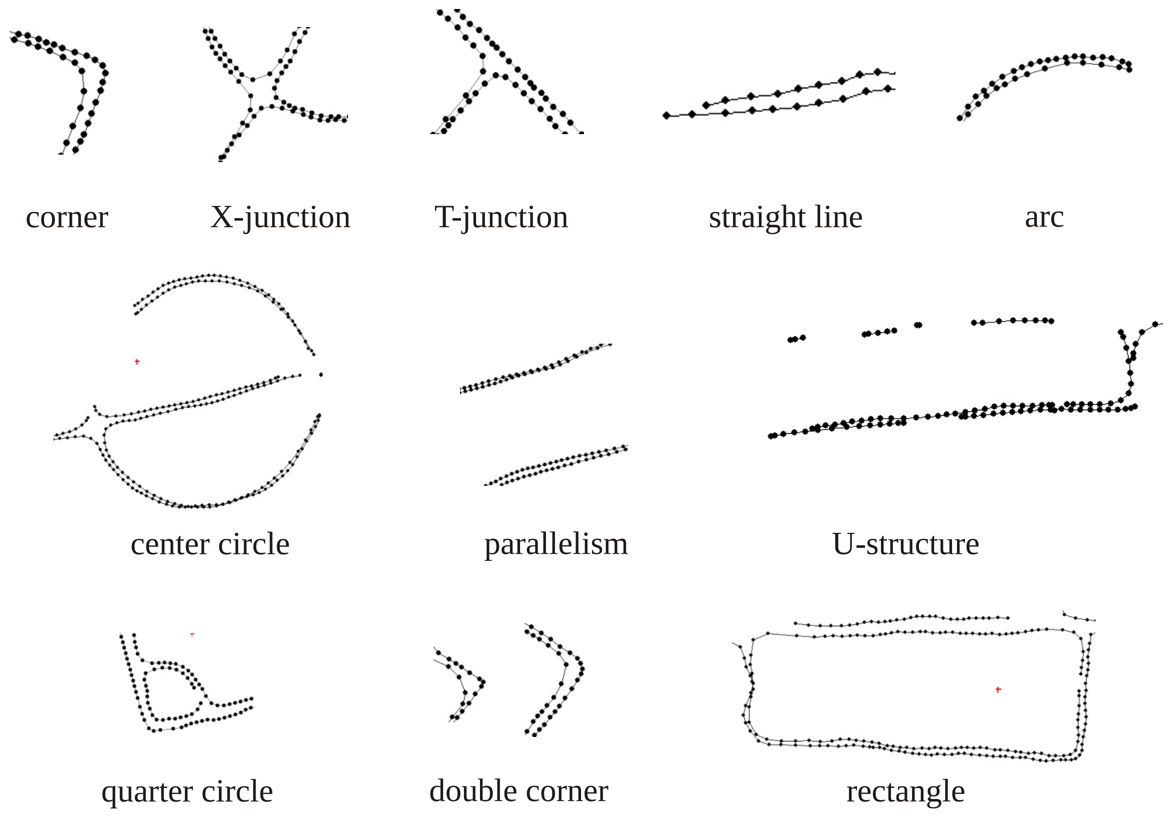


Figure 4.32: Examples of different types of features are illustrated. One can distinguish between *low-level features* that give rise to many possible robot poses or *high-level features* which yield only a few possible robot poses from which the feature could have been observed.

In order to put the feature detection process onto a well-defined base, we first establish a mathematically sound definition of what a high-level feature is. For that, we define two terms, the *quality* and the *quantity* of a feature type in the following. The *quality* of a feature type is the dimension of the space of possible robot poses once a feature of the respective type is detected. The *quantity* is the frequency of a feature in the task domain. Of course, we will favor features with a low quantity number. In the following we will establish these definitions more precisely.

In the context of the field lines, one of the simplest possible features is a single point on a line. When detecting such a point $\mathbf{p}^i$ in the image (subscript i for image), the point can be transformed from the image coordinate system into the local coordinate system of the robot (applying the distance function). Let's denote this representation with $\mathbf{p}^r$ (subscript r for robot).

$$\mathbf{p}^r := \mathbf{Y}_f^{-1}(\mathbf{p}^i) \tag{4.110}$$

Here $\mathbf{Y}_f^{-1}$ is the generic function which maps the feature f from image space to local world space.

Then we know the angle at which the point is seen and its distance. Let us assume for the moment that there is no uncertainty in the measurement. Then the question is: Where could the robot be so that a point on a field line is seen at the given angle and distance?

In order to express this question more formally, we have to make some definitions. First of all, given a feature type f (i.e. a point, line, corner,...) we define $\mathbf{L}_f$ to be the set of all poses (position and orientation) where such features exist within the field model, specified within the global coordinate system. For instance, $\mathbf{L}_p$ is the set of all points on the field lines, $\mathbf{L}_l$ the set of all straight line segments, $\mathbf{L_c}$ the set of all corners, etc...

Then given a feature type f, we ask for the set $\mathbf{X}_f$ of robot poses, such that for all $\breve{\mathbf{x}} \in \mathbf{X}$ there exists an instance $\mathbf{p} \in \mathbf{L}_f$, such that $\mathbf{p}$ would appear at the location of $\mathbf{p}^i$ in the image:

$$\mathbf{X}_f(\mathbf{p}^i) := \left\{ \breve{\mathbf{x}} | \exists \mathbf{p} \in \mathbf{L}_f : \mathbf{Y}_f(\breve{\mathbf{x}} \xrightarrow{point} \mathbf{p}) = \mathbf{p}^i \right\} \tag{4.111}$$

Note, that each element in $\mathbf{X}_f$ is a three dimensional vector with the last component specifying an angle. This last component complicates the following definitions, and we will first consider $\tilde{\mathbf{X}}_f$, which is $\mathbf{X}_f$ without the final component. Typically, the set $\tilde{\mathbf{X}}_f$ can be thought of as a union of k sets $\tilde{\mathbf{X}}_{1f}, \tilde{\mathbf{X}}_{2f}, ..., \tilde{\mathbf{X}}_{kf}$, with each set being an affine subset of $I\!R^2$. Then we define the *quantity* of a feature type to be k and the quality to be the dimension of the $\tilde{\mathbf{X}}_\mathbf{i}$ $i = 1, 2, ..., k$. In the case that the sets have different dimensions (which is not the case in this thesis) one should more carefully define the quality as:

$$\tilde{quality}(f) = max_{i=1,2,...,k}(dim(\tilde{\mathbf{X}}_{\mathbf{i}f})) \tag{4.112}$$

For instance, if we consider the center circle to be a feature type, $\mathbf{X}_{center\,circle}$ consists of two possible robot poses, one on each side of the playing field, from which the robot could have perceived the detected center circle $\mathbf{p}^i$ in the image. Thus, $\tilde{\mathbf{X}}_{center\,circle}$ decomposes into two subsets of dimension zero. Hence, $quality(center\ cirlce) = 0$. Accordingly the *quantity* of the center circle feature is two. If we consider a single point as a feature, the quality is two, because $\tilde{\mathbf{X}}_{point}$ consists of areas of possible points. The quantity in this case is infinite.

We will consider another example with straight line segments as features. For the sake of simplicity, imagine that our world model $\mathbf{L}_{straight\,lines}$ would consist of only 1 (instead of 17) straight line segment(s) and that $\mathbf{l}^i$ is a detected line segment in the image and $\mathbf{l}^r$ is the corresponding segment in the local coordinate system of the robot as depicted in figure 4.33b). Then figure 4.33c) shows the set $\tilde{\mathbf{X}}_{straight\,lines}$ and its decomposition into the sets $\tilde{\mathbf{X}}_{\mathbf{1}straight\,lines}$ and $\tilde{\mathbf{X}}_{\mathbf{2}straight\,lines}$.

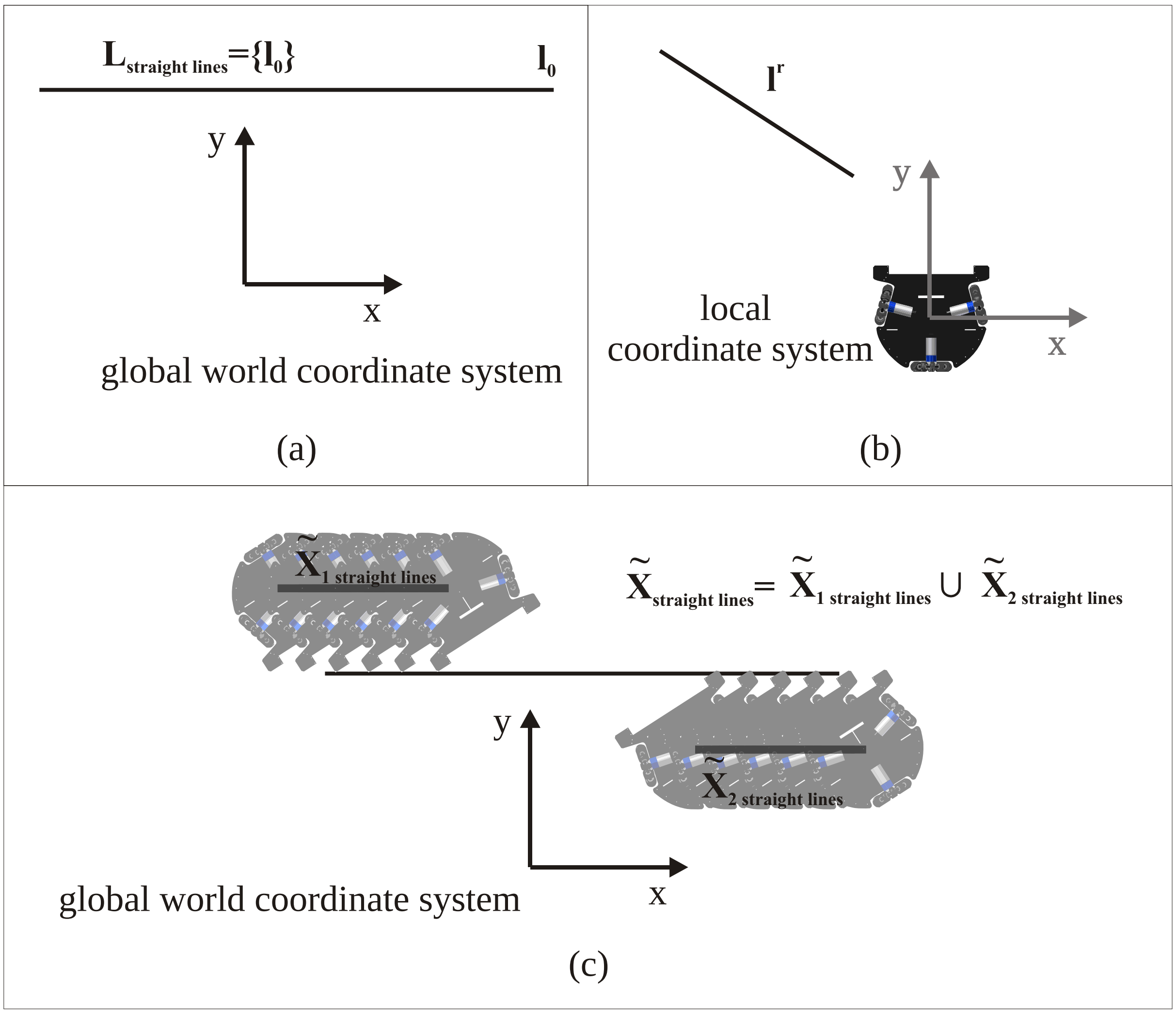
Now we can establish a hierarchy of features. At the top are features with quality zero, which we refer to as *high-level features*. Features having the same quality are ordered by their quantity with lower quantities yielding a higher position in the hierarchy.

In the following sections, we will describe a complex feature detection method. The method begins with low-level features such as points and lines and iteratively constructs higher-level features. At the very top of the construction process, there are five different high-level features: the center circle, and four different corners arising from the structure of the penalty area. Before we detail the detection process, we will first describe how the pose of the robot, which is the measurement vector $\mathbf{z}_{rec}$ of the third layer, can be inferred once a high-level feature is detected.

4.13.3 Direct Pose Inference by High-Level Features

Figure 4.34 shows the five high-level features we are going to detect. The four corners seem to be congruent, but their location within the penalty area will be recognized such that left,right, inner and outer corners can be distinguished.

Each feature type defines a local coordinate system. We refer to the origin of this



$L_{straight\ lines}=\{l_0\}$
l_0
y
x
global world coordinate system
(a)
l^r
y
local coordinate system
x
(b)
$\tilde{X}_{1\ straight\ lines}$
$\tilde{X}_{straight\ lines} = \tilde{X}_{1\ straight\ lines} \cup \tilde{X}_{2\ straight\ lines}$
y
$\tilde{X}_{2\ straight\ lines}$
global world coordinate system
x
(c)

Figure 4.33: This figure illustrates the definitions of *quality* and *quantity* of a feature. In this example, the feature type *"straight line segment"* is considered and it is assumed that the world model $\mathbf{L}_{straight\,lines}$ consists only of one line l_0. (see (a)). In (b) the robot is shown having detected a line l^i in the image which is transformed into the robot's local coordinate system, where it is denoted with l^r. Then figure (c) depicts the sets $\tilde{\mathbf{X}}_{\mathbf{1}straight\,lines}$ and $\tilde{\mathbf{X}}_{\mathbf{2}straight\,lines}$, which constitute the positions from which the robot could have seen the line in order that its perception fits to the model $\mathbf{L}_{straight\,lines}$. Since each set $\tilde{\mathbf{X}}_{\mathbf{1}straight\,lines}$ represents a one-dimensional line segment $quality(straight\,line) = 1$ and since there exist two sets, $quantity(straight\ line) = 2$.

system together with the orientation of the y-axis as a reference mark. Figure 4.35 shows the reference marks for the five high-level features. By specifying the pose of the reference marks in the global coordinate system, the locations of the features are uniquely given.

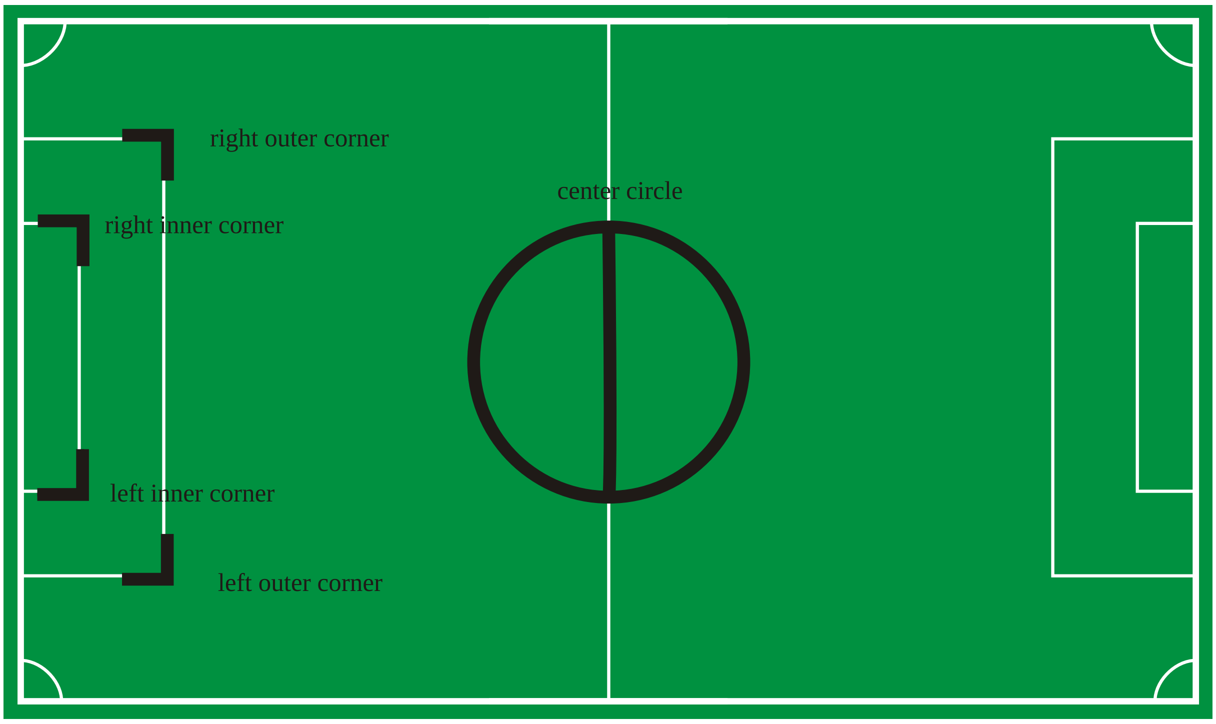


Figure 4.34: The center circle and four different corners are recognized by the system. Although the shapes of the different corners are identical, the system is able to identify the position of a detected corner within the penalty area. Within one side of the playing field, each of the corners represents a unique feature, since their position with respect to the penalty area is simultaneously detected in the recognition process.

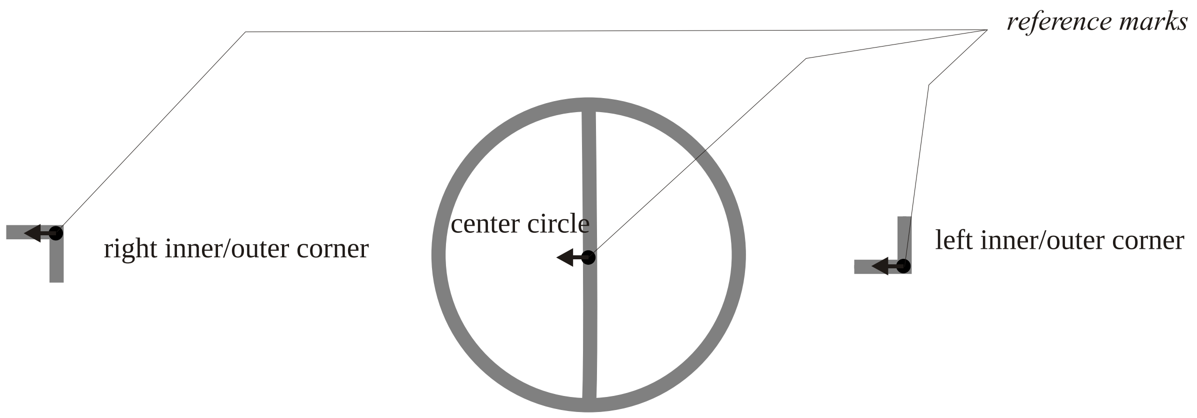


Figure 4.35: This figure shows the position of the reference marks considered in relation to the local geometry of the features.

In particular, the set $\mathbf{L}_f$ is the set of poses of reference marks in the global system. It specifies at which locations the feature f is instantiated in the line model. Figure 4.35 shows the sets $\mathbf{L}_{center\,circle}$ and $\mathbf{L}_{left\,inner\,corner}$. The sets of the other three corners are defined accordingly. When detecting a high-level feature, we actually infer the pose of its

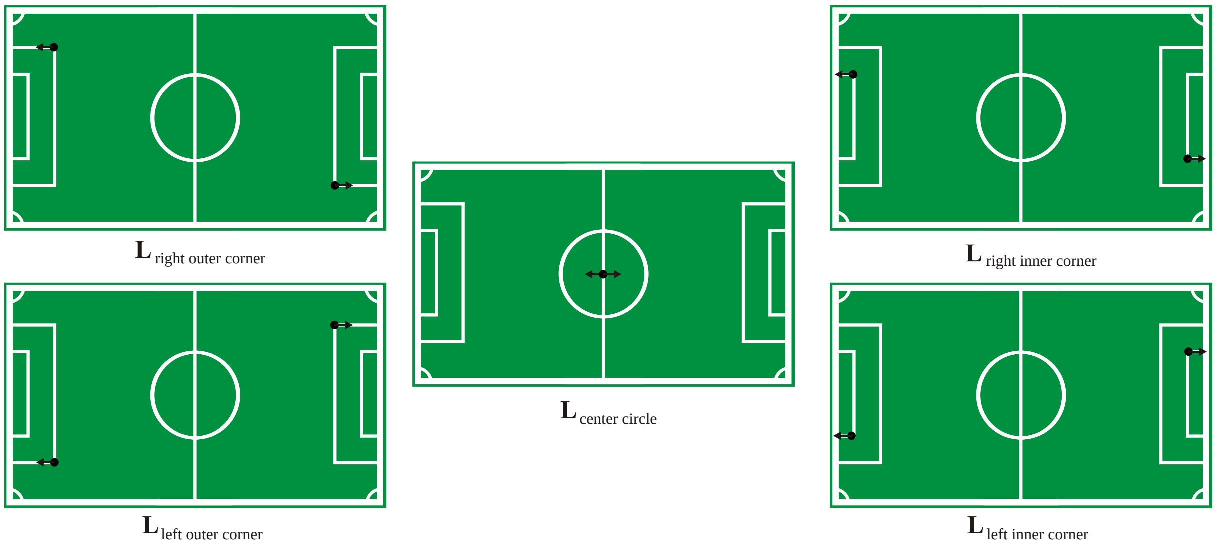


Figure 4.36: The set $\mathbf{L}_f$ of global poses of the reference marks of each type of feature is shown.

reference mark within the extracted field lines after having removed the distortion, that is, within the local coordinate system of the robot. Let us denote $\mathbf{q}_f^r$ as this detected pose of feature f. Then we compare $\mathbf{q}_f^r$ with the poses $\mathbf{p}_i \in \mathbf{L}_f$ and for each combination, we determine a relative transformation $\mathbf{M}_i$ (a movement consisting of a translation and rotation as defined on page 75) of the robot that matches the poses to each other. With $\breve{\mathbf{x}}_{k-1}$ being the last estimate of the robot's pose, consider figure 4.37 for the calculation of this transformation:

$$\mathbf{q}_f = (\breve{\mathbf{x}}_{k-1} \overset{point}{\longrightarrow} \mathbf{q}_f^r) \tag{4.113}$$

$$\mathbf{M}_i = \mathbf{q}_f \overset{point}{\longrightarrow} (p_i \overset{point}{\longleftarrow} (0\;0\;\frac{\pi}{2})^T) \tag{4.114}$$

Here, $\mathbf{q}_f$ is the pose of the reference mark in the global system corresponding to the last estimated robot pose, which is going to be adjusted.

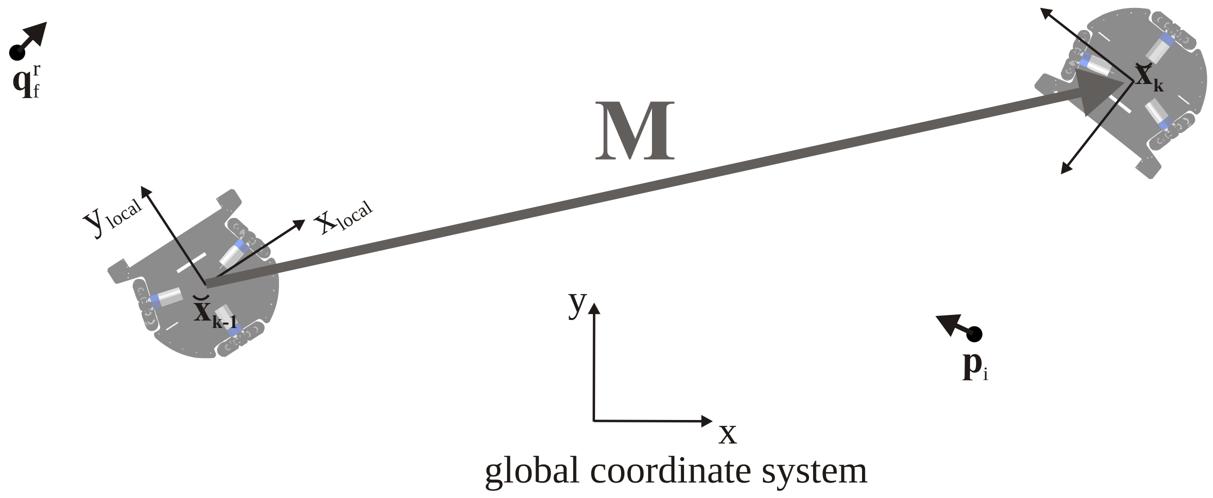


Figure 4.37: A high-level feature f is detected, the reference mark of which is at pose $\mathbf{q}_f^r$ in the local coordinate system corresponding to $\breve{\mathbf{x}}_{k-1}$ of the robot at time step $k-1$. Then pose $\mathbf{p}_i$ of the global model $\mathbf{L}_f$ is considered and our goal is to calculate the movement $\mathbf{M}$ of the robot, such that the perceived feature matches the model feature.

Finally, we take the smallest movement and define the corresponding robot pose to be the measurement vector of layer 3.

$$\mathbf{z}_{k,recognition} = \breve{\mathbf{x}}_{k-1} + \mathbf{M}_{i_{smallest}}. \tag{4.115}$$

Here, $\breve{\mathbf{x}}_{k-1}$ is the last estimate of the robot's pose and the smallest movement $\mathbf{M}_{i_{smallest}}$ is considered to be given in the global coordinate system. In order to determine the smallest movement $\mathbf{M}_{i_{smallest}}$, we have to define a metric on the space of movements. Thus, we have to relate a difference in orientation to a difference in translation. With $\mathbf{M} = \begin{pmatrix} \Delta x & \Delta y & \Delta\phi \end{pmatrix}^T$, we use the following distance function:

$$d(\mathbf{M}) = \Delta x^2 + \Delta y^2 + (\gamma\Delta\phi)^2, \tag{4.116}$$

with $\gamma = \frac{300\text{cm}}{\pi}$ enforcing the influence of the angle given in radians and Δx and Δy specified in centimeters. Thus, a rotation of 180 degrees is weighted as severely as a translation of about 300 centimeters.

This section described, how the robot's pose can be inferred, when detecting a high-level feature. The following sections are dedicated to the actual recognition process.

4.13.4 Smoothing the Lines

After the field lines are extracted from the images, the actual feature recognition starts. It consists of approximately 20 steps, beginning with a smoothing operation, which is required to simplify the detection of local maxima in curvature later.

Smoothing is efficiently done by procedure 4, which is listed in appendix A. Figure 4.38 shows the effect of smoothing.

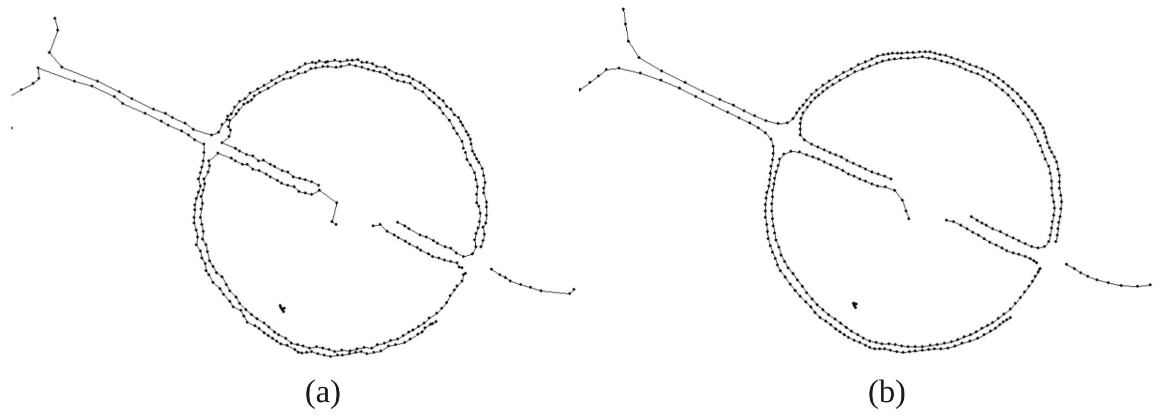


Figure 4.38: The lines (a) before and (b) after smoothing.

4.13.5 Splitting the Lines

The field lines consist of curved and straight lines. For the further detection process, we want to classify the perceived line contours into these two categories. However, as depicted in figure 4.39, the perceived lines often consist of concatenated curved and straight segments. To classify them separately, we have to split the lines at the junctions. The latter coincide with points of local maximum curvature. We retrieve these points by first calculating a curvature measure for each point and then finding the local maxima. Although sophisticated methods exist to calculate curvature measures([6]), we have adopted a very simple approach for the sake of efficiency. For each point, we reference two other points, one in front of and one behind the current point. With these three points, we define two approximate tangent vectors, one reaching from the left to the center point and one from the center to the right point. Finally, we define the curvature at the center point to be the angle by which the first vector has to be rotated to equal the second vector, divided by the distance between the centers of both vectors. In order to be resistent

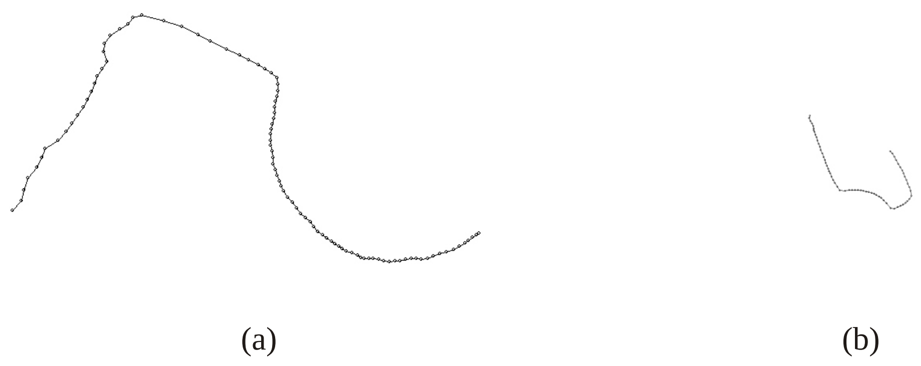

(a) (b)

Figure 4.39: Often a perceived line consists of concatenated straight and curved segments as illustrated near (a) the center circle and (b) a quater circle at the corner of the playing field.

against local noise in the curvature, we choose the enclosing points at some distance to the center point. Since all the points are approximately equally spaced, we can afford to use an index distance instead of a precise geometric distance. That is, for a point $p[i]$ at index i, we choose the enclosing points to be $p[i-w]$ and $p[i+w]$ ($w = 4$ in our implementation). This implies that the curvature can not be calculated for the first and last w points. The principle is illustrated in figure 4.40 and the corresponding pseudo code is given in procedure 5 (see appendix A).

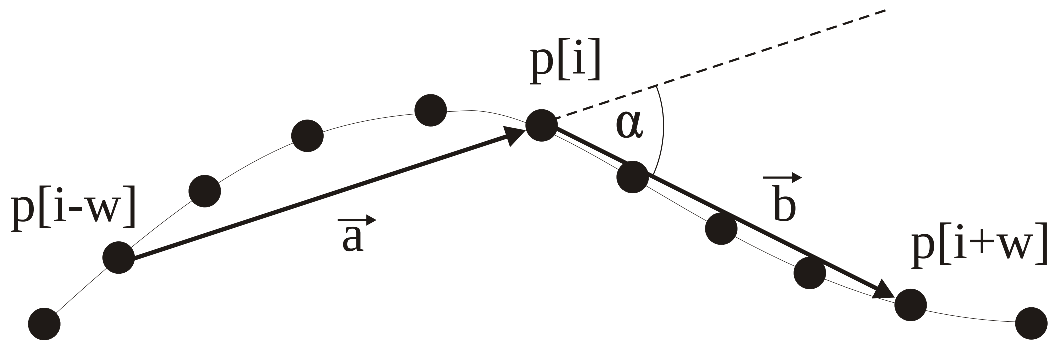


Figure 4.40: The curvature (dimension [1/cm]) at point $p[i]$ is defined to be the angle α by which $\vec{a}$ has to be rotated to equal $\vec{b}$, divided by the distance between the centers of both vectors.

To detect local maxima of the curvature measure, we have adopted the approach given by procedure 6 in appendix A. While traversing the curvature values, we detect intervals of values that exceed a given threshold and within each interval, we determine the index with the maximal value (see figure 4.41). To avoid extrema being too close together, a new interval is opened only if it is at least some given distance from the previous interval.

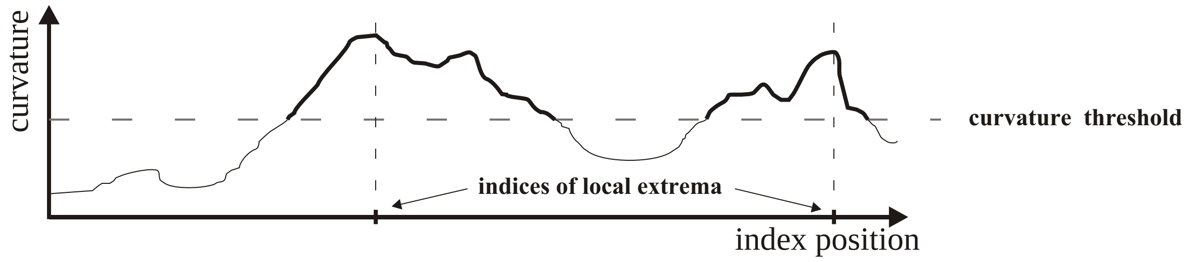


Figure 4.41: The local maxima are found by searching for the highest values within intervals that exceed a given threshold.

Figure 4.42 shows the locations of the split points for various lines. The splitting can efficiently be performed by alternating the entries in the incidence array. If line l is split at k locations, the corresponding entry $I[l]$ is replaced by $k+1$ new entries that specify the start and end points of each segment according to the location of the split points.

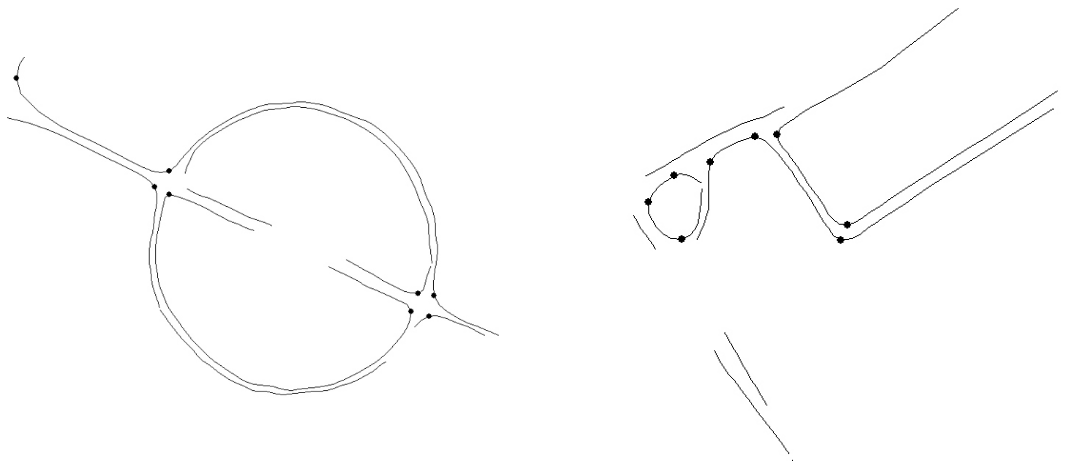


Figure 4.42: Some examples that demonstrate the location of the split points.

4.13.6 Corner Detection

This section describes a processing step which was applied in the system as it was in 2003, where the penalty area consisted only of one rectangle (see page 8). Although the processing step is not included in the actual system it might be interesting for researchers, who want to develop a similar feature detection method.

The split points not only determine the locations to subdivide the lines into parts, but as a byproduct also can yield the first features. We denote a split point at which the curvature is approximately 90° as a *corner feature* and we can distinguish two different types depending on whether the curvature is positive or negative. Remember that the lines are detected by stepping around the boundary of the regions neighboring the field-lines and that the sense of rotation is always the same (see page 63). Thus, in the case of a clockwise rotational direction, if the region has a convex boundary at the point where the corner feature has been detected, the curvature is negative, while it is positive if the boundary is concave. Hence, we denote the respective features as *convex* and *concave* corner features. Both cases are illustrated in figure 4.43. As illustrated in figure

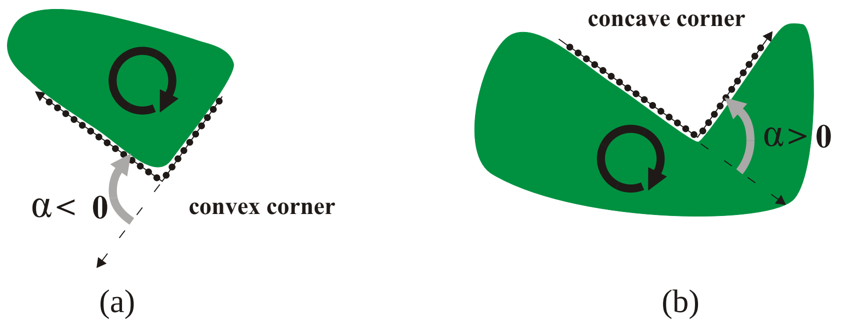


Figure 4.43: Depending on the shape of the region's boundary, a (a) convex or (b) concave corner feature is detected.

4.44, convex corner features occur quite frequently. In contrast, concave corner features occur only at the corners of the penalty area, which are directed inwards into the field. Essentially, concave corner features should also arise at the outer corner markings of the playing field, but the corner posts occlude the lines and no continuous sequence of points is extracted at these locations. Therefore, detecting a concave corner feature gives a very strong hint to the robot's position on the playing field. Figure 4.45 gives an overview of the locations on the playing field where the respective features emerged, underlying a playing field as it was defined by RoboCup in 2003.

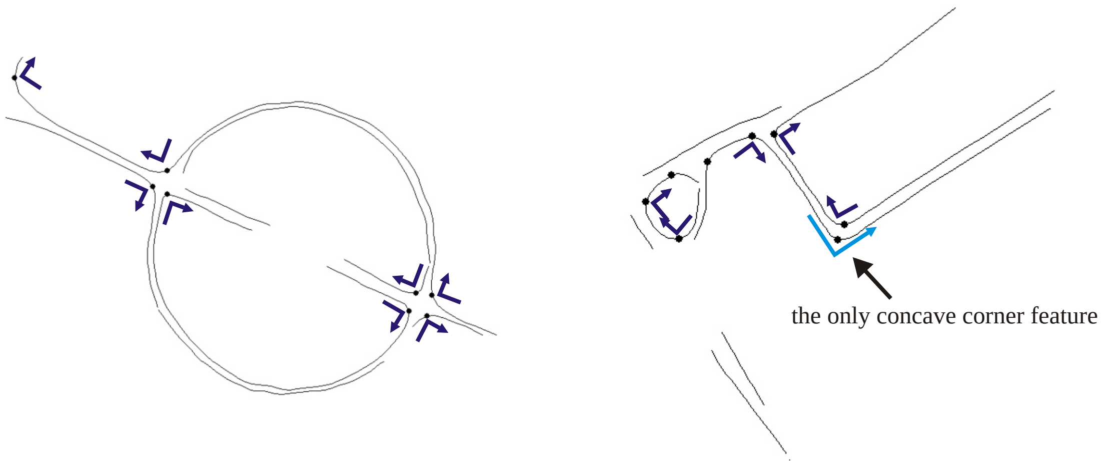


Figure 4.44: Convex corner features are drawn by dark blue corner arrows. The only detected concave corner feature is drawn by a longer light blue arrow.

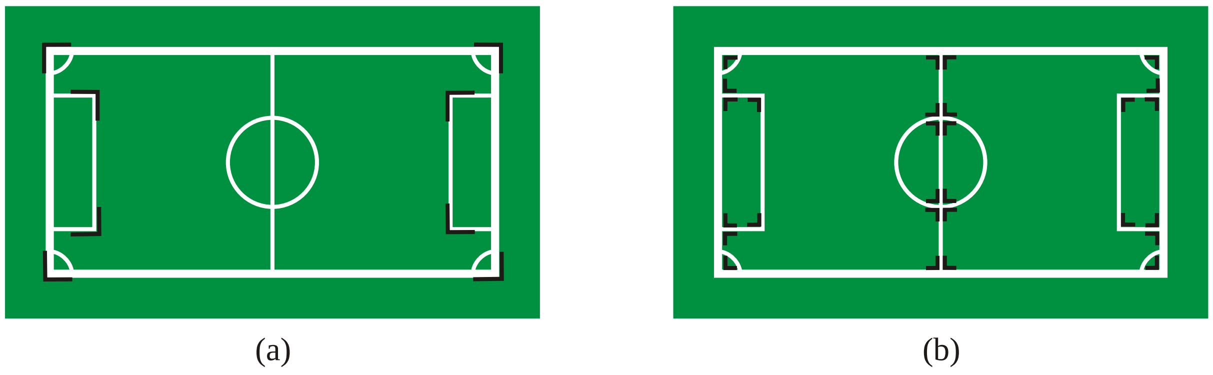


Figure 4.45: The locations of (a) concave and (b) convex corner features, underlying a field line structure as it was given in Padova 2003.

4.13.7 Classification

After the lines have been split, we want to classify each fragment to be either *straight* or *curved* and we employ the following test criterion: Similar to having calculated the curvature measure above, we determine the angle between two vectors, but this time spanned by the first, the middle and the last point of the current line fragment. If the absolute value of the angle exceeds a threshold t_ϕ ($t_\phi = 0.4$ radians in our application), then the line is declared to be curved, otherwise straight. Figure 4.46 shows a typical result and demonstrates that sometimes false classifications occur. The bent line passing along the corner of the penalty area has not been split because its curvature is too smooth and weak. Therefore, instead of having two straight fragments, the line has been classified as *curved*. This is wrong because the field markings of the penalty area do not consist of any curved lines. However, identifying lines as curved is primarily for the detection of the center circle and the corresponding detection process can cope with a small number of wrong classifications. The straight line fragments, the curved parts and the corner features serve as building blocks for more complicated features.

4.13.8 Constructing Arcs and straight lines

Previously, point sequences were classified into straight and curved segments. Next, we construct straight lines and circular arcs from the respective data. A straight line can be approximated by the start and end point of each point sequence. Similarly, one can simply construct an arc using the start, middle and end point by considering the perpendicular bisectors of the two segments. They intersect at the center of the circle including the arc (see figure 4.47).

In order to verify whether the points really have the shape of a circular arc, we approximate the radius r by the mean distance of the points $\mathbf{p_i}$ from the center $\mathbf{m}$.

$$r = \frac{1}{n} \sum_{i=0}^{n-1} \| \mathbf{p_i} - \mathbf{m} \| \tag{4.117}$$

where n is the number of points in the points sequence. Next, we compute the mean distance from this radius. Finally, we relate this measure to the radius in order to allow larger arcs to have a higher variance. That is, the relative deviation σ_{rel} from the radius

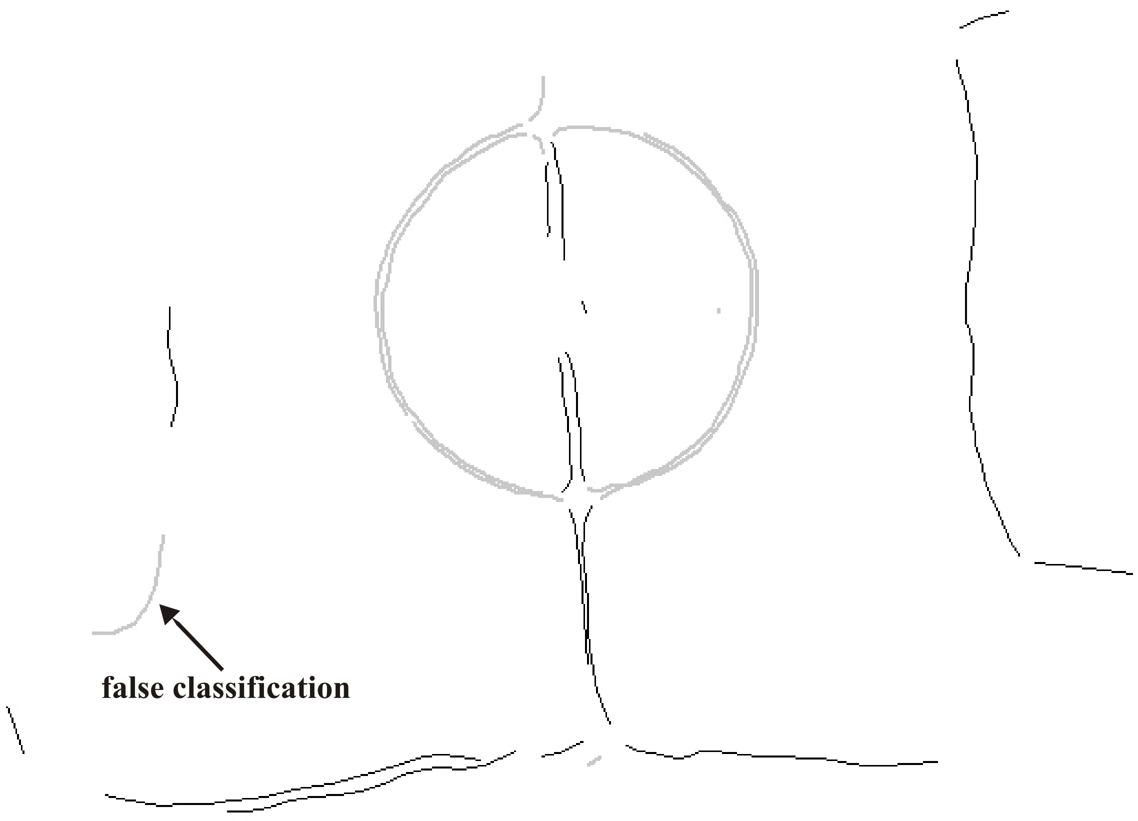


Figure 4.46: Line fragments classified as *curved* are drawn by thick gray strokes, while *straight* fragments are depicted as thin black lines. As illustrated on the left, false classifications can occur. Points of curved sequences are not filled.

r is

$$\sigma_{rel} = \frac{1}{rn} \sum_{i=0}^{n-1} |\| \mathbf{p_i} - \mathbf{m} \| - r|. \tag{4.118}$$

It seems that two loops are required, one for calculating r and one for calculating σ_{rel}, but this is not necessary. In terms of a random variable X being the distance of a point to the center, equation 4.117 is the expectation EX of X and equation 4.118 is the standard deviation of X divided by EX. That is,

$$\sigma_{rel} = \frac{\sqrt{\mathrm{Var} X)}}{EX}. \tag{4.119}$$

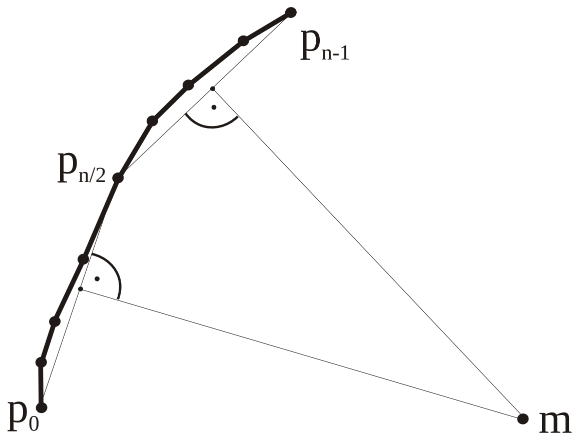


Figure 4.47: The three points p_0, $p_{n/2}$ and p_{n-1} of a point sequence of length n form two line segments. The center point $\mathbf{m}$ of a circular arc is approximated by calculating the intersection point of the perpendicular bisectors of the segments.

Since $\mathrm{Var}X = E[(X - EX)^2] = EX^2 - (EX)^2)$, we have

$$d_{rel} = \frac{\sqrt{EX^2 - (EX)^2}}{EX}. \tag{4.120}$$

Thus, r and σ_{rel} can be computed efficiently within the same loop by calculating EX and $E(X^2)$. In figure 4.48, some examples of arcs with the respective values of σ_{rel} are depicted. We consider arcs for which σ_{rel} exceeds 0.3 as unreliable and discard them.

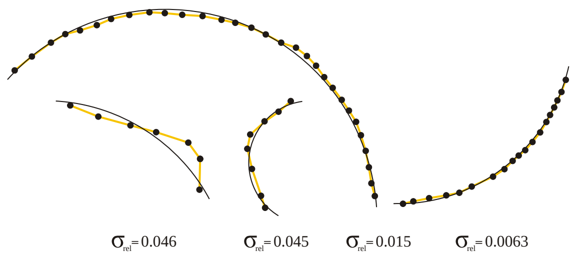
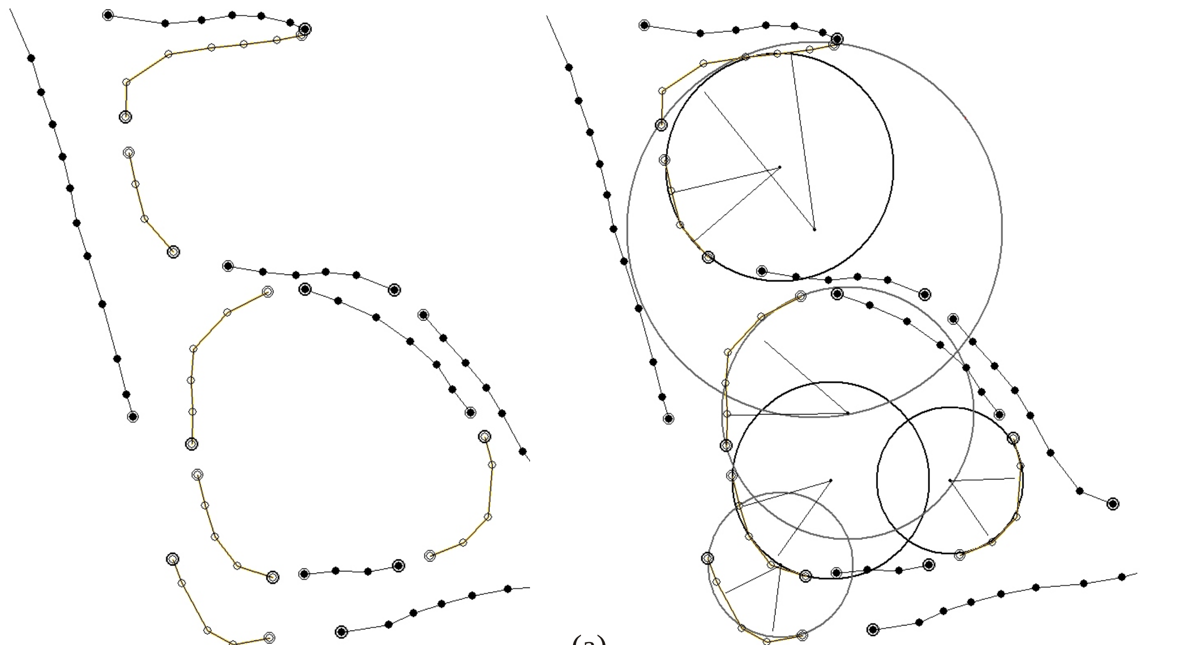


Figure 4.48: Some examples that illustrate the values of σ_{rel} for various curved point sequences. A precise arc with the estimated radius is shown for each point sequence. The closer the shape resembles the arc, the lower is σ_{rel}. Point sequences with $\sigma_{rel} > 0.3$ are discarded.


(a)

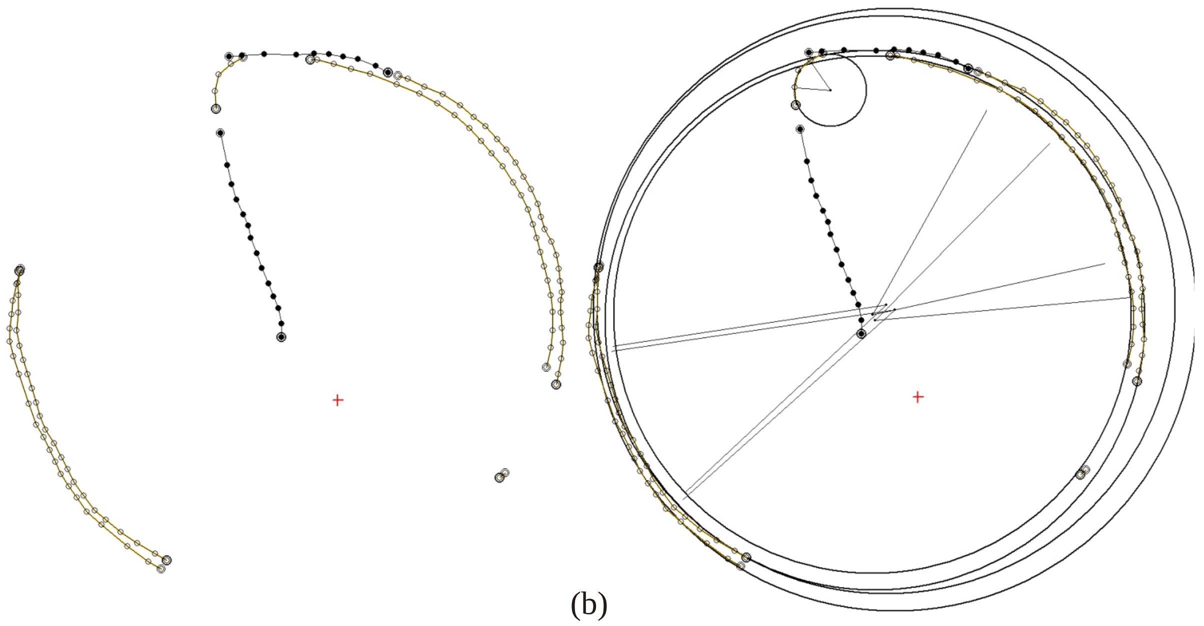

(b)

Figure 4.49: The goal is to detect the center circle. However, it can happen, that arcs are detected which do not originate from a circle. Two typical cases are illustrated. On the left of figure (a) and (b) the original point sequences are depicted. Circular arcs are constructed from each curved point sequence and the corresponding circle which includes the arc is depicted on the right. For gray circles $\sigma_{rel} > 0.3$ and they are discarded. Circles of accepted arcs are painted black. Also, the two perpendicular bisectors that were used to calculate the center point of each arc (see figure 4.47) are depicted by thin straight lines. If circular arcs originate from the same circle, their center points are close together.

4.13.9 Grouping Arcs and Detecting the Center Circle

Although the arc detection discussed above discards curved point sequences that do not have the form of a circular arc, arcs that are not part of the center circle could emerge. Some of these spurious arcs are part of the quater circles, but there are also cases, where arcs emerge from straight lines due to noise effects, detection errors, or a badly calibrated distance function. Figure 4.49 shows some of these cases.

To be robust against such outliers, we want to group arcs in order to determine whether there are several arcs supporting the same circle, or whether a single spurious arc has been detected. Also, grouping allows a more precise detection of the overall circle.

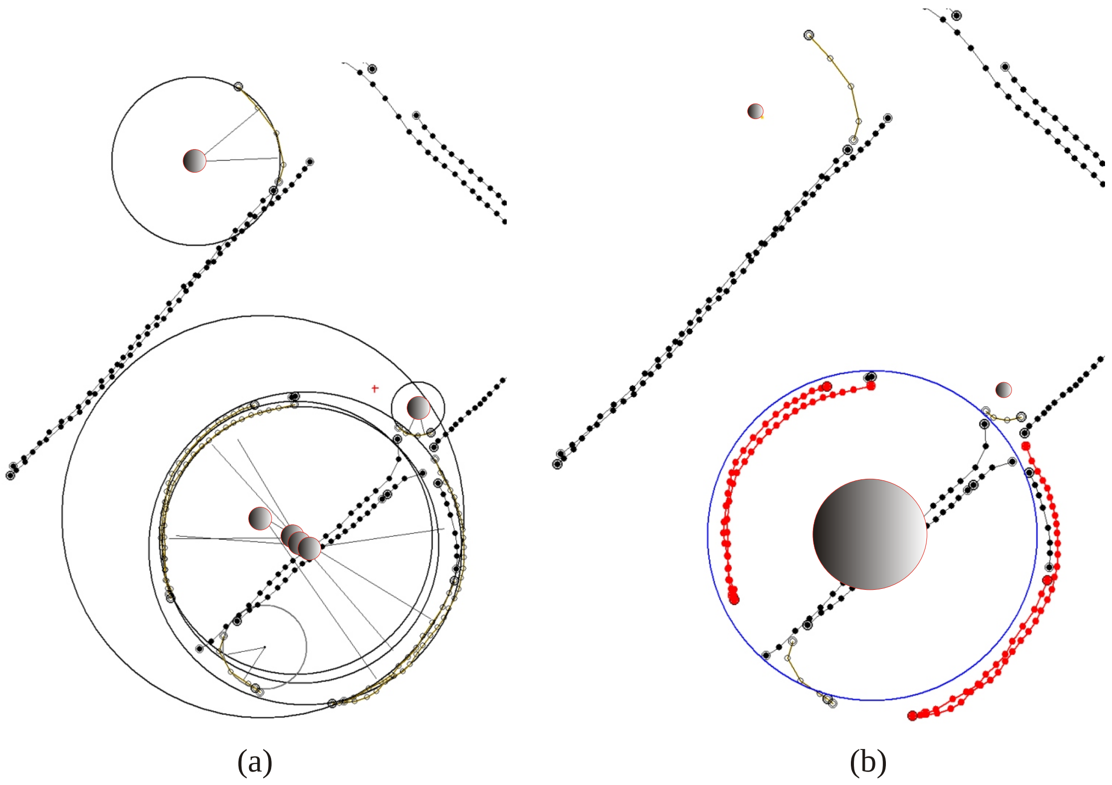


Figure 4.50: In figure (a) the center points of all detected arcs are depicted by small disks filled in with a smooth black-white transition. In figure (b) the resulting clusters are depicted with the area of the disks indicating the weight of each cluster. All point sequences that contributed to the greatest cluster are painted red. A circle is constructed with the center being the center of the main cluster and the radius being the average distance of the red points from the cluster center. However, the circle is just a first approximation and the solution will be refined by an iterative method later.

In order to group the arcs, we observe that the center points of arcs belonging to the same circle should be close together. Therefore, we search for clusters of the center points of the arcs. We will refer to them simply as center points in the following description of the cluster algorithm. Initially, we pose a cluster at the first center point. Then, we traverse the remaining center points and calculate the respective distance to the cluster center. If the distance exceeds a variable threshold $t_{dist} = 0.6r$ where r is the radius of the current arc, we open a new cluster at the respective position. Otherwise, we adapt the cluster center to represent the weighted mean position of all assigned center points. Here, the weights are the lengths of the arcs, which we approximate by the number of points of the corresponding point sequences. When several clusters have been opened, a center point first determines the closest cluster and then verifies whether the distance exceeds t_{dist}. In this way, we obtain a set of clusters and choose the one with the greatest weight. If the weight is greater than a threshold $t_w = 20$, we consider the cluster to be the approximate center of a circle. To determine a first approximation of its radius, we calculate the mean distance of all points of the arcs that have contributed to the cluster. However, as illustrated in figure 4.50 the circle is just an approximation. Up to now, the circle has emerged from point sequences that have been classified as *curved*. However, point sequences that originate from the center circle of the playing field could have been misclassified as *straight* and hence, are not part of the points from which the circle is determined. We want to include these points before refining the solution of the initially found circle. Figure 4.51 shows an example of such lines that are part of the circle, but have been classified as *straight*.

Remember that for classification, the angle between the two vectors spanned by the first, middle, and end point of each point sequence is used. If the absolute value of this angle exceeds a threshold $t_\phi = 0.4$, the point sequence is considered to be curved, otherwise straight. Now, in the presence of misclassifications, one could assume that the threshold $t_\phi = 0.4$ is too high. But when lowering the threshold, point sequences that emerge from straight lines are sometimes classified as curved. In fact, there is no threshold t_ϕ that ensures that all the data is classified correctly. Instead, an initial idea of the underlying structure of a point sequence has to be present in order to classify ambiguous data correctly.

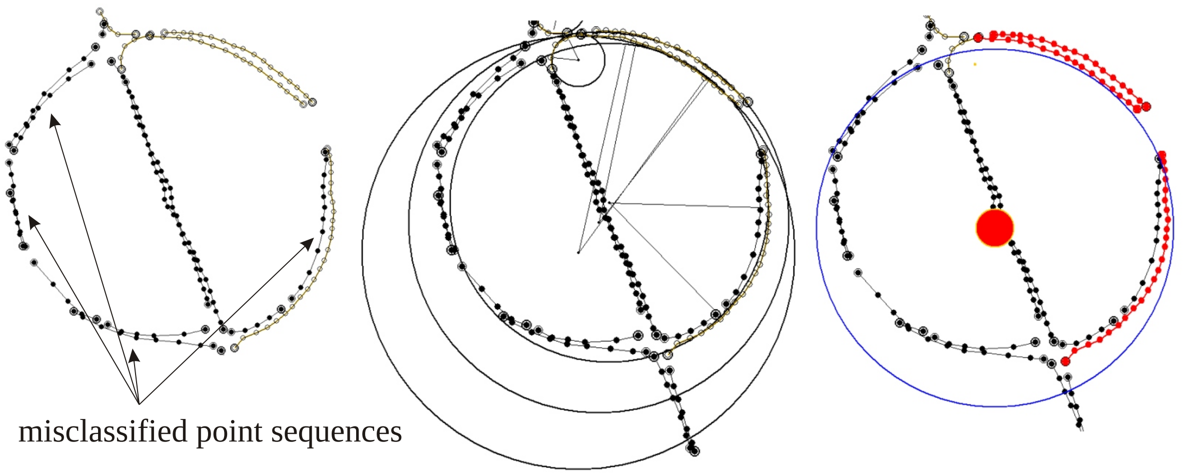


Figure 4.51: This figure shows the effect of misclassifications of point sequences for the detection of the center circle. In (a) several point sequences are classified as *straight*, although they originate from the center circle. Since arcs are only constructed from curved sequences (b), the detected circle is imprecise (c). Thus, before refining the solution, it is important to determine the point sequences which have been misclassified and to include the corresponding points when improving the solution.

4.13.10 Refining the Initial Solution of the Circle

We use the initial hypothesis of the circle to detect and reclassify ambiguous data. We consider all point sequences that have been classified as straight and investigate whether they could have been part of the hypothetical circle. Here, some tolerance is required, since the initial hypothesis is just an approximation of the real circle. Two conditions are verified. First, the distance from the center of the actual "straight" point sequence to the center of the hypothetical circle is compared with the radius of the circle. Second, the direction of the point sequence is compared with the tangent direction of the circle at a corresponding location. If the difference between the measured distance and the radius is more than half of the radius, we discard the point sequence. That is, we do not consider the point sequence to be a misclassified curved sequence that is part of the circle. Similarly, we discard the point sequence if the difference in the direction is too large. In order to describe the conditions more precisely, let $\mathbf{m}$ be the center and r the radius of the initial circle. Furthermore, let $\mathbf{p}$ denote the center of the actual straight point sequence and let $\mathbf{v}$ be the directional vector of the point sequence, which is the vector from the first to the last point of the point sequence. Then we consider the intersection point **s** of the circle with a ray beginning at the center of the circle and passing through **p** (see figure 4.52).

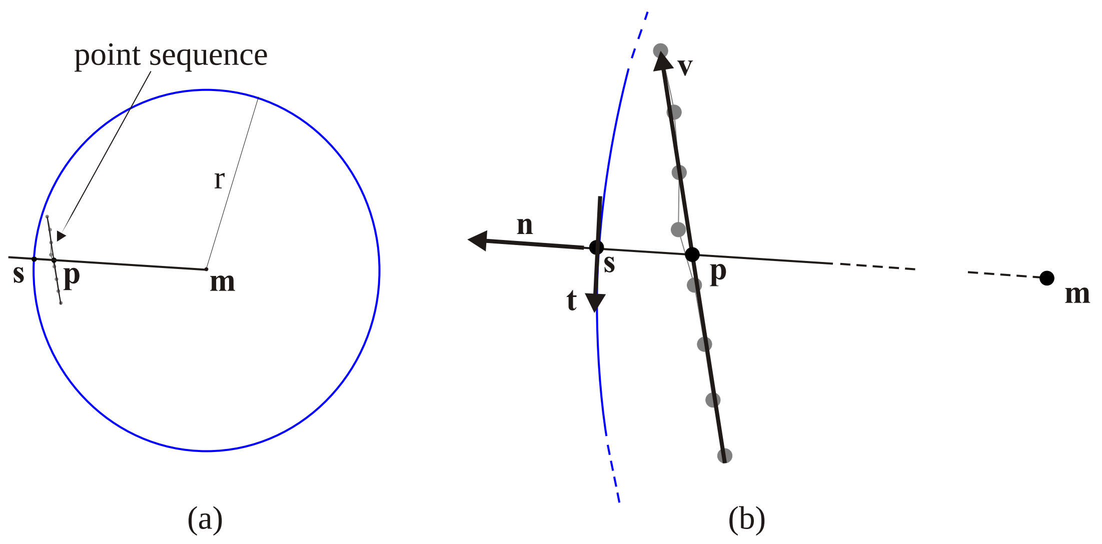


Figure 4.52: This figure illustrates the different symbols used in equations 4.121 to 4.125 to determine whether a point sequence belongs to the circle, although it has been classified as *straight*. This step is important in order to refine the initial solution of the detected circle. The meaning of the symbols is explained in the text. Part (b) shows a close-up of the geometry around the point sequence in part (a).

Formally, **s** is given by

$$\mathbf{s} = \mathbf{m} + r\mathbf{n}, \tag{4.121}$$

where

$$\mathbf{n} = \frac{\mathbf{p} - \mathbf{m}}{||\mathbf{p} - \mathbf{m}||}. \tag{4.122}$$

The tangent direction **t** of the circle at point **s** is obtained by rotating **n** 90 degrees. With $\mathbf{n} = (n_x \; n_y)^T$, we have

$$\mathbf{t} = \begin{pmatrix} -n_y \\ n_x \end{pmatrix}. \tag{4.123}$$

We discard the actual point sequence if

$$||\mathbf{p} - \mathbf{s}|| > 0.5r, \tag{4.124}$$

or, if

$$|\frac{\mathbf{t}^T\mathbf{v}}{||\mathbf{v}||}| > \cos 25^\circ. \tag{4.125}$$

In this way, we detect additional point sequences that also belong to the circle. Thus, we have the points from which the initial circle was constructed plus the points added due to the initial hypothesis of the circle. Next, we will adjust the position and radius of the circle to fit the points.

Here, we use Landau's method [52], a simple method that iteratively adjusts the center and the radius of the circle. We restrict the number of iterations to 4 where the computational costs per iteration are linear to the number of points. Figure 4.53 illustrates a typical example, showing the initial and refined circle.

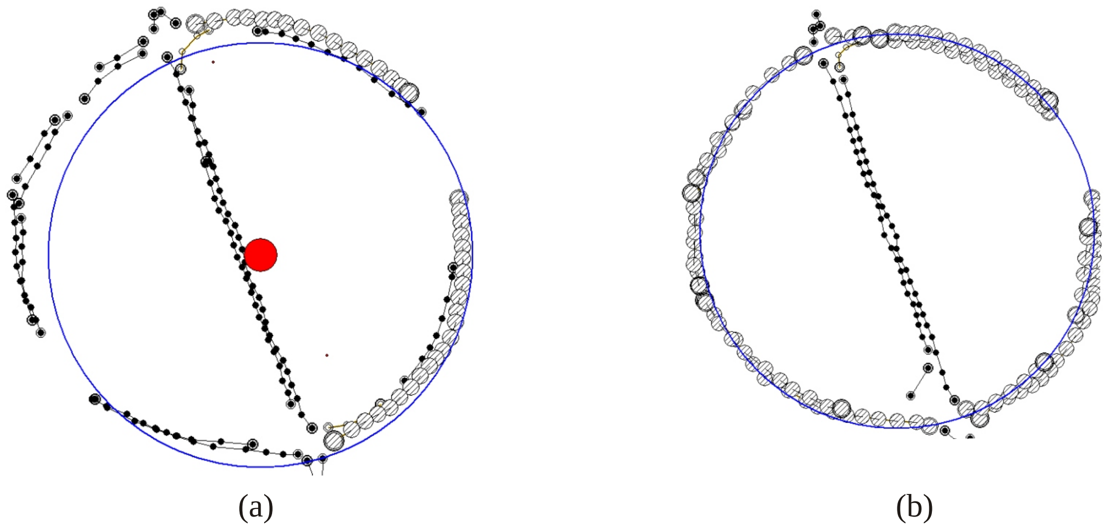


Figure 4.53: The position and radius of the initial circle is refined. Points which are considered to originate from the circle are painted with a diagonal pattern. Initially (a), only the points of curved point sequences are considered to be part of the circle. Thus, the initial circle is not optimal. In (b) additional points have been determined by identifying point sequences close to the initial circle. Then the initial circle is iteratively adjusted to the points by Landau's method [52]. Only few iterations are required (two iterations in this example).

Finally, we search for a straight line which passes roughly through the center of the circle. If such a line can be found, we consider the circle to be the center circle of the playing field.

4.13.11 Determining the Principal Directions

Straight field lines of the playing field are either parallel or perpendicular. We want to determine the orientations in the extracted contours. We will refer to them as *principal directions*. We will determine them with the straight lines constructed previously. Typically, spurious straight lines are present and we apply a clustering algorithm to cope with the outliers. For each line i, we calculate its orientation ϕ_i, normalizing the orientation to lay within $[0, ..., \pi]$. Each line votes for the angles ϕ_i and $\phi_i + \pi/2$. Hence, each line votes for a pair of perpendicular orientations. Here, we apply the same clustering method as for the center circle, but this time working on one-dimensional values. We open a new cluster if the angular difference to the best cluster exceeds 0.3 radians. Otherwise, the cluster center is adjusted, with the weights being the lengths of the contributing lines. Finally, the cluster with the greatest weight determines the first principal direction ψ_0. The second principal direction ψ_1 equals the first, rotated 90 degrees. In the following sections, we will consider two lines through the origin having the directions ψ_0 and ψ_1, respectively. We will refer to these lines as the *principal axes* $\mathbf{a_0}$ and $\mathbf{a_1}$. Figure 4.54 shows two examples.

4.13.12 Discarding Unreliable and Grouping Collinear Lines

Having determined the main axes $\mathbf{a_0}$ and $\mathbf{a_1}$ we consider three types of lines. Those which are perpendicular to $\mathbf{a_0}$, those which are perpendicular to $\mathbf{a_1}$, and those whose orientation differs from both ψ_0 and ψ_1 by more than 0.3 radians. We consider the latter lines unreliable and discard them. The following step is performed for both $\mathbf{a_0}$ and $\mathbf{a_1}$ together with the respective perpendicular lines. Therefore, we will write $\mathbf{a}$ instead of $\mathbf{a_0}$ and $\mathbf{a_1}$ in the following. Let $L = \{l_0, l_1, ..., l_n\}$ denote the set of lines l_i which are perpendicular to $\mathbf{a}$. Furthermore, let $\mathbf{m_i}$ be the midpoint of line l_i. We consider the orthogonal projections of all $\mathbf{m_i}$ onto the axis $\mathbf{a}$. Lines that are collinear will yield close projection points. Since $\mathbf{a}$ passes through the origin, each projection point of $\mathbf{m_i}$ can be represented by a single value t_i, which is the distance of the projected point to the origin. This relationship is illustrated in figure 4.55.

Collinear lines will have the same t_i values. However, since the lines are not precisely collinear, the t_i will differ slightly. Thus, to find groups of collinear lines that are perpendicular to $\mathbf{a}$, we search for clusters of similar t_i values. Again, we apply the same

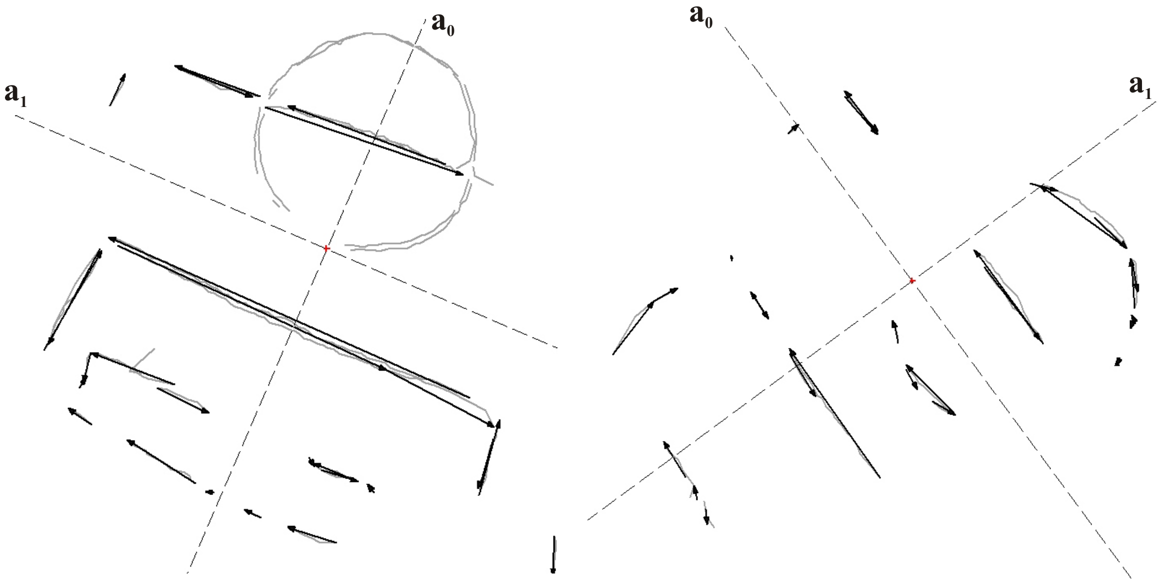
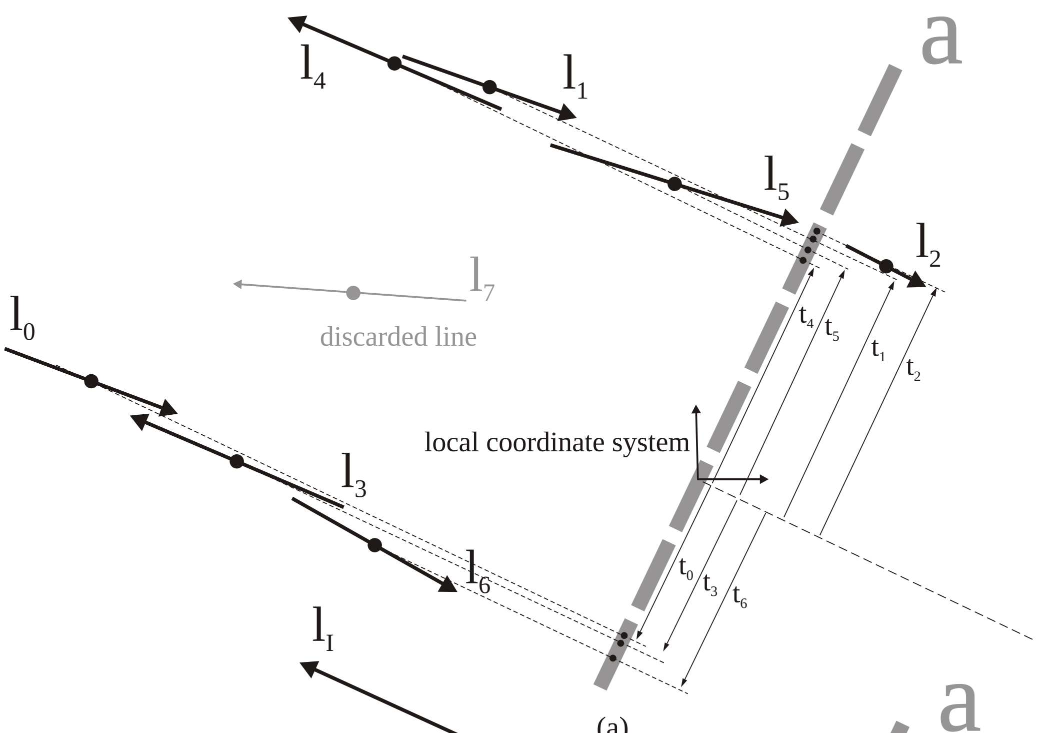
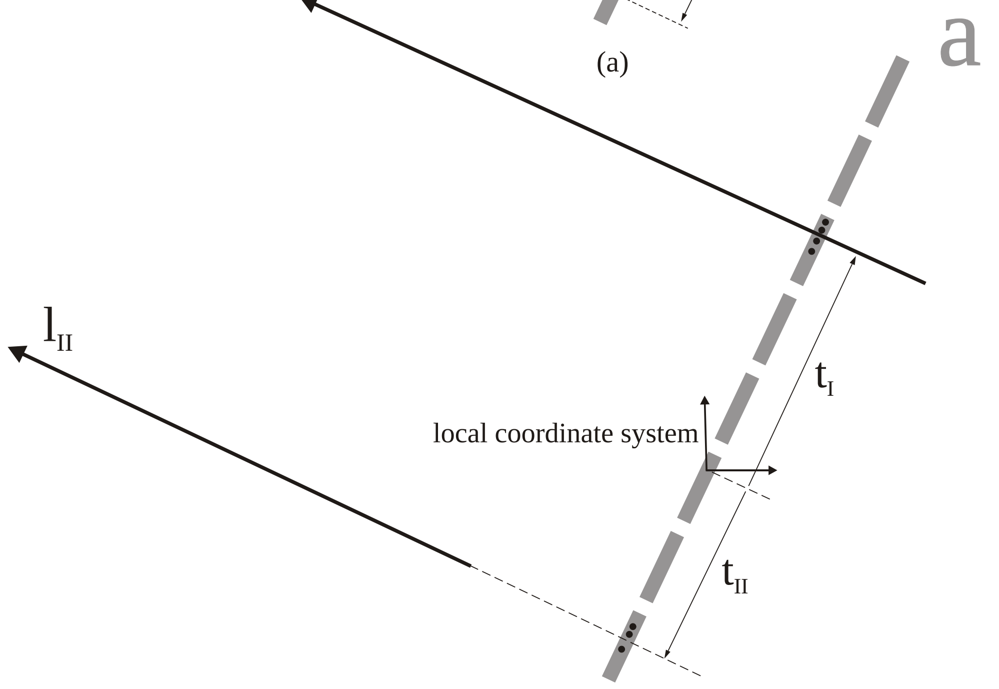


Figure 4.54: Two examples of the determination of principal directions are shown. The original line contours are painted gray. Straight lines are drawn by black arrows and the principal axes found are shown. As can be seen, the method is robust against outliers.

clustering algorithm described for the circle detection and determination of the principal directions. A new cluster is opened if the distance to an existing cluster is greater than 20 centimeters. Each cluster stores the lines that were assigned to it. Thus, each cluster represents a group of collinear lines which are perpendicular to the respective main direction. The lines within each group are replaced by a single line that encompasses the full range of the original lines. Finally, we sort the groups by their one-dimensional cluster centers t_j. Note that the difference $t_j - t_k$ of two groups of parallel lines is simply the distance between the lines. By sorting for t_j, we obtain a topological order of the groups of collinear lines, which will be very useful for the detection of the penalty area and the corresponding corners. Figure 4.56b) illustrates the results of this processing step for two examples.



(a)



(b)

Figure 4.55: In figure (a) straight lines that are approximately perpendicular to the principal axis **a** are considered. Lines whose orientation differs more than 0.3 radians (i.e. line l_7) from a perpendicular line to the axis are discarded. The midpoints of lines l_i are projected onto the principal axis **a**, yielding the values t_i. Clustering the t_i values yields groups of collinear lines at t_I and t_{II}. In figure (b), the lines in each group are replaced by a single line l_I and l_{II}, respectively.

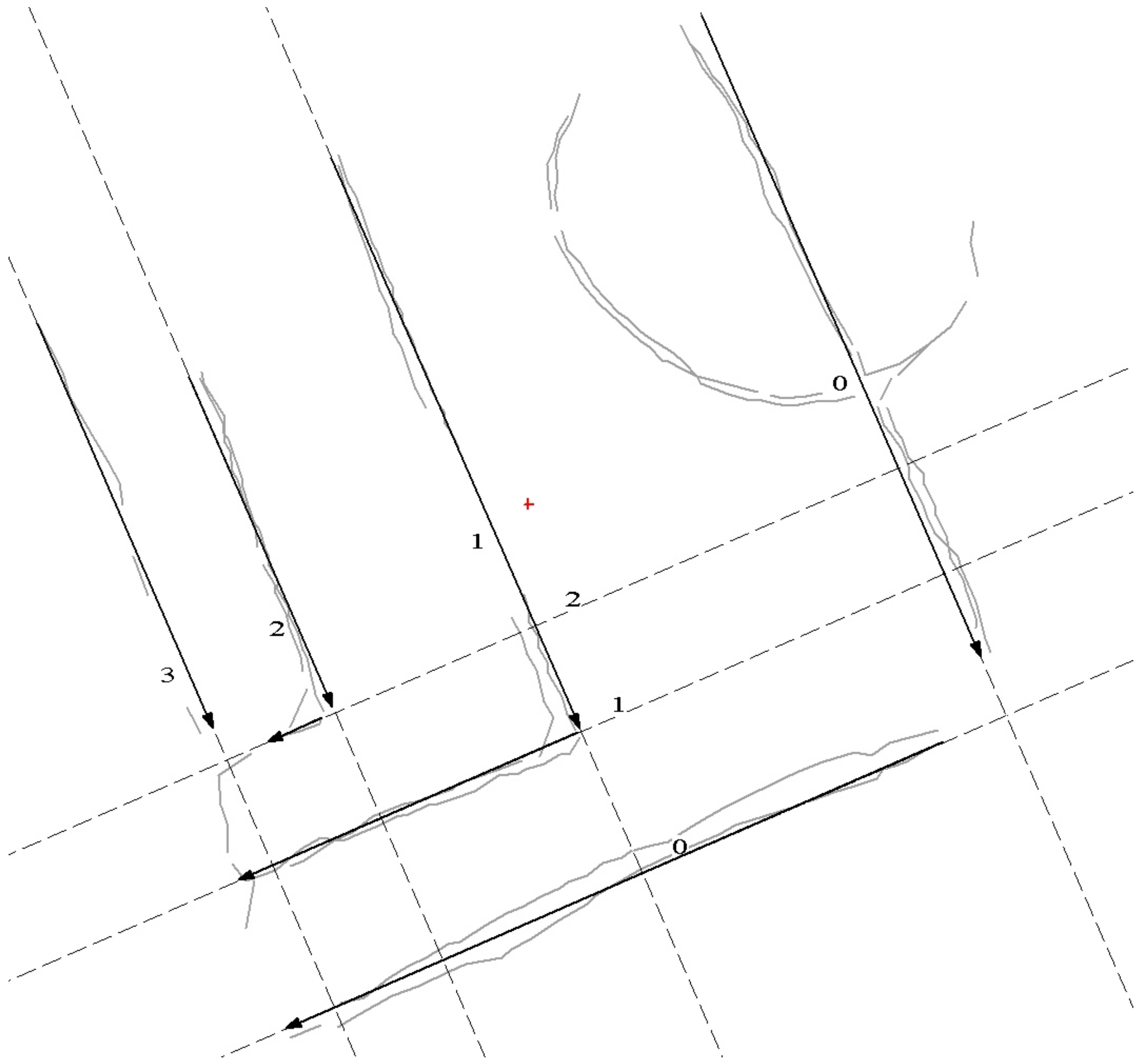


Figure 4.56: The results of grouping straight lines are shown. Straight collinear lines are grouped and joined to form single straight lines that are perpendicular to the principal axes. They are topologically ordered along the respective principal direction. The ordering is shown by the dashed thin lines and the corresponding numbers. The original contours and discarded straight lines are painted gray.

4.13.13 Detecting the Corners of the Penalty Area

The rectangle marking the goal area and the rectangle marking the penalty area produce three lines parallel to the baseline. The lines are spaced at a distance of 50 and 100 centimeters (see figure 4.57). Having grouped and sorted the sets of collinear lines, we can

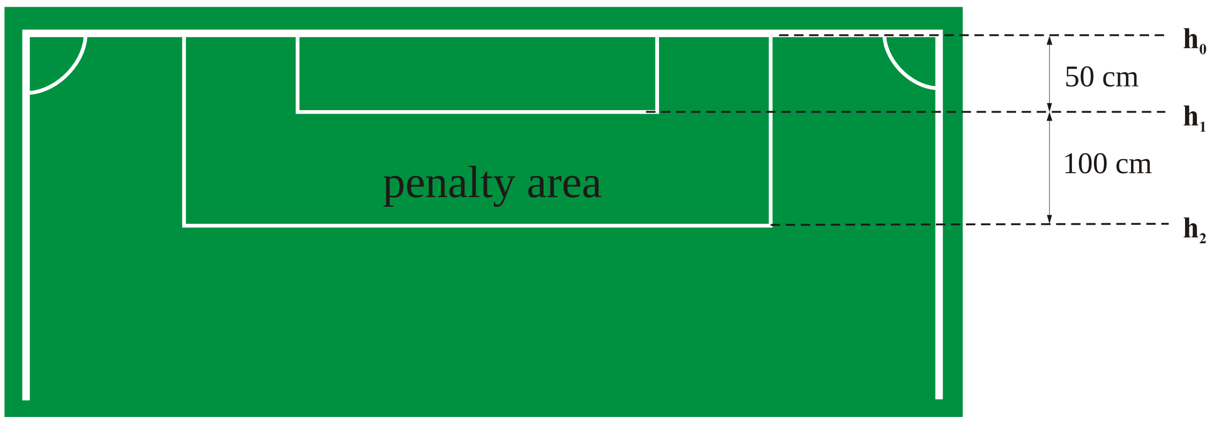


Figure 4.57: At the penalty area the three parallel lines $\mathbf{h_0}$, $\mathbf{h_1}$ and $\mathbf{h_2}$ are spaced at a distance of 50 and 100 centimeters. This structure can be easily recognized in the field lines.

easily detect such a structure. Here, we allow a tolerance of 20 centimeters when verifying the distances. Note that the structure emerges at the start or end of the sequence of sorted collinear line groups, since the lines are the outmost existing lines. Having detected such a structure, the direction towards the goal is now known, since the lines are spaced asymmetrically. Thus, we can distinguish between the left and right sides of the lines.

Some additional constraints are necessary in order to avoid spurious detections. First, if we find three parallel lines that have the given structure in their distances, we calculate the overall length of the structure in the direction of the lines. This length should not exceed the length of the penalty area, which is five meters. We allow a tolerance of 50 centimeters. A second constraint is that no lines perpendicular to the three lines are beyond the goal line.

In order to find the respective corners, we simply verify whether we find perpendicular lines whose endpoints are close to the given lines $\mathbf{h_1}$ and $\mathbf{h_2}$.

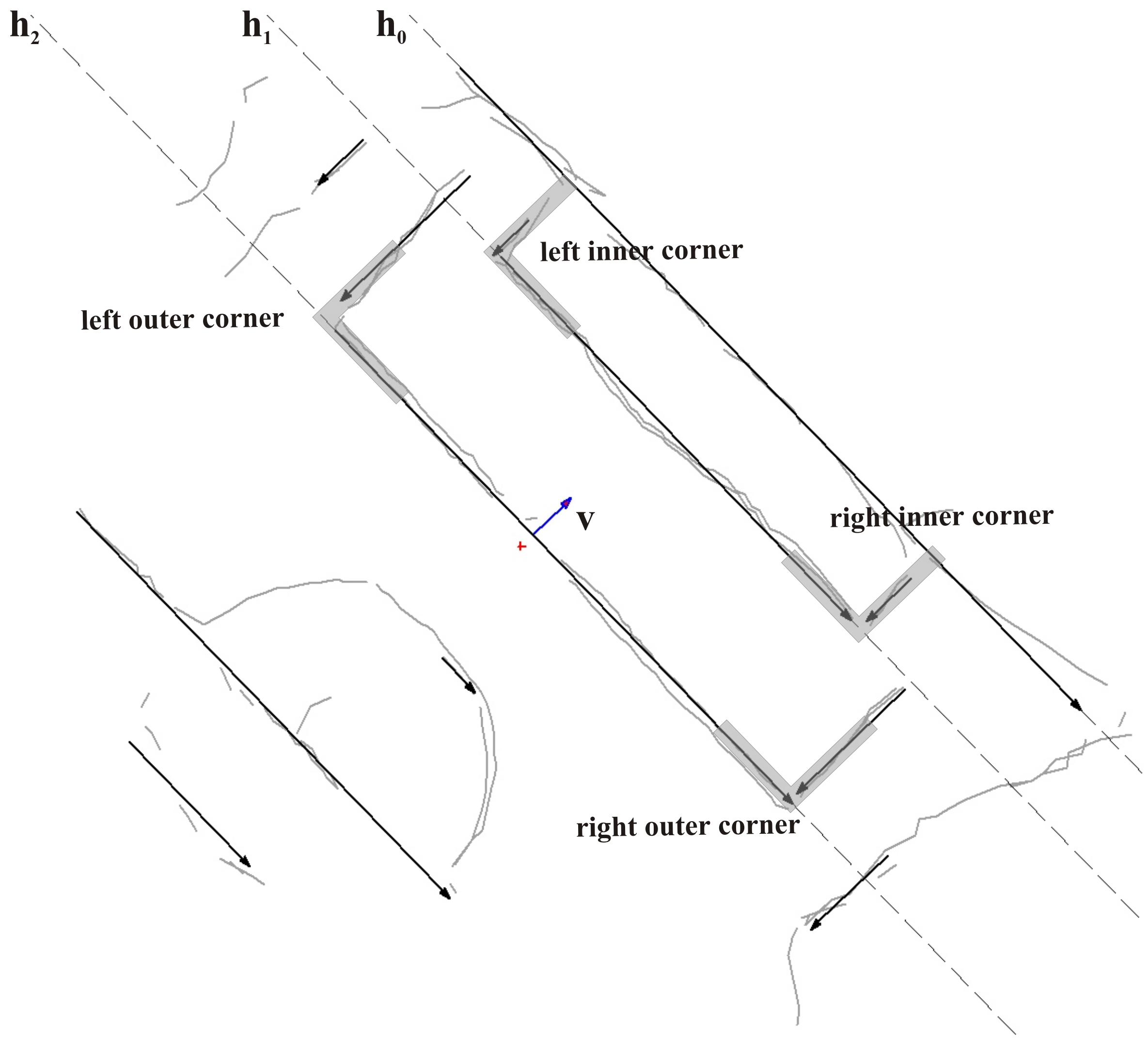


Figure 4.58: The corners are detected by identifying the line structure $\mathbf{h_0}$, $\mathbf{h_1}$, $\mathbf{h_2}$ and searching for perpendicular lines that are close to the endpoints of $\mathbf{h_1}$ and $\mathbf{h_2}$. The vector $\mathbf{v}$ indicates the direction towards the goal.

4.13.14 Results of the Feature Detection

With our omni-directional vision system, the robot cannot see arbitrarily far, at least not with a sufficiently high resolution. Thus, it is interesting to examine the distances up to which the robot can detect features. To verify from which positions on the playing field which feature can be detected, we let the robot execute a specific trajectory, which scans the field as illustrated in figure 4.59. Then we logged all locations where the respective features were detected and marked the corresponding position of the robot as shown in figure 4.59. In this experiment, we removed all obstacles from the field. As can be seen, the robot can detect a high-level feature from all positions. However, the playing field in our laboratory is smaller than the playing field used in the competitions, and using the large playing field a few locations exist where the robot is not able to detect any high-level features. These locations are between the center circle and the corners of the penalty area. However, overall localization is not affected by this because layer 1 and layer 2 are able to keep track of the localization without recognizing features.

Next, we consider how feature detection is influenced by occlusion. The most critical part are the corners of the penalty area. If an obstacle is directly on the corner, the corner is not detected. This is the reason why there are no markings in figure 4.59 at the locations of the corners. The robot itself occludes them while executing the trajectory. It might be possible to adjust the detection of the corners in a way that even the case of direct occlusion can be compensated. To do so, one must infer the corner position by calculating the intersection of two straight lines. However, the danger is that spurious corners arise from straight line segments. Thus, we decided to go for a more conservative detection algorithm that produces no false positives at the cost of sometimes not detecting a feature. Since there exist four different corners at the penalty area, it is extremely unlikely that all of them are occluded at the same time. Even if all four corners are occluded, the overall localization method is not affected, since the bottom two layers maintain localization without feature recognition, once they obtain a correct initial start position.

The detection of the center circle is extremely robust against occlusion. Even if only small parts of the circle are visible, the circle can be detected in most cases. Figure 4.60 illustrates different scenarios with obstacles occluding the center circle and shows under which circumstances the circle can be detected.

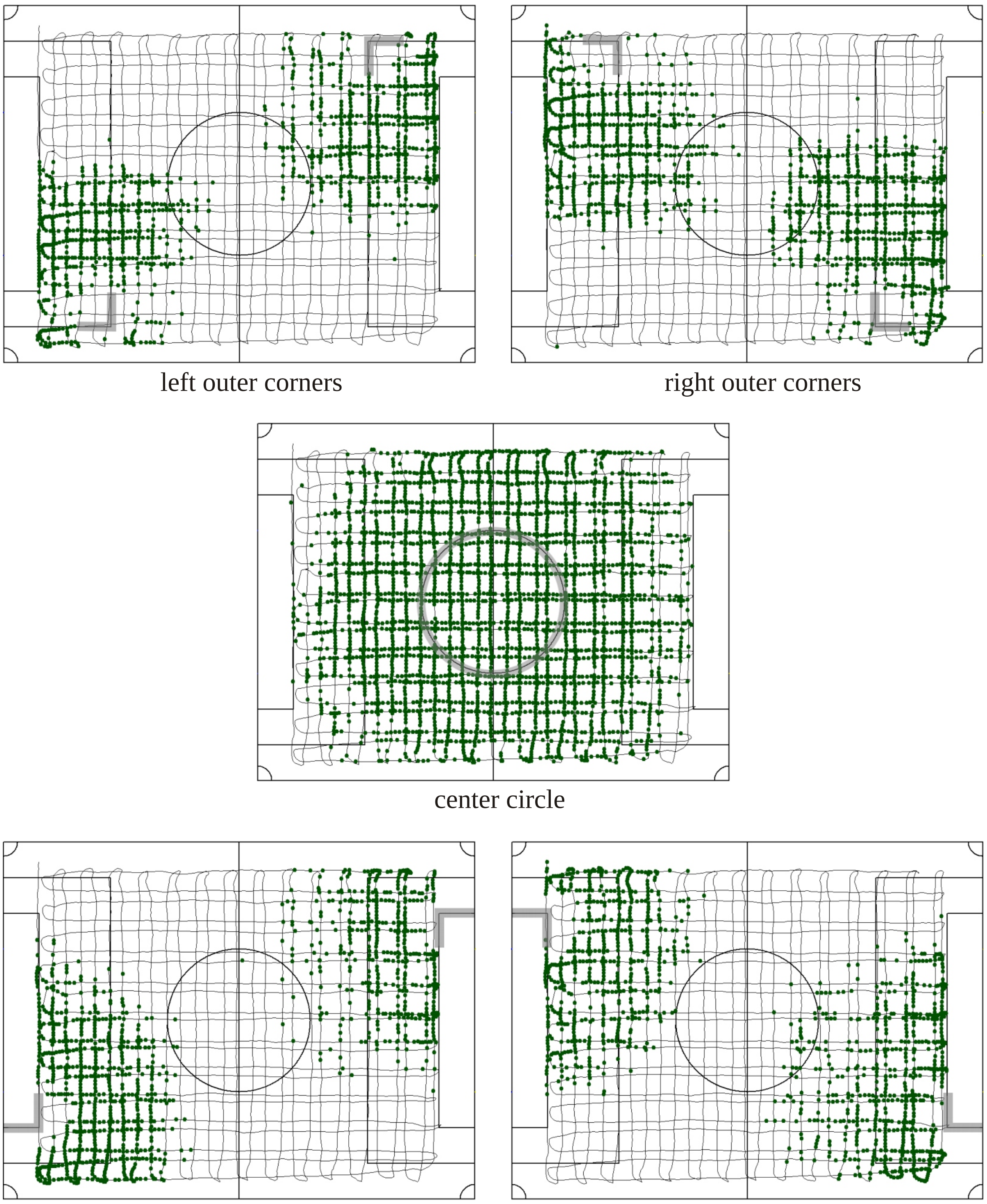


Figure 4.59: Each figure shows the robot executing a trajectory, which scans the field, with the respective positions marked from which a feature was detected.

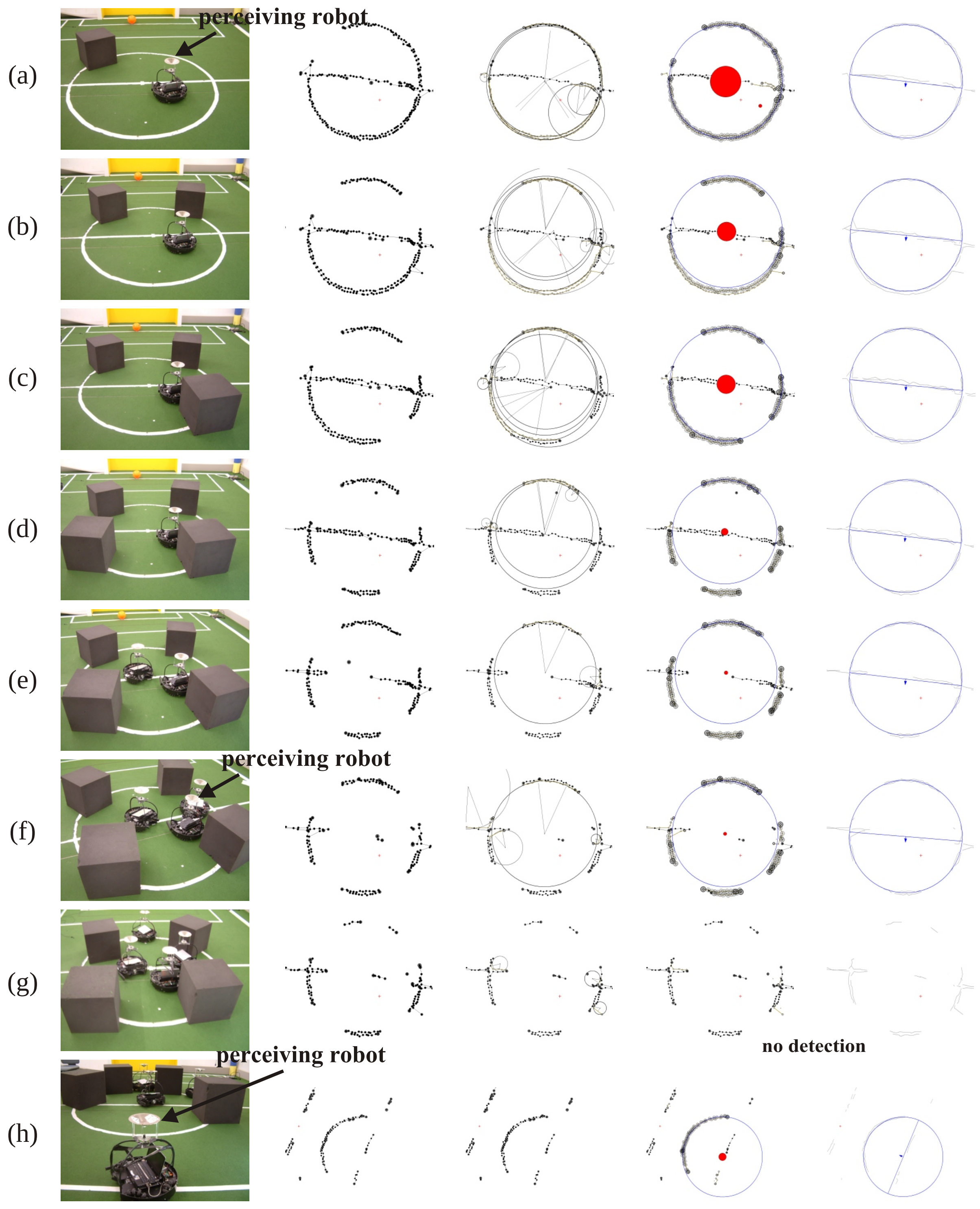


Figure 4.60: The center circle is successively occluded by an increasing number of obstacles and the processing steps for the center circle detection are illustrated (original contours, arc construction, extension, detection). The robot was unable to detect the center circle only in (g).

4.14 Results of the Overall Localization

In order to test the influence of different environments on localization we examined two different playing fields. The first had a green carpet and artificial lighting, while the second had a reflective linoleum floor with natural light shining from one side through an array of windows. However, since the reflections on the floor where almost white in the images, we reduced the influence of natural light by adding artificial light from above and using venetian blinds, although the blinds were not completely shut.

On both playing fields, we let the robot automatically move on a trajectory forming an eight (see figure 4.61). The robot did not loose its position on either field. After 10-20 minutes we stopped the experiment. Moreover, when the robot was manually transferred to an unknown position, the robot immediately found its correct position after perceiving a feature. The maximum positional error while driving was about 10 cm.

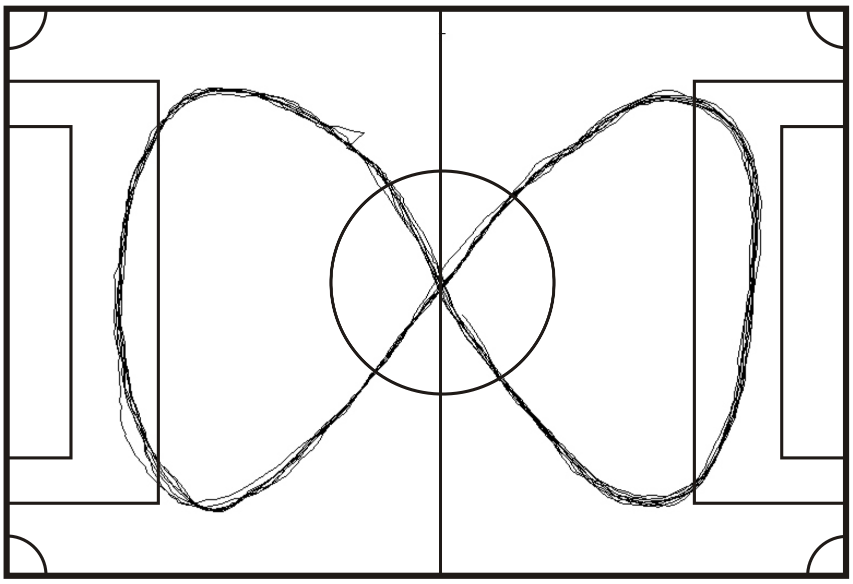


Figure 4.61: The path shows a robot moving on the field along a predefined figure. As can be seen, the average deviation is approximately 10 cm for a robot driving 0.8 m/s.

While the robot moved, we logged all extracted line contours in a file and later manually verified the feature recognition for all frames. Not a single false positive was detected on either field.

Finally, the system was employed during our participation at the world championships in 2004, Lisbon. Here, the playing field had a size of 12 × 8 meters and we played more than 10 games using up to 6 robots. During a game, localization is more difficult, since opponents and team robots are moving and continuously occlude different parts of the playing field. However, localization worked very well and we could not observe any failures. In particular, when robots had to be taken out of the field due to hardware problems and were manually placed at a new location, the feature detection method proved to be valuable for initial localization. In all cases, the robots where able to initially localize after a very short time, typically below a second.