Rheinische
Friedrich-Wilhelms-
Universität Bonn

Institute for Computer Science
Department VI
Autonomous Intelligent Systems

Rheinische
Friedrich-Wilhelms-Universität Bonn

Master thesis

Utilizing the Features of Field for Efficient Soccer Robot 6D Pose Localization

Author:
Yongfeng Zhong

First Examiner:
Prof. Dr. Sven Behnke

Second Examiner:
PD Dr. Volker Steinhage

Advisors:
Prof. Dr. Sven Behnke

Hafez Farazi

Submitted: March 6, 2016

Declaration of Authorship

I declare that the work presented here is original and the result of my own investigations. Formulations and ideas taken from other sources are cited as such. It has not been submitted, either in part or whole, for a degree at this or any other university.

Location, Date

Signature

Dedicated to my father ***Xingwan Zhong****, my mother* ***Xinglian Li****, and my beloved husband* ***Yucheng Jin****. Thank you for your unconditional and inconsistent supports to my studies. Thank you to give me an opportunity and guidance to prove and improve myself in every step of my life. I feel very lucky to grow up in such a happy family and pursue my master degree at one of the prestigious universities in Germany.*

Acknowledgements

I wish to express my sincere thanks to my advisor Hafez Farazi for his enormous help and motivation. It was quite an educational experience working with him. He was very patient and guided me throughout my thesis work. It would be very difficult without his support.
I further would like to express my gratitude to my supervisor Prof. Dr. Sven Behnke, for providing me a precious opportunity to work in his research group and an encouraging research environment in his Lab.
I also wish to thank Philipp Allgeuer, Nicola Krombach, and my colleagues who help me with my questions and doubts from time to time.

Abstract

For an autonomous robot, knowing its pose and the current state of the environment is essential, as it represents a basis of planning and reactive behavior. In case of a soccer playing robot in the RoboCup Humanoid league competition, detecting field features efficiently so as autonomously to achieve and maintain an accurate estimate of its pose on the field is nontrivial. Most current approaches only optimize a three degrees of freedom (3-DoF) (x, y, yaw) of robot pose using visual perception and dead reckoning (DR) data while retrieving another 3-DoF $(z, roll, pitch)$ of robot pose from IMU and Kinematic model. Note that the DR and IMU data are not always reliable, which may reduce the accuracy of the resulting estimate. In addition, new rules in RoboCup are creating new difficulties for solely color-based detection strategies.

To address these challenges, this thesis proposes a model-based visual tracking system that is able to track 6-DoF camera pose on the soccer field. First, we introduce a kernel-based line detection method for extracting observations from the input images. Then we describe a method for creating a set of 2D-to-3D point correspondences by associating the observations with the projected field model. As a result, we can solve the Perspective-n-Point (PnP) problem to find the estimated 6-DoF camera pose. Moreover, use of the RANSAC schema makes the optimization approach robust to outliers. Finally, tracking accuracy can be further increased by applying the Kalman filter for state fusion.

Experimental results show four main advantages of the proposed system in comparison to previous work: 1) the line detection method is less sensitive to light condition changes than the color thresholding-based approaches; 2) data registration accuracy is improved by identifying orientations of nodes and clustering the line segments; 3) the system is comparable and in some cases better than the traditional 3DoF pose localization approaches in terms of visual tracking accuracy; 4) the system is less vulnerable than traditional approaches to the error from DR or IMU data.

Contents

List of Figures

List of Algorithms

1 Introduction

1.1 Motivation

The goal of the international RoboCup committee is to develop a team of humanoid robots that is able to win against the official human World Soccer Champion team until 2050. Due to this purpose, the rules for soccer game are updated every year to make the playing situation progressively approaching the real human soccer competition environment. In order to complete this challenging goal, various of new technologies should be studied and developed.

Detecting field lines comes to the very first important step in vision perception, because field lines are main source of information in soccer robot localization. According to the changes of the Humanoid league rules, the soccer field is built with artificial grass, rather than carpet which was used before. Thus the field lines painted on the grass are no longer clear white. Moreover, the ball is specified to be at least 50% white, which makes it harder to differentiate whether it is a ball or a line segment. Due to these changes, the solely color-based detection method (Laue et al., 2009) that were extensively used in previous years is no longer sufficient for the current environment. Therefore, in order to address the new challenge, it is crucial to develop a new visual perception system that can detect the brighter line segments relative to their neighbor.

In addition, the more severe competition in RobotCup calls for higher accuracy in localization. However, for a 6-DoF pose $(x, y, z, \gamma, \beta, \alpha)^T$ where (x, y, z) denotes the position of the robot on the field; and (γ, β, α) describe the orientation in Euler angles of the robot, most of the current visual localization approaches only optimize three parameters $(x, y, \alpha)^T$ of robot pose while they retrieve the other three parameters $(z, \gamma, \beta)^T$ from IMU and dead reckoning (DR) data, from which the error may influence the resulting localization accuracy. Such approaches require the robot equipped with well calibrated IMU, which it is not always realistic to do so. Therefore, tracking the whole 6D camera pose in is essential for reducing the dependence of the system on DR, which may improve the tracking accuracy.

1.2 Contribution

In this thesis work, we present a visual tracking system which utilizes the field lines to track the 6D camera pose of a soccer robot. The main working steps of our vision tracking system are described as follows: First, we detect the boundary of the field and identifies the obstacles inside the field boundary, then by constructing line detectors, we find the high response pixels. Second, a purified skeleton is formed by using the local optimal values. Third, from the purified skeleton, a node graph is constructed, which is clustered to different line clusters afterwards. Fourth, we set up a set of 2D-to-3D point correspondences by associating the observations with the projected field model, followed by solving the Perspective-n-Point (PnP) problem to find the optimal camera pose. We implement three algorithms for optimization, hill climbing, EPnP, and, EPnP+RANSAC. Finally, we employ the Kalman filter for state fusion.

Based on the working steps of our system, we therefore can outline the research contributions of this thesis as follows:

- Proposing a kernel-based field line detection method that is less sensitive to light condition changes than thresholding-based method.
- Proposing a new approach for constructing node graph that represents the field line structure.
- Improving the data registration accuracy and robustness by considering the node orientations and clusters.
- Three different optimization approaches are evaluated and the most suitable one for our camera pose tracking task is chosen.
- Tracking the 6-DoF camera pose, thus reducing the dependence of our system on other sensor messages such as IMU data.

2 Related work

Vision-base tracking system proposed by this paper can be described in three steps. First, detecting line segments and construct a node graph that represents lines; second, associating the detections and model lines with high accuracy; third, using the resulting correspondences to find an optimal estimated pose of the camera, and keeping track of the pose even when there are some noise measurements. We discuss the related work according to these three aspects.

2.1 Methods for Object Detection

Most of the localization systems in RobotCup make use of the field lines. Many previous work has presented some approaches for extracting candidate pixels from field lines. One of the most popular approaches (Chiang et al., 2010; Gevers and Smeulders, 1999; Laue et al., 2009; Strasdat et al., 2007; Yang et al., 2012) is to use a color threshold to discriminate between field-line and green carpet. Therefore, individual pixels are classified based on pre-calibrated look up table. Detecting filed lines by thresholding has high efficiency in processing; however, it is not robust with regard to changing illumination, simply because the look up table needs to be calibrated every time when light condition changes. Even under the constant lighting conditions, if the robot looks to different directions, due to the reflection of light, a valid threshold may not work any more. The current environment of RoboCup competition has several changes, for example, the field is built with artificial grass and the lines painted on the grass are no longer clear white. Therefore, the changes of the field ground make it extremely difficult to extract effective lines candidate pixels by this color-table-based object detection approach. Röfer (2008) improves the robustness by using ambiguous color classes to resolve the ambiguities. The ambiguous color classes are resolved based on their unambiguous neighbors in the image. Härtl et al. (2014) propose a new object detection system where objects are found based on color similarities. The detection rate of their system has substantial improvement by changing illumination.

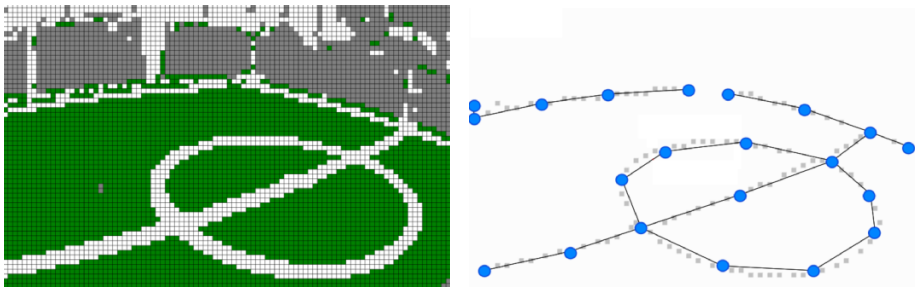
By contrast, numerous works (Canny, 1986) employ edge detection (Canny, 1986) to improve object detection. Instead of finding white pixels by thresholding,

Figure 2.1: Output of the probabilistic Hough line detection on the extracted edges(Farazi, Allgeuer, and Behnke, 2015)

this approach detects candidate line pixels by first applying Edge Detector to the images for detecting spatial changes in brightness (Farazi, Allgeuer, and Behnke, 2015). To detect the edges, different detectors, such as the Sobel Edge Detector (Yu-qian et al., 2006), Gabor filter (Mehrotra et al., 1992) might be applied. This method is quite robust to changes in lighting conditions, so there is no need to tune parameters for the changing lighting conditions. However, convolving by applying filters to a whole image is an expensive operation. Moreover, when edge detection is applied to those lines the result shows edges at both sides of an actual line. Those pixels having high response are not the center of the lines but the edges of the lines. Therefore, detecting lines from the edges input may not be able to get a line that center between two close edges.

After getting line candidate pixels, it is quite common to use a probabilistic Hough line detector to extracting lines from those pixels (Baist et al., 2005; Farazi, Allgeuer, and Behnke, 2015; Strasdat et al., 2007). However, it is hard to detect continuous line segments by this approach because Hough Transformation tend to output many small line pieces on each actual line (Figure 2.1). This problem can be addressed by using a proper line merging method (Farazi, Allgeuer, and Behnke, 2015).

Another approach for finding line segments is to find a node graph that repre-



(a) (b)

Figure 2.2: Node graph representation of field lines (a) subsampled image of pixels classified as "white" and "green"; (b) key node structure graph (Schulz and Behnke, 2012).

sents the geometry structure of the field (Figure 2.2). The graph structure has advantageous in verifying linear components and crossings. A typical method for constructing graph is proposed by Schulz and Behnke (2012). This method performs in sub-sampled images ($\frac{1}{8}$ size of the original images), thus, with proper smooth, it is able to reduce the line width approximately to one pixel when skeletonization. However, finding skeleton in the initial images is much more difficult because it tends to have multiple local optimal value while cutting through a line. Therefore, for using the method proposed by Schulz and Behnke (2012) for constructing node graph in the initial size images, some changes on finding skeleton pixels are needed.

Unlike above mentioned methods, this thesis work proposes a new approach that can detect lines more robustly. In particular, we enhance the responses of those relative brighter pixels by applying four different line filters on the images. Each filter is supposed to detect lines in a particular orientation. The local optimum pixels of the previous output are line candidate pixels, from which a purified skeleton is found. Finally, a node graph is constructed from the purified skeleton. Compared to previous approaches, our approach position the lines with higher accuracy.

2.2 Methods for Feature Matching

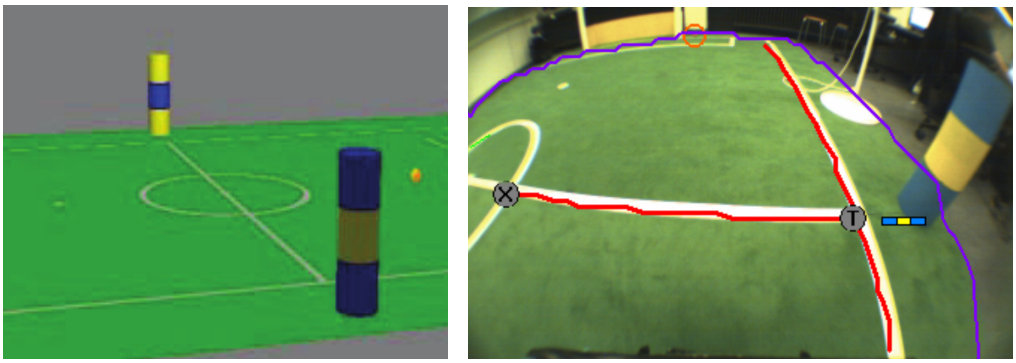
Previous visual based tracking approaches can be classified into two classes: 1) tracking objects by looking at optical flows in the image (Kendoul et al., 2009; Warren et al., 1988); 2) tracking by matching a model of the being tracked object to a part of the image (Kim et al., 2003; Papanikolopoulos et al., 1993). The main idea of tracking by optical flow is to analyze two consecutive frames of the video

stream to determine the direction of the movement in the game. The one proposed by Cheriyadat et al. (2008) use Kanade-Lucas-Tomasi sparse optical flow algorithm to track low-level feature points over time. The advantage of this algorithm is that it requires no complex shape or visual models for objects. This approach can be used for tracking balls or other robots on the field. However, it may fail if only a part of the object is observable. For example, in the soccer field, it would be difficult for the robot to track the large field model.

Drummond and Cipolla (2002) presents a framework for three-dimensional model-based tracking. The framework belongs to the second class of approaches of visual based tracking. It starts with identifying the visible edge features at each frame, which would be associated to the model edges in the next step, then the camera motion between two frames is optimized according to the result of data registration by using standard least-squares algorithm. This framework is used for tracking complex structures, from which multiple edges can be observed. For a simple model, like soccer field, essentially because most of the time it is partially observable, this framework may fail. Nevertheless, visual tracking by matching models still have comprehensive application in soccer robot environment. This is due to the fact that after detecting artificial landmarks (poles) and natural landmarks (corners, lines, goal posts), the most straightforward step is to associate the detections to the model and optimize the pose parameters based on such association.

During the RoboCup competition, different features of the environment (goals, field lines) is used as the localization information. Numerous previous works use only some particular landmarks like poles (Figure 2.3a) for localization (Chiang et al., 2010; Enderle et al., 2001; Minakata et al., 2008) . The triangulation method is used to find the coarse location of the robot. In these approaches, feature matching is straightforward because once a landmark is detected, the extracted features of this landmark can be associated to the pre-known artificial landmarks with high accuracy. In order to get a fast visual localization, Yang et al. (2012) defines 26 key points on the soccer field. Only the key points seen by a robot are used to calculate the robot and object position. If two or more key points are seen in the robot camera view, the space relationship of these key points is employed to adjust the camera pose. Unfortunately, according to the rule of humanoid soccer league, it is not possible to predefine key points and positioned them on soccer field. As we can see, all these approaches rely on the artificial landmarks, whereas, which are not allowed in competition any more.

By contrast, after removing artificial landmarks (colored poles) in Figure 2.3a, it is crucial to detect natural landmarks of the soccer field, such as field lines and goals in Figure 2.3b. Schulz and Behnke (2012) and Gudi et al. (2013) propose a method to recover lines from images and extract three types of intersections, which



(a) **(b)**

Figure 2.3: Field features: a) field landmark: poles; b) field crossings(Schulz and Behnke, 2012).

can be described as a T, L, and X crossing in Figure 2.3b. Those intersections information together with a prior knowledge of the field geometry can be used to determine the robot position and orientation. This method reduce dependence on color-coded robot soccer environment; however, the crossings are not always observable.

Strasdat et al. (2007) propose a method by using lines information for localization. By following a nearest neighbor approach, a tracking system computes the likelihood of line observations by evaluating the differences between expected and measured Hough coordinates $h = (\theta, \rho)$ of matched lines. Whereas in this approach, the system only takes into account the largest observed lines.

Our approach is developed based on state of the art of second class of visual based tracking approach. Instead of using observation model in existed approaches, we propose an approach based on point registration. To be specific, a node-graph is constructed from the observed images, meanwhile, the field model is projected to the image plane based on the estimated camera pose one step before. As an oriented point, by applying the nearest neighbor method, each observed node is assigned to a projected model point and associated with a 3D model point. Moreover, the orientation of each node are used for rejecting outliers. Our approach still works even if no particular landmarks or crossings are observed. Details of our approach are discussed in Chapter 4.

2.3 Methods for Visual Tracking

After finishing feature matching, a set of 2D-to-3D point correspondences is formed. In order to minimize the matching error, the current estimated camera pose need

to be updated by solving the Perspective-n-Point (PnP) problem. In humanoid league competition environment, one of the most popular localization mechanism is to maintain a 3D trunk pose $(x, y, \alpha)^T$ of the robot on the field (Röfer and Jüngel, 2003; Whelan et al., 2012; Yi et al., 2015) while they retrieve the other three pose parameters $(z, roll, pitch)^T$ from IMU and dead reckoning (DR) data. The camera pose can be transformed from robot trunk pose by using the transformation between camera link and trunk link retrieved from kinematic model. In order to update the estimate robot pose, both the feature matching result and the odometry data from motion model are integrated. This approach only has three parameters to be optimized, thus the optimization process can be faster than that of methods tracking 6D pose. However, this approach requires the robot equipped with well calibrated IMU, which is not always realistic. Otherwise the error from IMU data may have a negative effect on localization accuracy.

In our approach, we optimize the camera pose in 3D space with six degrees of freedom: $s_t : (x, y, z, \gamma, \beta, \alpha)_t^T$, where (x, y, z) denotes the position of the camera on the field; and (γ, β, α) describe the orientation in Euler angles of the camera. All of these six parameters would be tracked and optimized according the feature matching the result from camera, thus reducing the dependence of our system on other sensor messages such as IMU data.

Feature matching result is represented by a set of N 2D-3D point correspondences. Given such N correspondences, estimating the position and orientation of a calibrated camera is a problem named Perspective-n-Point (PnP). Different methods to the PnP problem (Bujnak et al., 2008; Ferraz et al., 2014; Wu and Hu, 2006) have been proposed in Computer Vision community, which increase the accuracy and reduce the computational complexity. These methods can be roughly classified into iterative and non-iterative techniques. Iterative PnP methods optimize a cost function calculated from all correspondences iteratively (Dementhon and Davis, 1995; Lowe, 1991; Lu et al., 2000; Quan and Lan, 1999). They usually attempt to minimize the sum of Euclidean distances between the 2D points and the projected points of their corresponding 3D points (e.g. 2D projection (Olsson et al., 2009)). Iterative PnP methods can deal with arbitrary numbers of correspondences and achieve excellent precision when they can converge. However, iterative methods are computationally expensive as they perform a greedy exploration of solution space. Furthermore, they are very sensitive to local optimum. Therefore, a good initial estimation of the camera pose to converge is required. Whereas most of the disadvantages of iterative PnP approaches can be overcome by non-iterative techniques. One of the most popular and efficient method is EPnP (Lepetit, Moreno-Noguer, et al., 2009). This method defines four control points to represents all 2D and 3D corresponding points. It reduces the PnP problem

to retrieve the position of four control points, whose weighted sum can be used to express all the 3D points. The algorithm runs much faster than most of the iterative ones, but can be slightly less accurate.

In our visual tracking system, both hill climbing method and EPnP method have been implemented for pose estimation. Since the non-iterative method EPnP tends to be sensitive to noise, it is often used within RANSAC scheme.

3 Foundation

3.1 Camera Model

It is important to be familiar with the camera model while using images captured by the camera for visual tracking. Thus, in this section, we introduce the standard pinhole camera model which is also known as central-projection camera model (Hartley and Zisserman, 2003). More specifically, we first introduce the mathematical relationship between the coordinates of a 3D point and its projection onto the image plane named perspective projection model. After that we introduce the intrinsic camera parameters followed by a brief discussion about external parameters matrix. In the end, we describe how to estimate the intrinsic parameters and how to handle the lens distortion.

3.1.1 The Perspective Projection Model

Perspective projection model (Figure 3.1) defines the relationship between a 3D point and its projection onto the image plane (Lepetit and Fua, 2005). Considering a 3D point $M = [X, Y, Z]^T$ in a world coordinate system, the corresponding 2D point $m = [u, v]^T$ on the image plane can be described with the following equation:

$$\lambda \widehat{m} = P\widehat{M} \tag{3.1}$$

where $\widehat{m} = [u, v, 1]^T$ and $\widehat{M} = [X, Y, Z, 1]^T$ are the homogeneous coordinates of points m and M; λ is a homogeneous scale factor to constrain the z-axis value of $\widehat{m}$ is 1; P is a 3×4 perspective projection matrix which can be decomposed to the following equation:

$$P = K[R|t] \tag{3.2}$$

where

- K is the 3×3 camera calibration matrix that depends on the internal parameters of the camera (focal distance, and radial lens parameters);
- $[R|t]$ is the 3×4 external parameters matrix depends on the position and orientation of the camera in the world coordinate. More specifically, R is a

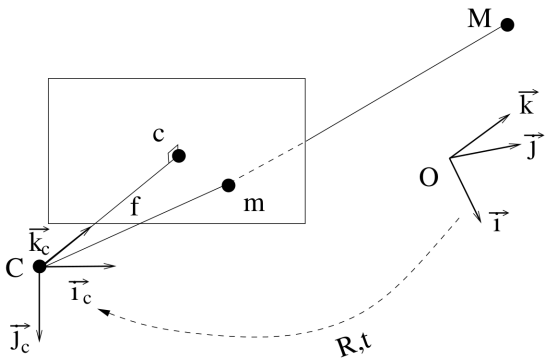


Figure 3.1: Ideal pinhole camera model describes the relationship between a 3D point M and its corresponding 2D projection m onto the image plane (Lepetit and Fua, 2005).

3×3 rotation matrix and t is a 3×1 translation vector. More details about the calibration matrix and external parameters matrix will be discussed in the following sections.

3.1.2 The Calibration Matrix

As we mentioned above, K is the camera calibration matrix that only depends on the five intrinsic camera parameters as follows:

- focal length f,
- pixel size in x and y directions: s_x and s_y,
- and coordinate of principal point $c = [c_x, c_y]^T$.

Principal point c is the intersection of the optical axis and the image plane. Sometimes K also has a skew parameter s which is non-zero only if x and y directions are not perpendicular. K can be denoted as the following upper triangular matrix:

$$K = \begin{bmatrix} f_x & s & c_x \\ 0 & f_y & c_y \\ 0 & 0 & 1 \end{bmatrix} \tag{3.3}$$

where f_x and f_y are proportional to the focal length f: $f_x = f/s_x$ and $f_y = f/s_y$.

In most of 3D tracking methods, the intrinsic camera parameters are supposed to be given. Those parameters are independent from the scene being observed by the camera. Therefore, it is always preferable to estimate the parameters offline

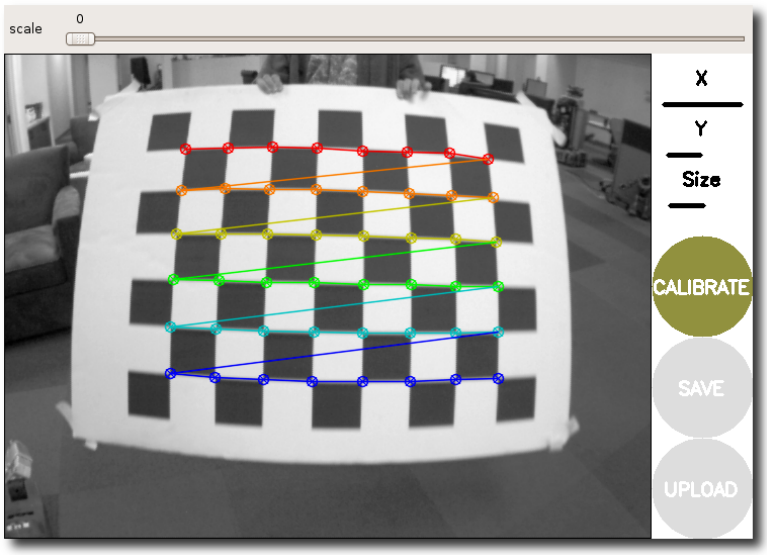


Figure 3.2: A calibration grid used for the estimation of the camera calibration matrix (Bradknox, 2015).

before visual tracking. In fact, camera calibration is a step to determine all the camera parameters from the applied camera model. Various methods for geometric camera calibration are presented in previous work (Melen, 1994; Slama et al., 1980; Tsai, 1987; Z. Zhang, 2000). These calibration methods can be roughly classified into three classes: photogrammetric calibration (Hatze, 1988), calibration from view geometry (Hartley and Zisserman, 2003), and self-calibration (Pollefeys et al., 1999). Visual tracking greatly benefits from the development of calibration techniques, which provides more accurate parameters. One of popular methods uses a known planar calibration pattern, such as a checkerboard (Q. Zhang and Pless, 2004) (Figure 3.2). Moreover, the parameters are optimized according to the detection of image corners, corners at the intersections of black and white squares, and corners at the intersections of two groups of grid lines.

3.1.3 The External Parameters Matrix

It is worth mentioning that the camera is not always at the origin of the world coordinate system. There is a transformation between the camera coordinate and the world coordinate. This transformation can be described by a rotation matrix R and a translation vector t. Therefore, the 3×4 external parameters $[R|t]$ matrix actually refers to the position and orientation of camera respectively. When the internal parameters have been estimated separately, the target of visual tracking is to estimate R and t with a set of 2D to 3D correspondence.

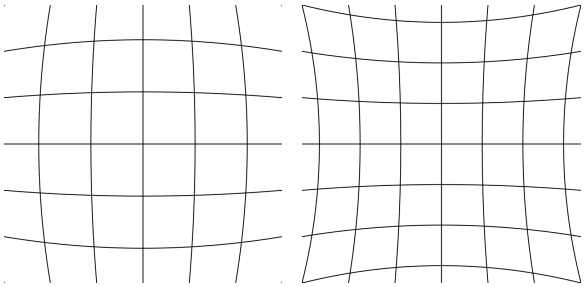


Figure 3.3: Two types of lens distortion: (a) barrel; (b) pincushion.

3.1.4 Lens Distortion

Due to distortions caused by camera lenses, the perspective projection model of the camera is not sufficient to define the relationship between 3D points in world coordinate and corresponding 2D points in image coordinate. There exist several types of lens distortion, such as Barrel distortion and Pincushion distortion, and both of them are the most common distortions (Hugemann, 2010) (Figure 3.3). Barrel distortion, $f(r) < 1$, i.e. the off-center distances of points near the image borders are less than they should be after an ideal perspective mapping. Pincushion distortion, $f(r) > 1$, has a visible effect that lines that do not go through the center of the image are bowed inwards or towards the center of the image.

Because the lens distortions are radially symmetric, or approximately so, arising from the symmetry of a photographic lens. The distortion parameters can be corrected by applying proper algorithmic transformations to the digital photograph. The most well know method is Brown's distortion model (Brown, 1966), also known as the Brown-Conrady model based on previous work of Conrady (Conrady, 1919). By applying the corrected distortion parameters to the captured images, the distortion effect can be eliminated, as shown in Figure 3.4.

Usually we take into account the radial and tangential factors for the distortion; the radial factor uses the following formula:

$$\begin{aligned} x_{corrected} &= x(1 + k_1 r^2 + k_2 r^4 + k_3 r^6) \\ y_{corrected} &= y(1 + k_1 r^2 + k_2 r^4 + k_3 r^6) \end{aligned} \tag{3.4}$$

With above formula, an old pixel point at (x, y) represents the coordinates in the input image, and its position on the corrected output image is $(x_{corrected}, y_{corrected})$

Tangential distortion occurs because the image taken lenses are not perfectly

(a) **(b)**

Figure 3.4: Undistorting an image. (a) Due to distortion effect, projections of straight lines are curved. (b) After undistortion, the projections of straight lines are straight.

parallel to the image plane. The distortion can be corrected via the formulas:

$$\begin{aligned} x_{corrected} &= x + [2p_1xy + p_2(r^2 + 2x^2)] \\ y_{corrected} &= y + [p_1(r^2 + 2y^2) + 2p_2xy] \end{aligned} \tag{3.5}$$

As a consequence, we have five distortion parameters presented as one row matrix in 5 columns:

$$Distortion_{coeficients} = (\kappa_1, \kappa_2, \kappa_3, \kappa_4, \kappa_5) \tag{3.6}$$

3.2 Camera Pose Estimation

3.2.1 Camera Pose Parameterization

In order to measure the camera pose, the R and t should be properly parameterized. The representation of translation t is straightforward. However, the parameterization of R is challenging.

It is well known that a rotation in 3D space has only three degrees of freedom. Thus, it is not suitable to employ 3×3 rotation matrix with nine parameters to represent the rotation information in tracking process. A variety of rotation representations have been proposed, which include rotation matrices, Euler angles, unit quaternions, Axis-Angle pair, and Exponential Map (Rowenhorst et al., 2015). Each parameterization has distinct advantages and disadvantages with respect to the ease of use for calculations and data visualization. We will briefly discuss two of them that have been used in our approach, Euler angles and quaternions. More

details about rotation representation can be found in the paper of original work (Eberly, 2002; Grassia, 1998).

1. Euler angles

   The idea behind Euler angles is to split a complete rotation of the coordinate system into three constitutive rotations around X, Y and Z axes respectively. It is worth to know that the resulting rotation depends on the order in which the three rolls are performed. The Euler angles can be converted to a rotation matrix. For instance, a set of Euler angles (α, β, γ) representing a first rotation around Z axis by an angle α, a second rotation around Y axis by an angle β, and a final rotation around X axis by an angle γ, can be converted to a rotation matrix (Edmonds, 1996) as follows:

$$R = \begin{bmatrix} \cos\alpha & -\sin\alpha & 0 \\ \sin\alpha & \cos\alpha & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} \cos\beta & 0 & \sin\beta \\ 0 & 1 & 0 \\ -\sin\beta & 0 & \cos\beta \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos\gamma & -\sin\gamma \\ 0 & \sin\gamma & \cos\gamma \end{bmatrix} \tag{3.7}$$

   Even though Euler angles are more human understandable, they have disadvantages, for instance, ambiguity and gimbal lock. The conversion from a general rotation to Euler angles is ambiguous because the same rotation can be obtained with different sets of Euler angles (Hughes et al., 2013). Furthermore, Euler interpolation works well when the axis of interpolation coincides with one of the X, Y, and Z rotation axes, but it is not as good as interpolating arbitrary orientations. However, these problem can be avoided by using quaternions representation of rotation.

2. Unit quaternions

   Unit Quaternions (quaternions with the magnitude equal to one) that form a four-dimensional vector space are extensively used to represent rotations. A unit quaternion is denoted by $q = \omega + xi + yj + zk$, where ω, x, y, z are real numbers and the 4-tuple (ω, x, y, z) s unit length, with $i^2 = j^2 = k^2 = ijk = -1$. Bellow are some properties of unit quaternions:

   - The corresponding rotation matrix R of a quaternion q is defined as follows:

$$R = \begin{bmatrix} 1 - 2y^2 - 2z^2 & 2xy + 2wz & 2xz - 2wy \\ 2xy - 2wz & 1 - 2x^2 - 2z^2 & 2yz + 2wx \\ 2xz + 2wy & 2yz - 2wx & 1 - 2x^2 - 2y^2 \end{bmatrix} \tag{3.8}$$

- The inverse (reciprocal) of a quaternion is given by:

$$q^{-1} = \frac{\overline{q}}{[n(q)]^2} \tag{3.9}$$

  where $\overline{q}$ is the quaternion conjugate given by $\overline{q} = \omega - xi - yj - zk$; and $[n(q)]$ is the quaternion norm given by $[n(q)] = \sqrt{\overline{q}q} = \sqrt{\omega^2 + x^2 + y^2 + z^2}$.
- The quaternion multiplication corresponds to matrix multiplication. Thus the multiplication of two quaternions represents for composing the corresponding two rotations: performing one rotation and then performing another one. The product of two unit quaternions $q_n = \omega_n + x_n i + y_n j + z_n k$ for $n = 0, 1$ is defined by distributing the product over the sums. The most noteworthy is that the order of operands is important because multiplication of quaternions is not commutative.

$$\begin{aligned} q_0 q_1 = &(w_0 w_1 - x_0 x_1 - y_0 y_1 - z_0 z_1) + \\ &(w_0 x_1 + x_0 w_1 + y_0 z_1 - z_0 y_1) + \\ &(w_0 y_1 - x_0 z_1 + y_0 w_1 + z_0 x_1) + \\ &(w_0 z_1 + x_0 y_1 - y_0 x_1 + z_0 w_1) \end{aligned} \tag{3.10}$$

Unit quaternion is one of the most widely used rotation representations. Because unit quaternion allows each rotation to be represented relative to a reference point uniquely. In comparison to Euler angles, the quaternions are much more efficient in interpolation. Moreover, the quaternions are widely used for obtaining smooth rotation by interpolating between orientations, which is known as spherical linear interpolation (slerp) (Mukundan, 2002).

3.2.2 Reprojection Error Function

Given N 3D reference points $\widehat{M}_i = [x_i, y_i, z_i, 1]^T$, $i = 1, 2, \ldots, N$, in the word coordinate system, and their corresponding projections $\widehat{m}_i = [u_i, v_i, 1]^T$, on the image plane, the pose estimating problem aims to determine the perspective projection matrix P, namely the rigid transformation that relates images to the known objects. The pose estimation problem aims to minimize the sum of reprojection errors between the projected 3D points and their corresponding observed points on 2D image plane.

$$[R|t] = arg \min_{[R|t]} \sum_{i}^{N} dist^2(P\widehat{M}_i, \lambda \widehat{m}_i) \tag{3.11}$$

The optimization approaches depend on the definition of distance function. Let $\widehat{u}_i = P\widehat{M}_i = [\widehat{u}_i, \widehat{v}_i, \widehat{w}_i]^T$, and $u_i = \lambda\widehat{m}_i = [u_i, v_i, w_i]^T$. Then two most widely used definitions about the distance in current pose optimization techniques can be defined as follows:

- Algebraic distance: $d^2_{alg}(u_i, \widehat{u}_i) = (v_i\widehat{w}_i - w_i\widehat{v}_i)^2 + (w_i\widehat{u}_i - u_i\widehat{w}_i)^2$.
- Euclidean distance: $d^2_{edu}(u_i, \widehat{u}_i) = \| u_i - \widehat{u}_i \|^2$.

3.2.3 EPnP: a Non-iterative Approach for Pose Estimation

The Perspective-n-Point (PnP) problem refers to the problem to estimate the perspective projection matrix P without any prior knowledge of camera pose by given n 3D-to-2D point correspondences. As we discussed in Section 3.1.1, $P = K[R|t]$. Considering the camera has been calibrated, which means K is given, the target of PnP problem comes to determine camera orientation represented by rotation matrix R and camera position represented by translation vector t. The constraint equations can be formulated as $\lambda m_i = PM_i$, which can be rewritten to the following two equations:

$$\begin{cases} \frac{P_{11}X_i + P_{12}Y_i + P_{13}Z_i + P14}{P_{31}X_i + P_{32}Y_i + P_{33}Z_i + P34} = u_i \\ \frac{P_{21}X_i + P_{22}Y_i + P_{23}Z_i + P24}{P_{31}X_i + P_{32}Y_i + P_{33}Z_i + P34} = v_i \end{cases} \tag{3.12}$$

Given a set of N correspondences, it will generate $2 \times N$ number of above equations, which can be rewritten in the form $Ap = 0$, where p is a vector made of the coefficients P_{ij}. Such a constrained problem can be solved by using Singular Value Decomposition (SVD).

Different methods for solving the PnP problem have been proposed by Computer Vision community, which can increase the accuracy and reduce the computational complexity (Bujnak et al., 2008; Ferraz et al., 2014; Wu and Hu, 2006). Here we briefly introduce an efficient method for solving PnP problem, EPnP presented by Lepetit, Moreno-Noguer, et al. (2009). EPnP is a non-iterative approach with higher accuracy and lower computational complexity than non-iterative state-of-the-art methods. Moreover, EPnP is much faster than iterative methods only with a little loss of accuracy. The core idea of EPnP is to express N 3D points as a weighted sum of four virtual control points. Then the problem is reduced to a less complex problem to estimate the coordinates of these control points in the camera referential. The less complex problem can be done in O(n) time to express these coordinates as weighted sum of the eigenvectors of a 12×12 matrix and solve a small constant number of quadratic equations to pick the right weights.

More details about EPnP can be found in the paper of original work (Lepetit, Moreno-Noguer, et al., 2009).

Here we briefly summarize a linear formulation of the problem that results from the EPnP algorithm:

- Choose 4 of the 3D-to-2D correspondences named as control points:

$$c_j^c \sim c_j^w, j = 1, \ldots, 4 \tag{3.13}$$

  where we specify the points in the image coordinate with superscript c and the points in 3D world coordinate with superscript w.

- Express each reference point as a weighted sum of the control points:

$$\begin{aligned} p_i^w &= \sum_{j=1}^{4} \alpha_{ij} c_j^w, \\ p_i^c = \sum_{j=1}^{4} \alpha_{ij} c_j^c, \; &with \sum_{j=1}^{4} \alpha_{ij} = 1 \end{aligned} \tag{3.14}$$

  where the α_{ij} are homogeneous barycentric coordinates. They are uniquely defined and can easily be estimated.

- Perform the 2D projections of the image points:

$$\forall i, \lambda \begin{bmatrix} u_i \\ v_i \\ 1 \end{bmatrix} = K p_i^c = K \sum_{j=1}^{4} \alpha_{ij} \begin{bmatrix} x_j^c \\ y_j^c \\ z_j^c \end{bmatrix} \tag{3.15}$$

  where u_i represents the 2D projections of p_i^w.

- The concatenation of equation (3.15) for all N correspondences can be expressed as a linear system $Mx = 0$ where M is a $2N \times 12$ known matrix. The solution belongs to the null space or kernel of M, and can be expressed as:

$$x = \sum_{i=1}^{N} \beta_i v_i \tag{3.16}$$

  where the set v_i represents the columns of the right-singular vectors of M corresponding to the N null singular values of M.

The above discussed approaches do not require initial assumption of camera pose and they are efficient; however, they are rather sensitive to noise. If a wrong 3D-

to-2D point correspondences occurs, the algorithm will converge to an absolutely wrong estimation.

3.2.4 Hill Climbing Method: an Iterative Approach for Pose Tracking

Hill climbing method (Russell et al., 1995) is a local search optimization technique that can be used for tracking the camera pose. In terms of accuracy, it is highly advantageous to continuously estimate the external parameters matrix based on prior knowledge about the camera pose acquired one step before.

Hill climbing method is popular because it is relatively simple to be implemented. Although more advanced algorithms may give better results, hill climbing works well in most normal situations. It is an iterative algorithm that attempts to maximize (or minimize) a function $f(x)$, where x can be either discrete states or continuous states. When hill climbing is applied in a continuous space, the algorithm is called gradient ascent (or descent). It starts with a random solution to a problem, afterwards incrementally makes small changes to reach a better solution. Once the algorithm cannot find any better neighbors, it will be terminated (Russell et al., 1995).

The pseudo code of hill climbing method is described in Algorithm 1.

Similar to other local optimizer, hill climbing method has drawbacks, for instance, it is not guaranteed to find the best global optimal solution; on the other hand, it may get stuck on local optimum solutions. Fortunately, regrading our visual tracking task, the camera pose estimated from the previous step can be served as a good starting solution for hill climbing method. Consequently, hill climbing method will have lower chance of getting stuck on local optimum.

3.2.5 RANSAC for Robust Estimation

The result of PnP problem will strongly be affected by outliers which are spurious observations. In practice, a PnP solution is usually used with combining RANSAC schema to eliminate outliers. Fischler and Bolles (1981) propose RANdom SAmple Consensus (RANSAC) algorithm which is a popular technique designed to cope with a large proportion of outliers in input data. RANSAC is a resampling technique that generates candidate solutions by using the smallest possible observations required to estimate the underlying model parameters (Derpanis, 2010). This resampling technique maximizes the probability that at least one subset contains no outliers and therefore it produces a valid hypothesis. The RANSAC algorithm pseudo code is shown in Algorithm 2.

Algorithm 1: Hill Climbing Algorithm

```
Data: startPose, measurements
Result: optimalPose
optimalPose = startPose;
while True do
    L = NEIGHBORS(currentPose);
    nextError = +INF;
    nextPose = NULL;
    for ∀x ∈ L do
        if Error(x) < nextError then
            nextPose = x;
            nextError = Error(x);
        end
    end
    if nextError < Error(optimalPose) then
        optimalPose = nextPose;
    else
        break;
    end
end
return optimalPose;
```

The number of iterations (N) is chosen high enough to ensure that the probability that at least one of the sets of random samples does not include an outlier is high enough. Let u represent the probability that any selected data point is an inlier and $v = 1 - u$ the probability of observing an outlier. N iterations of the minimum number of points denoted m are required, where

$$1 - p = (1 - u^m)^N \tag{3.17}$$

and thus with some manipulation:

$$N = \frac{\log(1 - p)}{\log(1 - (1 - v)^m)} \tag{3.18}$$

Regarding our 3D tracking tasks, the PnP algorithm randomly chooses 6 of correspondences to estimate camera pose. For each of this estimation, the 3D points projected close enough to their corresponding 2D points are considered as inliers. RANSAC performs iteratively until the best estimation camera pose is found.

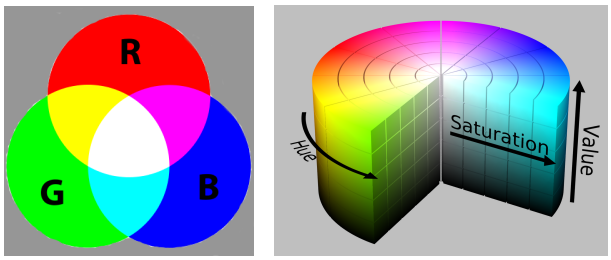
Algorithm 2: RANSAC algorithm

Data: data - a set of points S
model - a model that can be fitted to data
k - the minimum number of data required to estimate the model
N - the maximum number of iterations allowed in the algorithm
d - a threshold value for determining whether a data point is consistent with a parameter estimate
τ - a threshold for the fraction the inlier number to the total number of points
Result: Bestfit model parameters
$iter\_num = 0$;
while $iter_{num} < N$ **do**
  $iter\_num++$;
  Select randomly k points among S;
  Generate a best fit model M from the k points;
  $inliers = \phi$;
  for $\forall p \in S$ **do**
    if $dist(p, M) < d$ **then**
      $inliers = inliers \cup p$;
    end
  end
  $inlierPct = \| inliers \| / \| S \|$;
  if $inlierPct > \tau$ **then**
    Generate an optimal model M_o from the $inliers$;
    berak;
  end
end

3.3 Visual Feature Detection

3.3.1 Image Color Formats

A color image is a digital image that includes color information for each pixel. Such color information can be stored in different formats. Before we do the further feature detection in the image, it is necessary to identify the color format of input images and the color format for further image processing. RGB color format is based on the additive of color components red (R), green (G) and blue (B) (Figure 3.5a), the HSV color format describes a color by using the three color components hue (H), saturation (S), and brightness value (V) (Figure 3.5b). In our vision system, images are captured in RGB format and then converted to the HSV format. We do this conversion because unlike RGB, HSV is closer to the



(a) (b)

Figure 3.5: Two color format. (a) An additive color model: RGB. b) HSV color wheel.

way how human eyes perceive color (Gonzalez et al., 2004). Moreover, HSV color format is very popular for designing and editing because it gives the user a good perception of the resulting color for a certain value.

The following equations shows the transformation from RGB to HSV.

$$\begin{cases} H = \begin{cases} (6 + \frac{G-B}{C_{MAX}-C_{min}}), & if\ R = C_{Max} \\ (2 + \frac{B-R}{C_{MAX}-C_{min}}), & if\ G = C_{MAX} \\ (4 + \frac{R-G}{C_{MAX}-C_{min}}), & if\ B = C_{MAX} \end{cases} \\ S = C_{MAX} - C_{min}/C_{MAX} \\ V = C_{MAX} \end{cases} \tag{3.19}$$

where H is hue with range $0° \sim 360°$; S means saturation, with range $0 \sim 1$; V represents value with range $0 \sim 255$. The RGB values are confined by equation (3.20) :

$$\begin{cases} C_{MAX} = max(R, G, B) \\ C_{min} = min(R, G, B) \end{cases} \tag{3.20}$$

where C_{MAX} and C_{min} represent the maximum and minimum value in the RGB color components. The further image processing can be achieved based on this HSV color format.

3.3.2 Image Segmentation

Even though the simple color segmentation and blob detection methods are no longer good enough for detecting field lines in current RoboCup humanoid league competition environment because it is still a fundamental process in field boundary detection or obstacle detection. In a brief, image segmentation aims to divide an image into multiple regions having a strong correlation, and then to find the

disjoint areas corresponding to real-world objects. For different segmentation purposes, the segmentation process can be applied based on different properties of the image, such as color, texture, and brightness. Various image segmentation techniques (Khan, 2013) have been developed, some of the most important and extensively used image segmentation techniques are surveyed below.

- Threshold based image segmentation
  The most common algorithms based on global knowledge are represented by histograms. Histogram thresholding is the simplest segmentation process. A threshold is set to decide whether a pixel belongs to an object or background. Thus, the value of the threshold should be chosen properly. In a static environment, this kind of simple thresholding approach can be successful. However, it is rather difficult to determine a single threshold that is suitable for segmenting every images captured under different environment settings. Thus, thresholding approach is very sensitive to noise. Several improved methods exist such as pre-processing by different filters to eliminate the outliers (Gonzalez et al., 2004).
- Edge based image segmentation
  Edge based image segmentation divides the image by observing the changes in intensity or pixels of an image. Many kernels can be used for detecting edges such as Difference of Gaussian (DoG) and Laplacian of Gaussian(LoG) (Marr and Hildreth, 1980). Segments are detected based on edge information.
- Region based image segmentation
  Region based image segmentation divides an image into different regions based on pre-defined criteria, i.e., color, intensity, and object. It can be obtained through a growth process in which a pre-selected seed is used. The region growing can be considered as a sequential clustering or classification process. Region based segmentation is simple and noise resilient compared to other methods.

3.3.3 Line Detector

Before we introduce the line detector, we firstly review some basic knowledge about edge detector. The main purpose of edge detection is to identify sharp discontinuities from an image. There are many methods for edge detection; and most of them can be classified into two categories, the search-based and the zero-crossing based.

The search-based methods detect edges in two steps. Firstly, it convolves the input image with an adapted mask to generate a gradient image. And then it

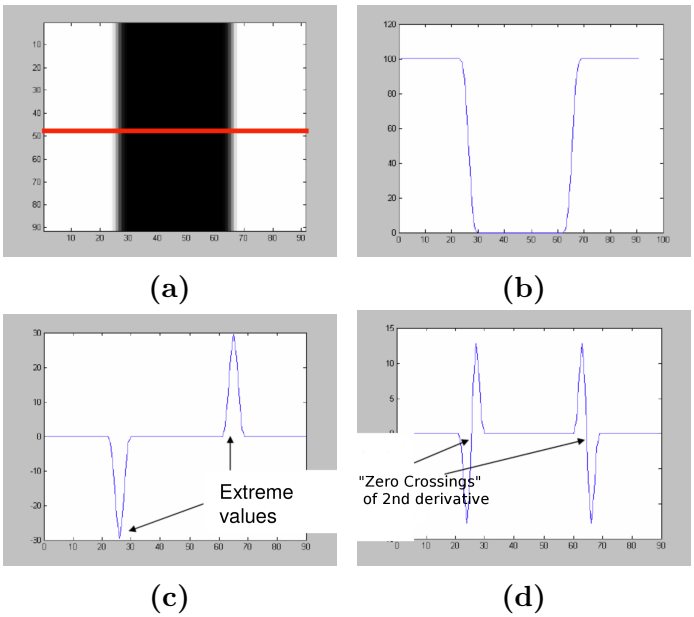


Figure 3.6: Line detectors. (a) A line with two vertical edges. (b) Intensity function (along horizontal scan line). Edges are regions with a high slop. (c) The first derivative of the intensity function. High slopes are found from the maximum /minimum of the first derivative. (d) High slopes are found from the zero crossings of the second derivative.

detects edges by finding the local maximum and minimum of gradient magnitude (Figure 3.6c). Many classical operators such as sobel, prewitt, and robert are the first order derivative detectors (Gonzalez et al., 2004).

The zero-crossing based methods extract zero crossings in a second-order derivative expression is computed from the image (Figure 3.6d). Marr and Hildreth (1980) proposes two typical operators, DoG and LoG, to calculate the second order derivative. In the first step, both of the operators smooth an image by applying convolution to the image with Gaussian kernel of certain width σ_1. In the second step, DoG smooths the image with another Gaussian kernel with width σ_2, while LoG uses Laplace for edge detection. However, it is possible to merge the two steps to a single one: convoluting the image with LoG kernel or DoG kernel. The behavior of the LoG edge detector is largely governed by the standard deviation of the Gaussian smoothing filter used in the LoG filter.

The edge detector will highlight those pixels that are most likely on edges. The algorithm for further model fitting such as probabilistic Hough line detector can be applied to find edge lines among those edge pixels. Ideally two close and parallel edges lines can be merged to find a field line, which is bordered by two edges. However, many line fitting algorithms do not work well for detecting continuous

edge line segments. Thus, merging those small detected edge segmentations with noise angles is a tough task.

Similar to edge detector, the line detector is also to convolve the image with a kernel. However, unlike edge detectors which detect two edges from a line with a certain width, line detector aims to detect one single line that centers between two edges. This kernel should be designed properly so the center of the line gets higher response than other areas.

3.4 Recursive State Estimation

In probabilistic robotics, a concept of belief is used to represent the robot's internal knowledge about the state of the environment (Thrun et al., 2005). Probabilistic state estimation algorithms compute belief distributions over possible world states. Pose tracking aims to update the pose distribution in the space of possible camera poses by incorporating new measurements.

The most general algorithm for dynamic state estimation is the Bayes filter algorithm. To our best knowledge, the Kalman filter is the best studied technique for implementing Bayes filters. We will describe the Bayes filter and the Kalman filter in the following two sections respectively.

3.4.1 The Bayes Filter

We firstly explain some denotations of Bayes filter. A belief is a probability to a possible hypothesis. We will denote the belief over a state variable x_t by $bel(x_t)$ which is an abbreviation for the posterior.

$$bel(x_t) = p(x_t|z_{1:t}, u_{1:t}) \tag{3.21}$$

Equation (3.21) shows the probability distribution over the state x_t at time t, which is conditioned on all past measurements $z_{1:t}$ and all past controls $u_{1:t}$. Occasionally, it is useful to calculate a posterior before incorporating the measurement z_t at time t. Such a posterior will be denoted as follows:

$$\overline{bel}(x_t) = p(x_t|z_{1:t-1}, u_{1:t}) \tag{3.22}$$

This step is named as prediction as it predicts the state at time t based on previous state posterior before incorporating the measurement at time t. Calculating $bel(x_t)$ from $\overline{bel}(x_t)$ is called correction or the measurement update.

One important assumption of the Bayes filter is the Markov assumption which postulates that past and future data are independent if one knows the current

state x_t. So our belief and posterior formula can be simplified as follows:

$$\begin{cases} \overline{bel}(x_t) = p(x_t|u_t, x_{t-1}) \\ bel(x_t) = p(x_t|z_t, u_t) \end{cases} \tag{3.23}$$

3.4.2 The Kalman Filter

The Kalman filter is a universal tool for recursive state estimation and it has been extensively applied to 3D pose tracking. There are many extensions to the Kalman filter, such as the Extended Kalman Filter (EKF) (Julier and Uhlmann, 1997), The Unscented Kalman Filter (UKF)(Wan and Van Der Merwe, 2000), and we only describe it here the most basic form. (Thrun et al., 2005) is a good reference dedicated to the Kalman filter.

The Kalman filter represents state distributions by the moments representation: at time t, the belief is represented by a Gaussian with mean μ_t and the covariance Σ_t. Posteriors are Gaussian if the following three properties and the Markov assumptions hold:

- The next state probability $P(x_t|u_t, x_{t-1})$ must be a linear function in its arguments with added Gaussian noise. This is expressed by the following equation:
$$x_t = A_t x_{t-1} + B_t u_t + \epsilon_t \tag{3.24}$$
where matrix A_t and B_t are called the transition matrix and the random variable ϵ_t is a Gaussian random vector that models the randomness in the state transition. Its mean is zero and its covariance is denoted R_t.
- The measurement probability $p(z_t|x_t)$ must be linear in its arguments, with added Gaussian noise:
$$z_t = C_t x_t + \delta_t \tag{3.25}$$
where δ_t represents the measurement noise and C_t is a linear transition matrix.
- Finally, the initial belief $bel(x_0)$ must be normally distributed, with the mean of this belief μ_0 and the covariance Σ_0.

The Kalman filter algorithm is depicted in Algorithm 3. The input of the Kalman filter is a belief at time $t-1$ represented by μ_{t-1} and Σ_{t-1}. The update process can be divided into two steps:

- Prediction step: updating $\overline{bel}(x_t)$
$\overline{\mu}_t$ and $\overline{\Sigma}_t$ which represent the belief $\overline{bel}(x_t)$ are calculated by incorporating the control u_t. $\overline{\mu}_t$ is updated by using the deterministic version of the state

Algorithm 3: The Kalman filter algorithm (Thrun et al., 2005)

Data: $\mu_{t-1}, \Sigma_{t-1}, u_t, z_t$
Result: μ_t, Σ_t
$\overline{\mu}_t = A_t \mu_{t-1} + B_t u_t$;
$\overline{\Sigma}_t = A_t \Sigma_{t-1} A_t^T + R_t$;
$K_t = \overline{\Sigma}_t C_t^T (C_t \overline{\Sigma}_t C_t^T + Q_t)^{(-1)}$;
$\mu_t = \overline{\mu}_t + K_t(z_t - C_t \overline{\mu}_t)$;
$\Sigma_t = (I - K_t C_t)\overline{\Sigma}_t$;
return μ_t, Σ_t

transition function of equation (3.24).The covariance $\overline{\mu}_t$ is updated depending on the linear matrix A_t and the covariance of random noise R_t.

- Correction step: updating $bel(x_t)$
  The Kalman gain denoted as K_t, is computed in the fifth line of Algorithm 3. It specifies the degree to which the measurement is incorporated into a new state estimate. The μ_t is adjusted proportion to the Kalman gain and the deviation of the actual measurement z_t. Finally, the new covariance of the posterior belief Σ_t can be calculated.

In the context of 3D tracking, during each updating process of the Kalman filter, the prior belief $\overline{bel}x_t$ can be used to project 3D model to the image frame. After comparing the projected model with the observed features, the 2D to 3D data correspondences can be generated. The optimal estimated pose for these correspondences can be considered as a new measurement z_t and incorporated to the correction step of the Kalman filter.

The Kalman filte has been widely used to combine noise measurements from different image frames and stabilize the camera moving trajectory. However, it restricts the state probability distribution to be Gaussian and the state transition must be linear.

3.5 Summary

In this chapter, we first explain the perspective projection model of the camera and run into details of each element in this model. We discuss two methods for estimating the camera pose, which are a non-iterative approach EPnP, and an iterative approach hill climbing method, followed by a brief description for RANSAC algorithm for robust estimation. Furthermore, we presented two image processing techniques, image segmentation and line detection, which are highly

related to our tasks. In the end, we introduce the Kalman filter for recursive state estimation.

4 Vision-based 3D Pose Tracking Using Field Lines

This chapter introduces our visual tracking system, which is designed to be used for tracking the 6-DoF camera pose on the soccer field. The initial estimated camera pose can be acquired by transforming the trunk pose of a robot according to kinematic model, where the trunk pose is provided by the user. The main steps of our visual tracking algorithm are as follows:

1. Extracting features (field boundary, obstacles, and field lines) from the captured image.
2. Converting the detected field lines to observations that can be used for matching.
3. Generating 2D projected field model from 3D field model based on estimated camera pose.
4. Establishing correspondences between 2D observations and 3D model points.
5. Determining the optimal camera pose s_t that minimizes the error function.
6. Using multi-hyphothesis approach to find the optimal camera pose after losing track.
7. Using Kalman filter for statistical state fusion.

An overview of our visual tracking system is shown in Figure 4.1

4.1 Object Detection

Before using the camera for vision tracking task, the camera internal parameters and distortion parameters are pre-calibrated. In our vision system, the image is captured in the RGB color format with a resolution 640×480. We first convert RGB format to the HSV format using the method we mentioned in Section 3.3.1. We use HSV color format due to that it gives the user an intuitive perception of the resulting color for a certain color value.

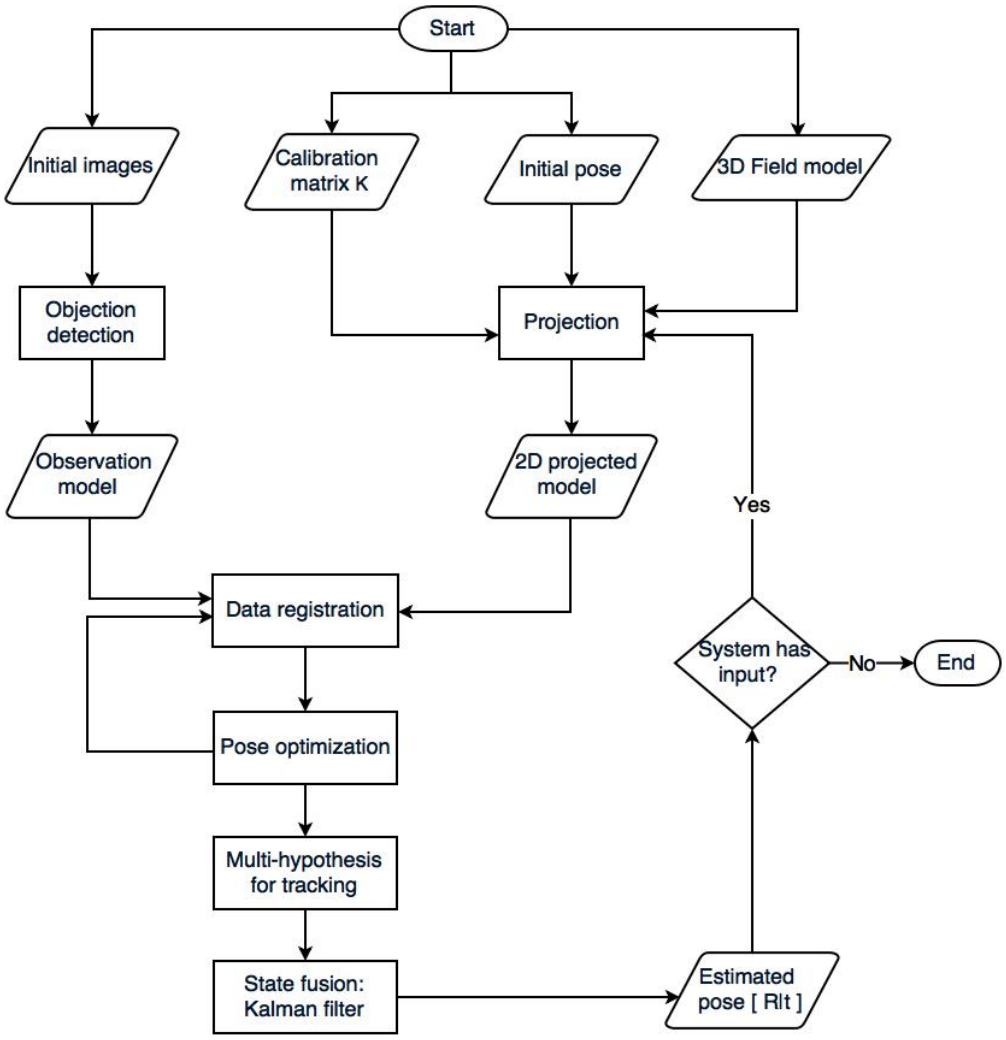


Figure 4.1: Overview of our visual tracking system.

4.1.1 Field Detection

In soccer robot competition, every interesting object is located inside the green field. It is very important to know where the field ends. Because knowing the position of filed ends enables us to eliminate some unwanted detections such as white lines outside the field. Thus we can focus on further processing on the area inside the field boundary. As a result, the processing speed can be increased.

In our vision system, we employ a strategy to find the field boundary, which is similar to the strategy proposed by Farazi, Allgeuer, and Behnke (2015). The basic idea of this approach is to find some biggest green regions based on color segmentation, thereby a convex hull is found from those green regions. However, due to the effect of lens distortion, finding convex hull from initial image may cause many false positive areas (Figure 4.2a). Therefore, we should do the convex hull finding on undistorted image frame. Whereas undistorting the whole image is time consuming, and doing further process on unsorted images requires much more computational spaces because the size of undistorted image is much larger than that of the initial image. For example, in our case the resolution of initial image is 640×480 (Figure 4.2a), while the size of undistorted image is 1661×1251 (Figure 4.3). So we present an approach to find the convex hull only by undistorting a minimal set of boundary points. Our boundary finding algorithm can be described as follows:

- Binarizing the color according to the green color ranges
  The user defines ranges for green color in H, S, V dimensions. The color at pixel position (i, j) in input image is classified according to the following equation:

$$g_{ij} = \begin{cases} 1 & if\ (H_{ij} \in [H^g_{min}, H^g_{MAX}] \wedge S_{ij} \in [S^g_{min}, S^g_{MAX}] \wedge V_{ij} \in [V^g_{min}, V^g_{MAX}]) \\ 0 & otherwise \end{cases} \tag{4.1}$$

  where g_{ij} denotes whether the color is green or not; H^g_{min}, H^g_{MAX}, S^g_{min}, S^g_{MAX}, V^g_{min}, V^g_{MAX} are user defined green color thresholds. The binarizing output of image shows in Figure 4.2b where the white pixels correspond to the pixels that have been classified to the green.
- Finding regions of interest from the binarizing image
  Continuous regions, so-called contours are retrieved by using the algorithm proposed by Suzuki (1985). Each contour is stored as a vector of points. As shown in Figure 4.2c, a huge number of contours (in blue) are detected, some of which are rather small. Based on the assumption that the field locates on the bottom of the image, and should be relatively large, only those contours

with the area larger than a certain threshold and with the bottom close to the bottom of the image, will be chosen for generating the field hull. In Figure 4.2c, only one contour is chosen.

- Finding the convex hull on the undistorted point set
  In Section 3.2.5, we present a formula of correcting points which can undistort contours points easily. Then we could find a smallest convex region enclosing undistorted contours points by using the algorithm proposed by Graham and Yao (1983).
- Projecting convex hull to the initial image
  We sample a set of points along each edge of the detected convex hull as shown in Figure 4.2d. Those sample points are distorted back to initial image; and the detected field boundary is visualized with the yellow lines in Figure 4.2a.

4.1.2 Obstacle Detection

Obstacle detection is not only required by robot navigation on the field, but also essential in the step of line detection. The information of size and position of obstacles enables us to eliminate some false detections of field lines.

The obstacle detection is based on the assumption that obstacles are represented as big black areas in the image. Thus, the obstacle detection depends on the black color segmentation. The detection process consists of three steps:

- Binarizing the color according to the black color ranges

  Similar to field detection, we introduce the following formula to classify the pixels

$$b_{ij} = \begin{cases} 1 & if\ (H_{ij} \in [H^b_{min}, H^b_{MAX}] \wedge S_{ij} \in [S^b_{min}, S^b_{MAX}] \wedge V_{ij} \in [V^b_{min}, V^b_{MAX}]) \\ 0 & otherwise \end{cases} \tag{4.2}$$

  where b_{ij} denotes whether the color is black or not; H^b_{min}, H^b_{MAX}, S^b_{min}, S^b_{MAX}, V^b_{min}, V^b_{MAX} are user defined black color thresholds. Figure 4.4b visualizes a binary output of black color.

- Searching regions of interest within the field boundary

  We use the same method as the one we used for finding green field boundary (Suzuki, 1985) to find connected black areas. The connected black areas show the potential locations of obstacle regions in an image.

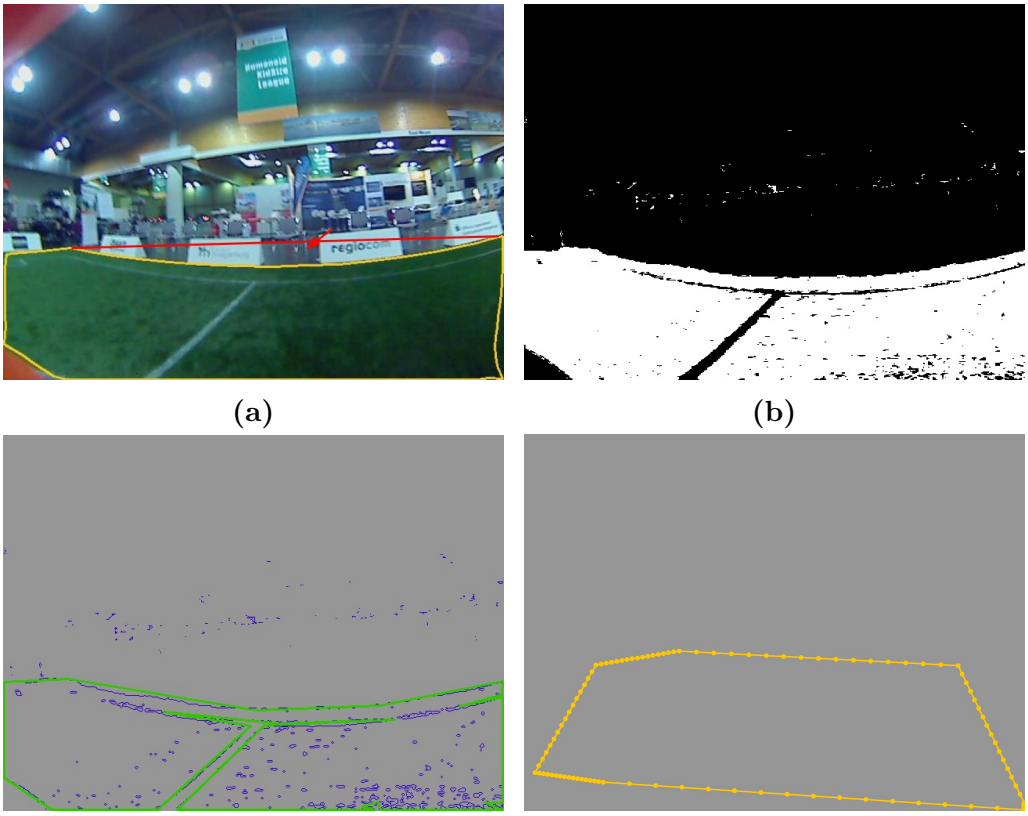
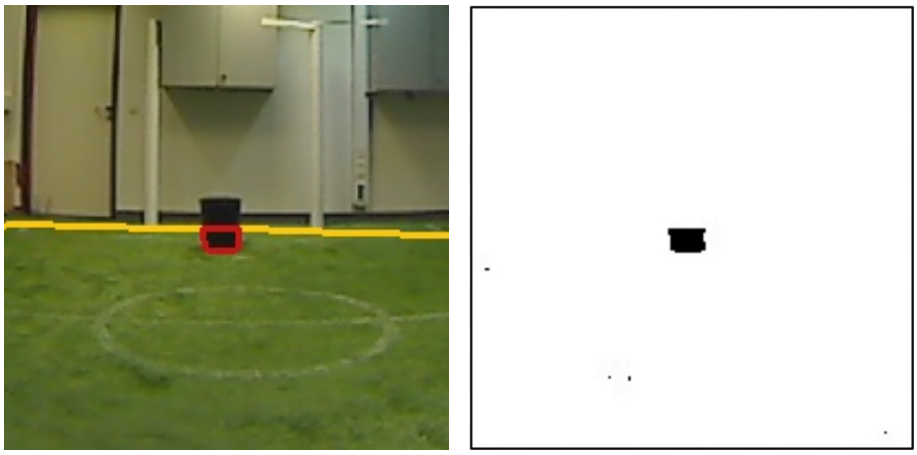


Figure 4.2: Field boundary detection. (a) Convex hull of the green regions in the image (yellow), and the unwanted area (red arrows). (b) Green binary image. (c) Contours based on segmentation. (d) Boundary points of the undistorted convex hull.

Figure 4.3: An undistorted image.



(a) **(b)**

Figure 4.4: Obstacle detection. (a) Obstacle inside the green field (red). (b) Black binary image.

- Refinement of potential obstacles

  This step aims to find a convex hull of each black region whose area exceeds a certain threshold. Black regions with small area are ignored. Figure 4.4a visualizes a convex hull of a detected obstacle.

4.1.3 Field Line Detection

4.1.3.1 Line Filters

The pixels on white lines will be relatively brighter than its neighboring green pixels. To detect field lines, we apply some line filters to the brightness channels (V channel) of the input HSV image to enhance the brightness responses of pixels on lines. In the output matrix, the higher the value is, the higher the probability of the corresponding pixel belongs to a line. Before describing our line kernels, it is necessary to briefly review the state of the art of edge detection techniques.

Among many image processing techniques, edge detection is an effective technique to identify pixels in an image in which the brightness changes sharply. A widely used method of edge detection is the Difference of Gaussians (DoG) (Marr and Hildreth, 1980). This method consists of subtracting two Gaussians (Figure 4.5), where a kernel in blue has a standard deviation smaller than the other one in green. The detected edge of this image result from the convolution between the DoG kernel in red (Figure 4.5) and the input image.

Our goal is to detect one single line that centers in two edges; however, the existed edge detection methods are not sufficient to reach this goal. In order to

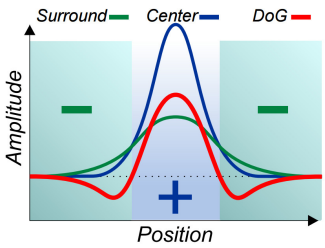


Figure 4.5: The DoG kernel.

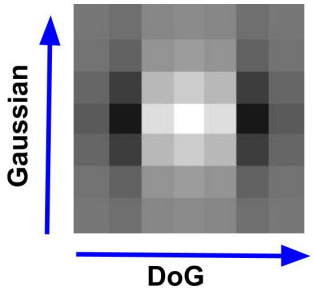


Figure 4.6: A 2D line filter for detecting vertical lines.

make the DoG kernel work for line detection, we therefore modify it as follows: changing the widths (σ_1, σ_2) of two Gaussians so that the number of positive values of DoG kernel is almost the same with the line width. By applying such DoG kernel to the brightness channel, the closer a pixel to the center of the lines, the higher response it gets.

The DoG kernel depicted in Figure 4.5 is a 1D horizontal kernel. In order to make the result more robust to noise, a vertical Gaussian kernel is applied. Thus the response of a pixel is a weighted sum of the value of itself and its neighborhoods. In Figure 4.6 depicts the resulting 2D line filter which is constructed by a 1D horizontal DoG kernel and a 1D vertical Gaussian kernel. It is noteworthy that the kernel is represented in 2D, nevertheless, the convolution can be performed much quicker on two separated 1D kernels.

In order to detect lines in different widths and orientations, we construct a set of kernels in four different orientations and three different sizes Figure 4.7. Meanwhile, we divide the input image into three parts: the bottom part, the middle part, and the upper part. Normally, when a pixel gets close to the bottom of the image, the captured object will be close to the camera as well. Therefore, at the bottom part of the image, the captured lines tend to be bigger that other parts. Hence we apply the largest four oriented kernels to the bottom part of the image, the middle sized kernels to the middle part, and the smallest kernels to the upper part.

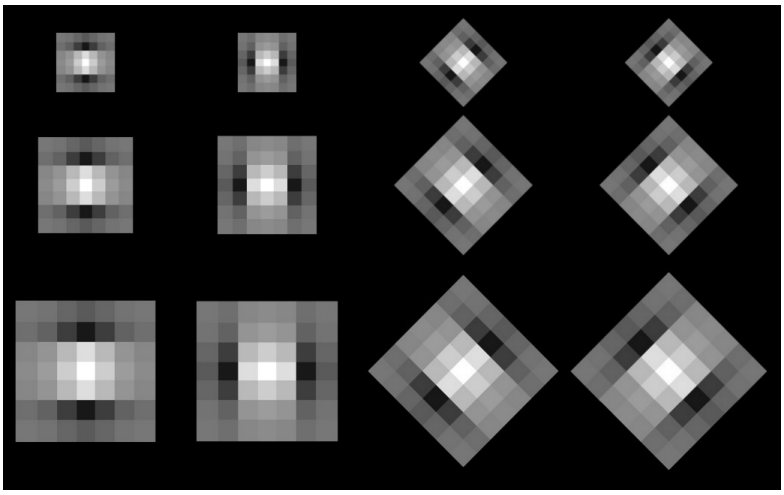


Figure 4.7: Four kernels used for detecting oriented line segments in three different sizes.

We refer brightness response image to the output of line detection on brightness channel. The higher the response is, the higher the probability of the corresponding pixel belonging to a line. For the sake of computational costs, our method only detects the lines inside the estimated field boundary. The responses outside the field boundary are all set to zero. Moreover, in order to remove some false detections, if one pixel is very close to or inside an obstacle convex hull, which is detected in Section 4.1.2, the response of this pixel is also set to zero. Figure 4.11b and Figure 4.12b visualize two resulting brightness response images.

4.1.3.2 Finding Candidate Skeleton Pixels

In the previous step, the brightness values of pixels that may be centered on lines enhanced by line filters. The step aims to find those candidate skeleton pixels that have higher value than most of their neighborhoods.

Skeletonization (Behnke et al., 1997) is used to find the optimal values among their neighbors. It introduces a simple operator which observes 3×3 pixel regions to decide if the central pixel belongs to the skeleton. For all pixels inside the estimated field boundary, the number c_{ij} of neighboring pixels (8-neighborhood) having an equal or higher brightness response than that of the center pixel is computed. If c_{ij} is less than three, which means the brightness response of the center pixel is higher than the majority of its neighboring pixels, the central pixel at position (i, j) will be considered as a candidate skeleton pixel. We classified the candidate skeleton pixels into three types according to the value of c_{ij}: type 1 ($c_{ij} = 0$), type 2 ($c_{ij} = 1$) and type 3 ($c_{ij} = 2$). Figure 4.11c and Figure 4.12c are

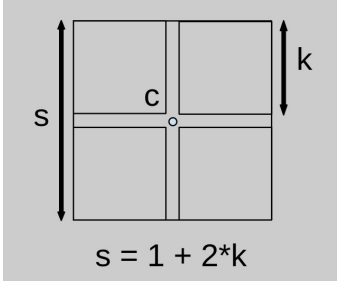


Figure 4.8: k and its relationship with side length s.

examples of the resulting candidate skeleton pixels.

4.1.3.3 Refinement of Skeleton

Skeletonization aims to approximately reduce the line width to one pixel. However, the lines are not purely white, which causes a lot of noise inside line segments. Therefore, in most cases, the line is still bigger than one pixel after skeletonization as shown in Figure 4.11c. If we construct node graph directly on the skeleton (Schulz and Behnke, 2012), many small loops inside a line segment may be formed. Therefore, a lot of effort is required for eliminating those wrong connections.

Here we propose an approach to refine the skeleton. We first fit squares to the candidate skeleton pixels, and then the centers of squares are used as purified skeleton pixels to construct a node graph.

A square is determined by its center $c(c_i, c_j)$ and side length s. Fitting squares means finding a set of $c(c_i, c_j)$ and and their corresponding s. Each skeleton pixel should be inside one and at most one square. The main steps of finding a square that containing a skeleton pixel $P(i, j)$ are introduced as follows:

Step 1: Initialization: $c = P$ and $s = 1$.

If there is only one skeleton pixel if you cut through the line, then the initial value $s = 1$ and center $c = P$ is optimal. Whereas further increasing of s is needed if there are multiple skeleton pixels along the line width.

Due to the symmetry, s is increased by two in each step. We use a parameter k to denote the length of the square in each side of the center. Thus:

- $s = 1 + 2k$;
- initial $k = 0$;
- and k is increased one by one.

Step 2: The size s stops to increase if one of the following condition is met:

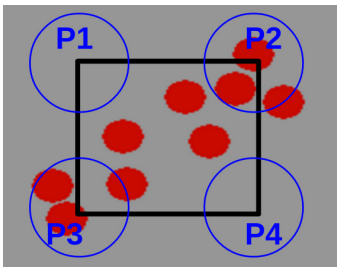


Figure 4.9: Judging a corner having a neighborhood or not: P_2, P_3 have neighborhood skeleton pixels, while P_1 and P_4 do not have. The red points are skeleton pixels.

Condition 1: At least two of its four corners have no neighborhood skeleton pixels.

Condition 2: With the increase of side length, one side of this square meets another square.

Condition 3: s exceeds the user defined maximum value of side length s_{max}.

In Condition 1, a corner P has no neighborhood skeleton pixel, which means that the Euclidean distance between P and its closest skeleton pixel P_c is larger than a user defined threshold. As shown in Figure 4.9, the radius of the circles is the distance threshold. It is obvious that the corners P_2 and P_3 have neighborhood skeleton pixels while P_1 and P_4 have no neighborhood skeleton pixel.

Condition 2 prevents the square to overlap with other squares. Furthermore, when the input skeleton contains much noise, the Condition 1 is hard to reach, thus a too large square may be created; however, Condition 3 is introduced to avoid too large squares by stopping the increases of size. Normally, if s is as large as the line width, the square is definitely able to cover all skeleton pixels along the line width. Thus s_{max} is chosen slightly bigger than the line width. As explained in Section 4.1.3.1, line detection uses different scales of kernels for detecting lines with different widths. Similarly, we also use three different user defined s_{max} for different parts of skeleton region. When the width of line become larger, a larger s_{max} should be defined.

Step 3: Shifting center c.

A square centered at c with size $s = 1 + 2k$ is found after performing first two steps. In this step, the center c shifts to one of the skeleton pixel inside the square so that the new square:

- contains maximum number of skeleton pixel;

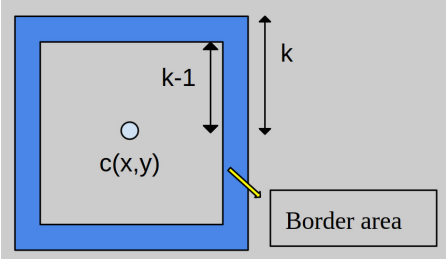


Figure 4.10: Computing the number of skeleton pixels inside a square.

- contains the initial skeleton pixel $P(i, j)$;
- and has no overlap with other squares.

In order to calculate the number of skeleton pixels efficiently, we use the similar idea behind the Integral Image (Viola and Jones, 2001) to generate a matrix of the number of skeleton pixels. The number of skeleton pixels ($Num_k(x, y)$) inside a square centered at $c(x, y)$ with size $s = (2 * k + 1)$ can be computed by adding the $Num_{k-1}(x, y)$ with the number of skeleton pixels in the border area Num_{border}: $Num_k(x, y) = Num_{k-1}(x, y) + Num_{in\_border}$ Figure 4.10.

The pseudo code of fitting squares algorithm can be found in Appendix B. Figure 4.11d and Figure 4.12c visualize two resulting squares.

4.1.3.4 Graph Representation of Field Lines

In this section, we discuss our approach for constructing node graph by connecting neighboring squares. Before constructing the neighborhood graph, some notations are introduced: in accordance with the common notation of graph theory (Bollobás, 2013), let a graph G be a pair of disjoint sets (V, E), where $V = V(G)$ is a set of vertices of G and $E = E(G)$ is a subset of the set $V \times V$. In our task, the set of vertices $V(G)$ is equal to the set of centers C of the squares found in previous step:

$$V(G) := Set_{centers} \tag{4.3}$$

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

$$E(G) := \{(v, w) \in V \times V : w \in N(v) \wedge v \in N(w)\} \tag{4.4}$$

where $N(v)$ denotes the neighborhood of node v.

Because the $V(G)$ can be easily initialized as the centers of the resulting squares, our task focus on finding the neighborhood nodes of each node, finding the set

of edges $E(G)$. Some definitions used in finding edge process are introduced as follows:

- The distance between two nodes $v_i(x_i, y_i)$ and $v_j(x_j, y_j)$ is the Euclidean distance between them.

$$dist(v_i, v_j) = \sqrt{(x_i - x_j)^2 + (y_i - y_j)^2} \tag{4.5}$$

- The neighborhood nodes of node v_i, are a set of nodes that connected with v_j.

$$N(v_i) := \{v_j | v_j \in V(G) \wedge connected(v_i, v_j)\} \tag{4.6}$$

- The candidate neighborhood nodes of node v_i are a set of nodes with a distance to v_i smaller than a user defined threshold.

$$CN(v_i) := \{v_j | v_j \in V(G) \wedge dist(v_i, v_j) < \theta_n\} \tag{4.7}$$

  where θ_n is the max distance for neighborhood nodes defined by user.

- The candidate connections of node v_i, are a set of edges between v_i and its candidate neighborhood nodes.

$$CE(v_i) := \{(v_i, v_j) | v_j \in CN(v_i)\} \tag{4.8}$$

- All candidate connections of graph G is a union set of all candidate connections of each node.

$$ACE(G) := \{(v_m, v_n) | (v_m, v_n) \in \cup_{i=1}^{N} CE(v_i)\} \tag{4.9}$$

  where N is the number of nodes of graph G.

- The weight of connecting node v_i and node v_j is evaluated by the following formula:

$$weight(v_i, v_j) = \frac{s_i * s_j * ske\_num_i * ske\_num_j * avg\_brightness\_response}{dist(v_i, v_j)} \tag{4.10}$$

  where:

  - s_i, s_j are sizes of square i and square j that centered at v_i and v_j respectively;

- ske_num_i, ske_num_j are the number of skeleton pixels inside two squares respectively;
- $avg\_brightness\_response$ is the average brightness response along the connection edge between v_i and v_j.

- The angle of node v_i and its two connected node v_j and v_k is the angle formed by connecting $v_j - v_i - v_k$. The minimum angle of node v_i is the minimum angle of node v_i and two of its connected node.

$$Min\_ang(v_i) := \{ang(v_j, v_i, v_k) | v_j, v_k \in N(v_i), ang(v_j, v_i, v_k) \leq ang(v_m, v_i, v_n), \forall v_m, v_n \in N(v_i)\} \tag{4.11}$$

For later usage, we define the $Min_{ang}(v_i)$ equals to π when the number of connected nodes of v_i, $num\_connected(v_i)$ is less than two.

By using the definitions described above, the process for finding the neighborhood nodes can be described as follows:

1. For each node v_i, find its candidate neighborhood nodes $CN(v_i)$.
2. Evaluate each candidate connection $CE(v_i)$ by using equation (4.10).
3. In each iteration, choose the candidate connection (v_m, v_n) so that:
   a) v_m and v_n is not connected;
   b) $weight(v_m, v_n)$ is highest among all weights of all candidate connections in $ACE(G)$;
   c) the degrees of v_m and v_n are less than four: $d_m < 4$ and $d_n < 4$;
   d) there is no triangle loop after connecting v_m and v_n;
   e) $Min\_ang(v_m)$ and $Min\_ang(v_n)$ are larger than an angle threshold α_{min_ang} after connecting v_m with v_n.
4. If $weight(v_m, v_n)$ is larger than a user defined threshold ω_{min}, connect v_m with v_n, go to step 3. Otherwise stop the iteration and output the node graph G.

Example node graphs can be found in Figure 4.11e and Figure 4.12d.

4.1.3.5 Line Clusters

The neighborhood graph G is able to represent the field lines in the image but not the individual field lines yet. Different field lines can be represented by a single connected region in a graph. The next field line detection step is to split the node graph into line clusters so each cluster contains only field line segments that belong to the same field line. This step is extremely helpful when doing further data registration, and more details can be found in Section 4.4.

To reach the goal that each cluster contains only field line segments that belong to the same field line or the center circle, we have some constraints for each cluster.

- A cluster consists of a set of ordered nodes.
- All nodes except ending nodes are connected with their two neighboring nodes.
- for each node v_i except two end nodes and its two neighboring nodes v_j and v_k, the angle, $ang(v_j, v_i, v_k)$ is larger than an angle threshold β_{min_ang}, which should be close to π, meaning that the neighboring edges should have similar orientations.

Due to distortion, the angles of line segments that belong to a same field line differs. Therefore, before extracting line clusters, we undistort nodes of graph G. With such an undistorted node graph UG and the constraints of cluster, a greedy algorithm can be defined to extract clusters of edges. The algorithm traverses from the first node v_i of graph G, then v_i and one of its neighborhood node v_j are added to current cluster. The node v_k is added to the current cluster if the following three constraints are met:

- v_k are connected with current node v_i;
- for any other connected nodes of v_i, say v_m, $ang(v_j, v_i, v_m) \leq ang(v_j, v_i, v_k)$;
- $ang(v_j, v_i, v_k) > \beta_{min\_ang}$

Once v_k is added to the current cluster, node v_k and v_i are disconnected; then v_j is replaced by v_i, and v_i is replaced by v_k. The algorithm continues to find the next v_k. A new cluster is created when there is no v_k satisfies the above constraints. Once a new cluster is created, the old one will be added to a list of the found clusters and its edges will be removed from the current graph as well. Figure 4.11f and and Figure 4.12e illustrate the line clusters generated by the cluster extraction algorithm. An example shows the process of cluster extraction algorithm can be found in Appendix A.

As described above, the main property of a line cluster is that all nodes in the same cluster belong to the same field line or center circle.

4.2 From Visual Features to Observations

The output of previous line detection process is a set of node clusters. Each cluster is a set of ordered nodes which are connected with their one or two neighboring nodes. The neighboring edges have similar orientations.

Our purpose is to associate each detected nodes to a 3D model line point with high accuracy. To make it more efficient when doing the data registration, we extract following high lever information for each observed cluster.

- The orientation of each node v_i, $ort(v_i)$, is the average orientation of its neighboring edges.

$$ort(v_i) = \begin{cases} ort(v_j, v_i) & if\ \{v_j\} = N(v_i) \\ avg(ort(v_j, v_i), ort(v_i, v_k)) & if\ \{v_j, v_k\} = N(v_i) \end{cases} \tag{4.12}$$

  where $avg(ort_1, ort_2)$ calculates the average orientation of angle ort_1 and ort_2. When calculating the average angle of two angles, it is worth mentioning that the two angles should be normalized to the range $[-\pi, \pi]$ before doing the numerical average of these two angles. We consider the orientation of each node is undirected, for example, 0 and π are considered as the same orientations. So $ort(v_i)$ is normalized to the range $[-0.5\pi, 0.5\pi)$.

- The angle of cluster m, avg_ang_m, is the average angle of all nodes of this cluster:

$$avg\_ang_m = avg(ort(v_1), ort(v_1), \ldots, ort(v_n)) \tag{4.13}$$

  where n is the number of nodes of cluster m. It is noteworthy that it is necessary to pre-process the orientation of each node before calculating the numerical average of these angles.

- The weight of each node, $weight(v_i)$, is the sum of half length of its two neighborhood edges:

$$weight(v_i) = \begin{cases} \frac{1}{2} dist(v_j, v_i) & if\ \{v_j\} = N(v_i) \\ \frac{1}{2} dist(v_j, v_i) + \frac{1}{2} dist(v_j, v_k) & if\ \{v_j, v_k\} = N(v_i) \end{cases} \tag{4.14}$$

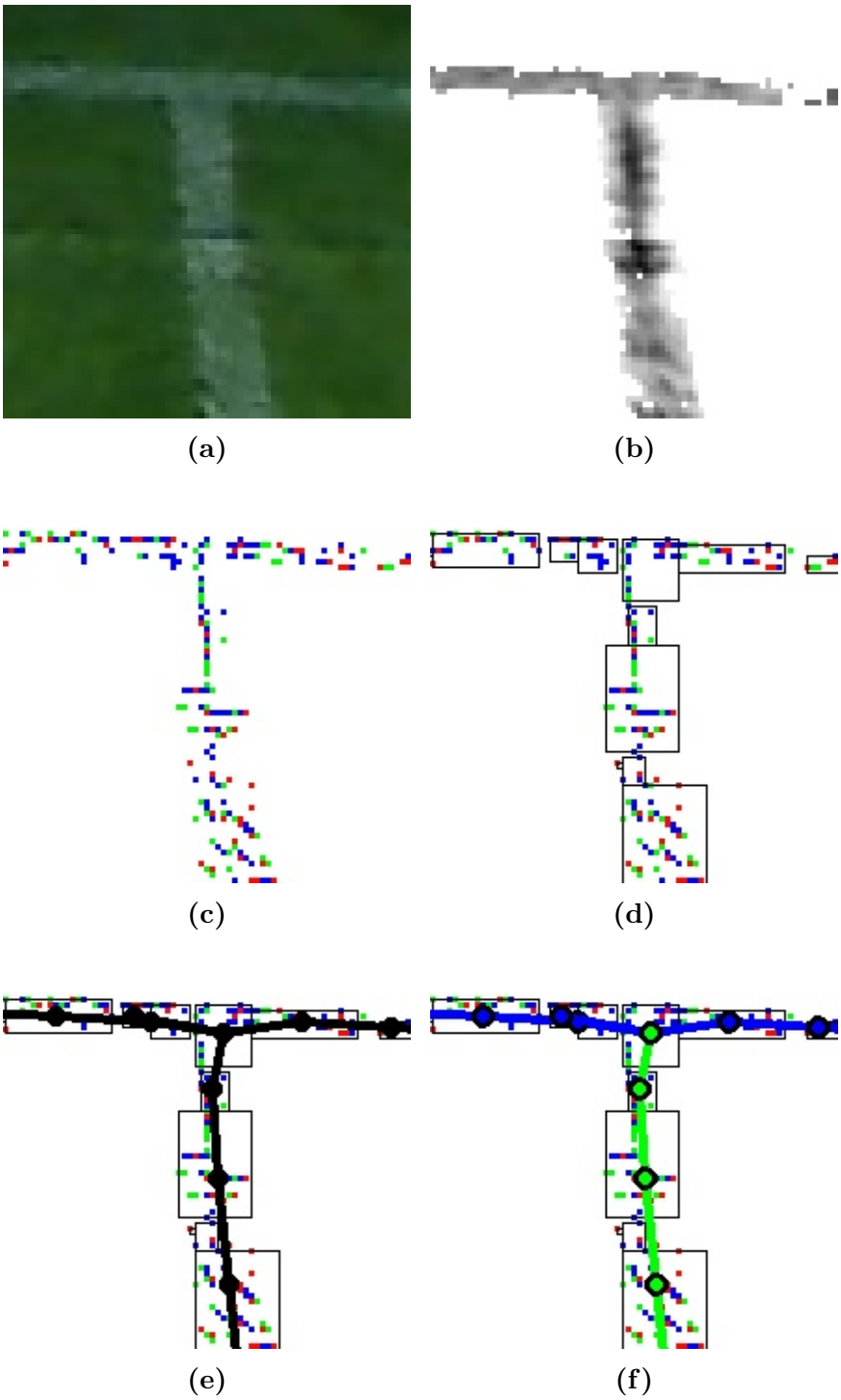
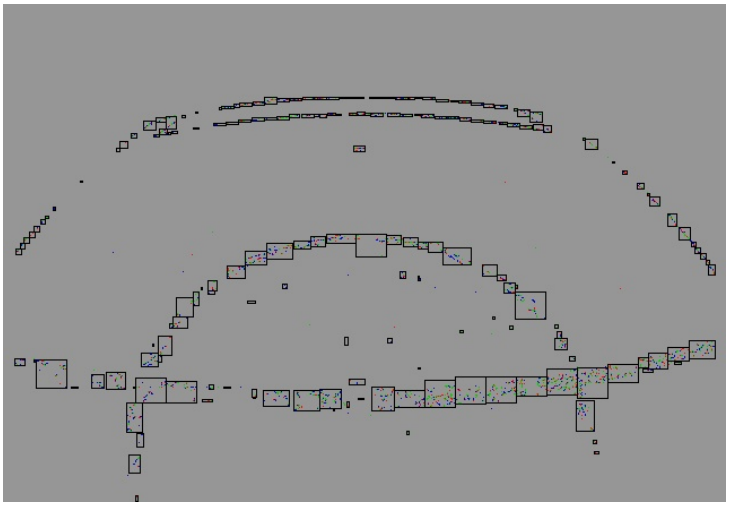


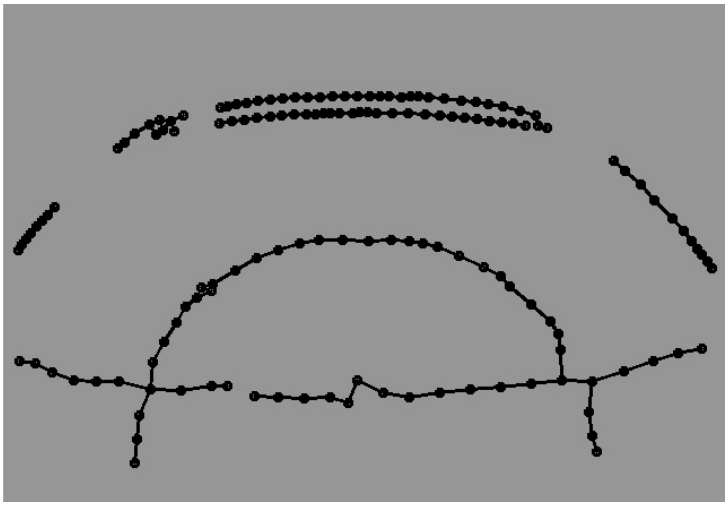
Figure 4.11: An example of field line detection process. (a)Input image. (b) The output brightness response image after applying line filters. (c) Three types of optimal values including: type 1 (red), type 2 (green) and type 3 (blue). (d) Bounding squares found from the optimal values. (e) A node graph constructed by connecting neighborhood squares. (f) Two clusters (in green and blue) extracted from node graph.

(a)

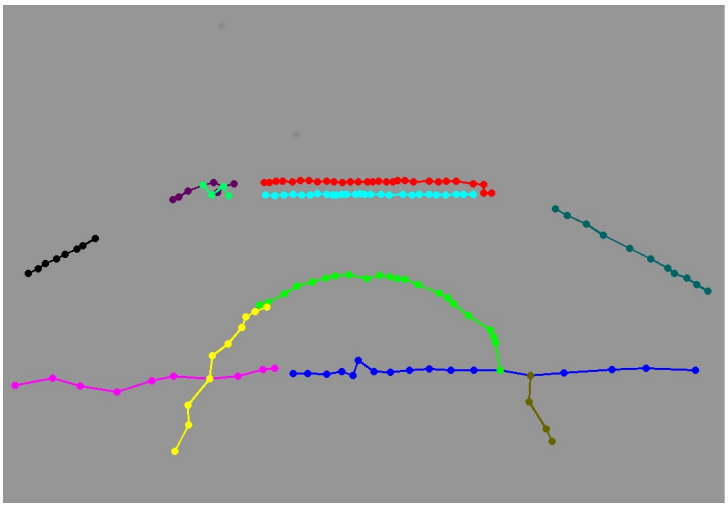
(b)



(c)



(d)



(e)

Figure 4.12: Visualization of field line detection process. (a) Input image. (b) Resulting gray image after applying line filters on brightness channel. (c) Skeleton pixels and the bounding squares. (d) A node graph. (e) Resulting clusters found from the undistorted node graph.

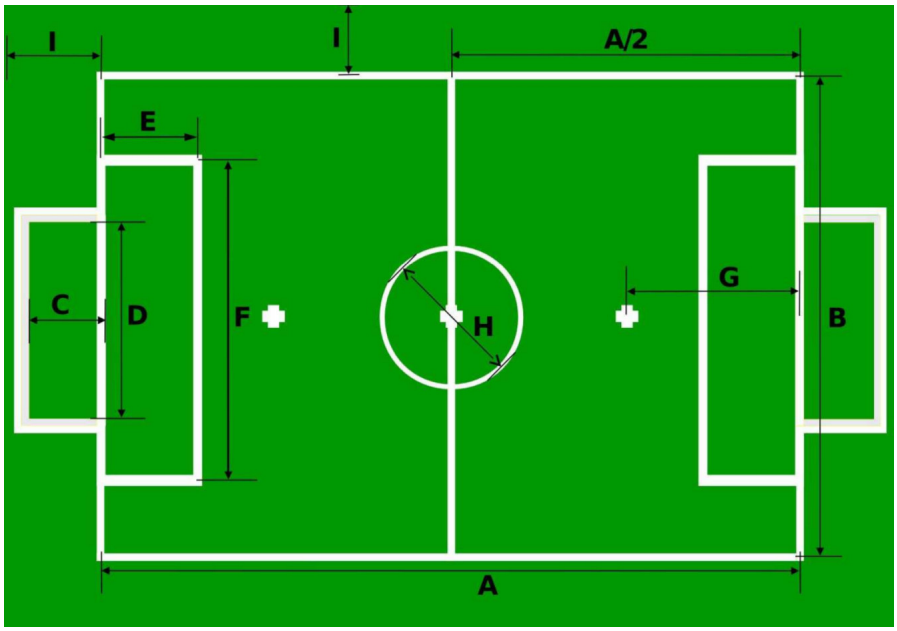


Figure 4.13: Humanoid robot soccer field (not to scale).

As a result, the observations can be represented as follows:

$$Obs = \{cluster_m | cluster_m = (V,\ ort,\ avg\_ang,\ Weight), m = 1, 2, ..., n\} \quad (4.15)$$

where n is the number of clusters; $V(cluster_m)$ are all nodes belong to $cluster_m$; $ort(cluster_m)$ are orientations of nodes; avg_ang is the average angle calculated from equation (4.13) ; $Weight(cluster_m)$ are weights for nodes.

4.3 Modeling the Field Lines

In this section, we discuss our method for projecting the 3D field line model to the image plane. The humanoid league competitions take place in a well defined environment. As shown in Figure 4.13, the field model consists of two goals and field lines (Soccer, 2015), which has been pre-known to the soccer robots. Table 4.5 shows the dimensions of the soccer field for RoboCup TeenSize competition and the soccer field for test in our laboratory.

In order to associate our observations to the 3D model, it is necessary to determine which parts of lines are visible and where they are located in an image plane. For this purpose, we project the 3D field models to the image plane. In this process, we only leverage the filed lines painted on the grass for tracking. The projection of field lines can be decomposed into three main steps:

Dimension labels	Dimensions	SF	SFL
A	Field length	900	545
B	Field width	600	410
D	Goal width	260	260
E	Goal area length	100	60
F	Goal area width	500	340
G	Penalty mark distance	210	130
H	Center circle diameter	150	120

Table 4.1: Dimensions of the rectangular field of soccer play (in cm), and SF is short for Humanoid robot soccer field for TeenSize competition; SFL is short for Soccer field in Laboratory

Step 1: Sampling points on the field lines

As shown in Figure 4.14, (O, x, y, z) is the world coordinate system, with the origin located on the center of the field. We sample a set of points M on field lines uniformly. The coordinate of each 3D point is denoted as $M_i = [X_i, Y_i, Z_i]^T$. It is obvious that the Z value of each point is 0 because only field lines on the ground are used for tracking.

Step 2: Projecting each 3D point to 2D image plane and removing those projected points outside image boundary

A 2D projected model point set m is generated by projecting each point M_i to m_i. To project a 3D point $M_i = [X_i, Y_i, Z_i]^T$ to its 2D corresponding point $m_i = [v_i, u_i]^T$, we use the perspective projection model described in Section 3.1.1. The external parameters matrix $[R|t]$ is calculated from the current estimated camera pose. In our system, the estimated camera pose at time t, s_t is denoted as the following 6-tuple:

$$s_t : ((x, y, z, \gamma, \beta, \alpha))_t^T \tag{4.16}$$

The translation vector can be acquired immediately from the position, that is $t = (x, y, z)^T$, and the corresponding rotation matrix R can be acquired from the Euler angles by equation (3.7).

In order to check whether m_i are visible in captured distorted image, we calculate the distorted point $m'_i = [v'_i, u'_i]$ of m_i . If m'_i is outside the rectangle of captured image ($v'_i \notin [0, 640], u'_i \notin [0, 480]$), m_i is removed from set m.

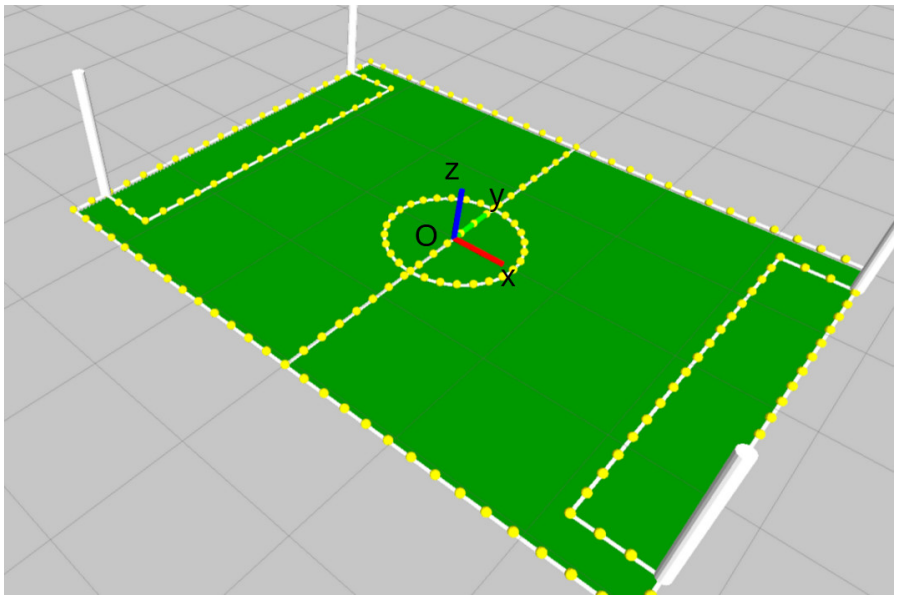


Figure 4.14: (O, x, y, z) is the world coordinate system. The yellow points are 3D points sampled on field lines.

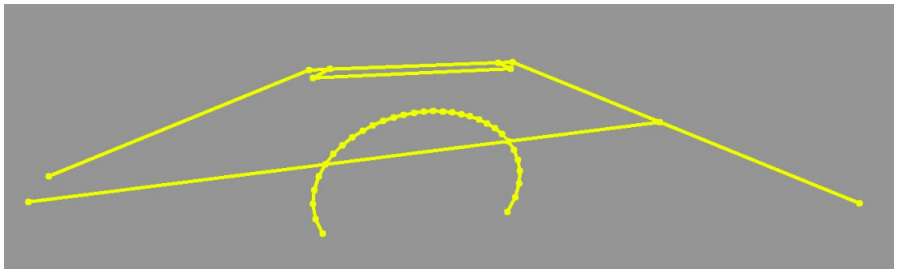


Figure 4.15: A 2D Projected field model.

Besides, for those points belongs to a same straight field line, only two ending points are kept. Therefore, only one single line segment will be generated for a straight field line. The removing of points inside a straight line improves the efficiency of data registration, which will be introduced in Section 4.4. However, by connecting the neighborhood points, a center circle consists of several line segments.Figure 4.15 shows a resulting projected model.

Step 3: Normalization of the representation of the projected model

For later usage, the projected model is denoted as a set of elements. Each element is a set of line segments.

$$P_m = \{E_m | E_m = Lines_m,\ m = 1, 2, \ldots, n\} \tag{4.17}$$

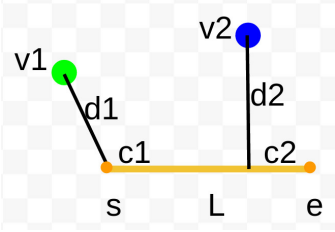


Figure 4.16: Examples of showing the distance of a node to a line segment: the closest point on line segment L to $v1$ is $c1 = s$, so $dist(v1, L) = \| v1 - c1 \|_2$: while the closest point on line segment L to v2 is the projected point $c2$, so $dist(v2, L) = \| v2 - c2 \|_2$.

where n is the number of elements. The element corresponding to a straight field line consists of only one single line, while the one corresponding to the center circle may include several line segments. For each line segment, L_k, it contains the following information:

- the starting point s_{L_k} and the ending point e_{L_k};
- and the angle of this line segment ang_{L_k}, which is normalized to $[-0.5\pi, 0.5\pi)$.

4.4 Data Registration

After obtaining observations and a projected model, we need to assign the observed nodes to the points on the projected model, at the same time assign them to points on 3D field. Some definitions used in data registration process are introduced as follows:

- The distance of a node v_i to a line segment L_j is the Euclidean distance between v_i and c_i, where $c_i$is the closest point to v_i on the line segment L_j.

$$dist(v_i, L_j) = \| v_i - c_i \|_2, \ where\ c_i = argmin_{p_i}(\| v_i - p_i \|_2), p_i \in L_j \tag{4.18}$$

- The distance of a node v_i to a model element E_m is the distance between v_i and L_j, where L_j is the closest line segment to v_i among all line segments of E_m.

$$dist(v_i, E_m) = dist(v_i, L_k), , \ where\ L_k = argmin_{L_j} dist(v_i, L_j), L_k \in E_m \tag{4.19}$$

- The distance between an observed cluster $cluster_n$ and a model element E_m is the weighted average distance of each node in $cluster_n$ to E_m.

$$dist(cluster_n, E_m) = \sum_{i}^{n}(dist(v_i, E_m) \cdot weight_i) / \sum_{i}^{n} weight_i, \qquad (4.20)$$

  where n is the number of nodes in $cluster_n$.

The purpose of data registration is to find the closest point c_i from model elements for each node v_i of observation clusters. Because the nodes in a same cluster is supposed to belong to a same model element, instead of finding the closest model point of each node, our approach finds the closest model element E_m of each $cluster_n$ that meets the following conditions:

Condition 1: The distance between $cluster_n$ and E_m is minimum:

$$E_m = argmin_{E_o} dist(cluster_n, E_o), E_o \in P_m \qquad (4.21)$$

Condition 2: The angle difference between the orientation ort_i of each node v_i in $cluster_n$, and the orientation of the line segment L_j that is closest to v_i in E_m should be smaller than a user defined threshold :

$$angDiff(ort_i, ang_{L_j}) < \delta_{max\_dif} \qquad (4.22)$$

where δ_{max_dif} is the maximum difference. With this condition, a node can only be assigned the line segment with similar orientations.

If the distance between $cluster_n$ and its closest model element E_m is smaller than a user defined threshold $dist_{inlier}$, then nodes in $cluster_n$ are considered as inliers; otherwise they are considered as outliers. For each inlier node v_i, its closest point c_i can be obtained in the process of finding the closest model element E_m of each $cluster_n$. Figure 4.17 shows an example of associating observed node v to projected model point c.

The 3D model point $C_i = [X_i, Y_i, Z_i]^T$ corresponding to $c_i = [u_i, v_i]^T$ can be calculated by the back projection process. Specifically, with the prior knowledge that $Z_i = 0$, the value of X_i and Y_i can be acquired by solving equation (3.1). Thus a pair of correspondence from 2D point v_i to 3D model point C_i is formed.

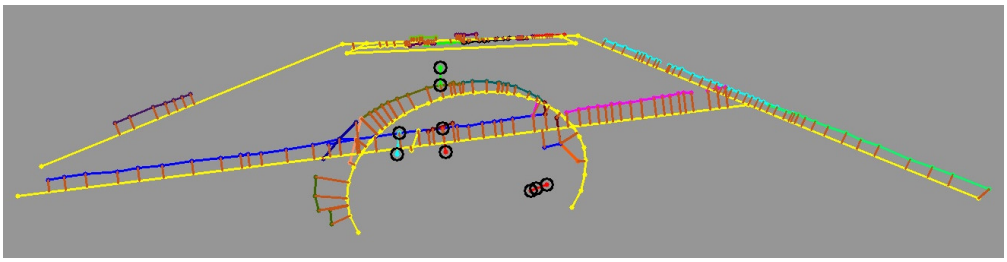


Figure 4.17: Data registration based on current estimated camera pose: the projected model is represented by yellow line segments; the observations are visualized using different colors for different clusters; each observed inlier node v_i is connected to its closest point c_i by a line segment in orange; nodes in black circles are identified as outliers.

4.5 State Optimization

In Section 4.4, we introduce our approach to set the correspondences between 2D points v and 3D model points C. Once the 2D-3D correspondences are known, the estimated camera pose can be updated to minimize the sum of reprojection errors between projected 3D points and their corresponding observed points in 2D image plane (see Section 3.2.2).

$$[R|t] = argmin_{[R|t]} \sum_{i}^{N} dist^2(K[R|t]\widehat{C}_i, \lambda\widehat{v}_i) \tag{4.23}$$

To solve this equation, both EPnP (see Section 3.2.3) and hill climbing method (see Section 3.2.4) are implemented in our system. The efficiency and effectiveness of these two methods will be evaluated by experiments in Chapter 5. As we all known, one drawback of non-iterative approaches (e.g. EPnP) for pose estimation is that they are rather sensitive to noise. Therefore, EPnP is often used in combination with RANSAC (see Section 3.2.5).

When using the hill climbing method, the error function to be minimized is the reprojection error as follows:

$$E(S_t) = \sum_{i}^{N} dist^2(K[R|t]\widehat{C}_i, \lambda\widehat{v}_i) \equiv \sum_{i}^{N} dist^2(\widehat{c}_i, \widehat{v}_i) = \sum_{i}^{N} dist^2(c_i, v_i) \tag{4.24}$$

where s_t denotes the current estimated camera pose. Equation (4.24) shows that the error is the sum of square distance between node v_i and corresponding projected model point c_i. In our approach, each node is assigned with a weight.

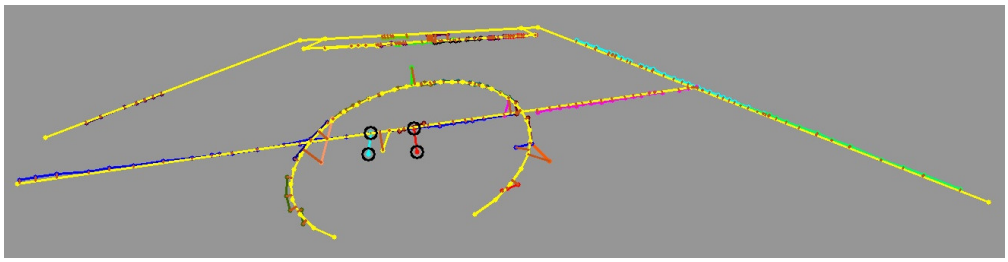


Figure 4.18: Data registration based on the camera pose after optimization.

Therefore the error $E(S_t)$ is the weighted sum of square distance between v_i and c_i:

$$E(S_t) = \sum_{i}^{N} dist^2(c_i, v_i) * weights_i / \sum_{i}^{N} weight_i \tag{4.25}$$

The main steps of hill climbing method are described as follows:

Step 1 Sampling a set of neighborhood states S' of s_t.

The neighborhood pose of s_t is generated by adding/subtracting a value η to/from one of the six dimensions.

Step 2 Evaluating each neighborhood state s'_i

For each neighborhood state s'_i, the data registration algorithm is applied to find the correspondences. Then the reprojection error is calculated according to equation (4.25). Finally we choose the best neighborhood state s'_k, having minimum reprojection error.

Step 3 Updating s_t to s'_k if $E(s_t) > E(s'_k)$; otherwise decreasing the step size: for example $\eta = 0.5\eta$, then going to Step 1.

The iteration stops if one of the following conditions meet: 1) $\eta < \eta_{min}$; 2) $E(s_t) < E(s'_k)$; 3) the iteration number exceeds the maximum iteration number.

Figure 4.18 visualizes the data registration result based on an optimized camera pose using the same observations with the one presented in Figure 4.17 . Obviously, in Figure 4.18 the observed nodes almost overlap with the projected model, which means that the sum of reprojection errors of the optimized camera pose is much smaller than the one presented in Figure 4.17 based on a estimated camera pose.

4.6 Multi-hyphothesis for Tracking

Even through our visual tracking approach is designed to track the camera pose all the time, the robot may lose track of the camera pose in some extreme cases. For example, after a period of time, there are no observations when the robot looks outside the field. As a consequence, the estimated pose s_t may be far away from the actual pose. The resulting correspondences generated based on such bad estimated state may contain many outliers. Thus, it is impossible for a robot to find an optimal pose based on such correspondences.

Instead of only tracking the pose, our vision system follows a multi-hypothesis approach to recover the optimal camera pose after getting lost. This approach generates multiple hypotheses around the current estimate pose s_t once the probability of being lost is high. The best hypothesis is chosen to be the next estimated state. Now, we introduce the definition for probability of being lost and how to generate hypotheses according to this probability.

The probability of being lost is closely related to the average reprojection error and the percentage of inliers. High average reprojection error is a strong indicator for being lost. Moreover, a low percentage of inliers, which means that the observations and the projected model are quite different or far away, is also a sign for being lost. Thus we combined these two values to be a normalize average error:

$$E(s_t)^N = \frac{\sqrt{E(s_t)}}{N_{inliers}} / \frac{N_{inliers}}{N} \tag{4.26}$$

where $E(s_t)^N$ denotes the normalized average error; $N_{inliers}$ is the number of inliers and N is the number of all observed nodes. The probability of being lost is calculated as follows:

$$p_{lost}(s_t) = \begin{cases} 1 & E(s_t)^N > \epsilon_2 \\ (E(s_t)^N - \epsilon_1)/(\epsilon_2 - \epsilon_1) & \epsilon_1 < E(s_t) < \epsilon_2 \\ 0 & E(s_t)^N < \epsilon_1 \end{cases} \tag{4.27}$$

where ϵ_1 , ϵ_2 are user defined error thresholds and $\epsilon_1 < \epsilon_2$.

Both the number of hypotheses n_t and sample range depend on the value of $p_{lost}(s_t)$:

$$n_t = p_{lost}(s_t) * N_{max\_h} \tag{4.28}$$

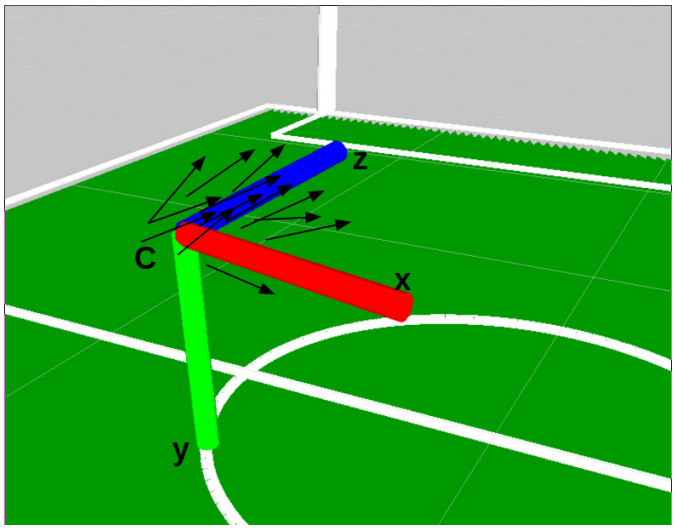


Figure 4.19: The (C, x, y, z) is the camera coordinate system, with z-axis pointing to the center of captured images. The origin of this coordinate is the estimated camera position. The black arrows are hypotheses generated around the current state.

where N_{max_h} is a user defined maximum number of hypotheses.

$$\begin{cases} range_x = p_{lost}(s_t) \cdot (0.5A) \\ range_y = p_{lost}(s_t) \cdot (0.5B) \\ range_z = p_{lost}(s_t) \cdot (0.5h) \\ range_\gamma = p_{lost}(s_t) \cdot (0.5\pi) \\ range_\beta = p_{lost}(s_t) \cdot (0.5\pi) \\ range_\alpha = p_{lost}(s_t) \cdot (0.5\pi) \end{cases} \tag{4.29}$$

where A is the field length; B is the field width; h is the height of the camera when the robot stands erectly. For each dimension, we sample a value between $[-range, range]$ and this value is added to the corresponding dimension of pose s_t. The output pose s'_t is truncated so that it is located inside the soccer field ($x \in [-A-0.2, A+0.2]$, $y \in [-B-0.2, B+0.2]$) and the height z is in the range $[0, h]$. The truncated state of s'_t is added to hypothesis list.

The hypothesis with minimum error is chosen as the optimal state estimate. Figure 4.19 visualizes a set of hypotheses generated around state s_t.

4.7 Probabilistic State Estimation

To stabilize the pose estimate, the output pose from Section 4.5 and Section 4.6 is smoothed with a specialized Kalman Filter. We use a normal Kalman filtering approach which has been introduced in Section 3.4.2. In the context of 6-DoF camera pose tracking, in addition to the rotation and translation parameters, the state vector s_t includes additional variables such as the translational and angular velocities. The motion model used in prediction step and the covariance matrix used in correction step are introduced as follows:

In our approach, the translational velocities of $(v_x, v_y, v_z)_t^T$ and angular velocity Δq_t (in quaternion form) are retrieved from IMU and dead reckoning. The position is updated as follows:

$$(x, y, z)_{t+1}^T = (x, y, z)_t^T + (v_x, v_y, v_z)_t^T \times \Delta t \tag{4.30}$$

The calculate the new orientation, we convert the orientation from the Euler angles $(\gamma, \beta, \alpha)_t^T$ to quaternion q_t. As we discuss in Section 3.2.1 the multiplication of quaternions represents composing the two rotations. Thus the new orientation q_{t+1} is calculated as follows:

$$q_{t+1} = \Delta q_t \cdot q_t \tag{4.31}$$

In correction step, estimating the covariance matrix of the measurements plays an important role. In our system, the approach proposed by (Bengtsson and Baerveldt, 2003) is used. This approach estimates the covariance by examining the shape of the error function. The idea is that the error function $E(s_t)$ was a quadratic, then the covariance of optimal least-squares estimate equals to:

$$cov(\widehat{x}) = \frac{1}{2}\frac{\partial^2}{\partial \widehat{x}^2} E(s_t)^{-1}\sigma^2 \tag{4.32}$$

An unbiased estimate s^2 of σ^2 in equation (4.30) would be $s^2 = E(s_t)/(N-3)$, where N is the number of correspondences. The final expression for the covariance estimate is:

$$cov(\widehat{x}) = 2\frac{E(s_t)}{(N-3)}\frac{\partial^2}{\partial \widehat{x}^2} E(s_t)^{-1} \tag{4.33}$$

In our system, the calculation of $\frac{\partial^2}{\partial \widehat{x}^2} E(s_t)$ is as follows:

$$\frac{\partial^2}{\partial \widehat{x_i}^2} E(s_t) \approx 2E(s_t) - E(s_t + \Delta x_i) - E(s_t - \Delta x_i), i = 1, 2, \ldots, 6. \tag{4.34}$$

where Δx_i is a 6-dimension vector, with the value in $i-$th dimension to be a small value η and the value in other dimensions to be 0.

Until now, the normal Kalman filtering mechanism can be applied.

4.8 Summary

In this chapter, we introduce each part of our visual tracking system step by step. Specifically, we first propose a kernel-based method for detecting line candidate pixels inside the field boundary. And then from those candidate pixels, we find a purified skeleton for constructing node graph, which is divided into different line clusters in the next step. A good observation model is generated by extracting high level information from these line clusters.

Then we introduce a method for setting up a set of 2D-to-3D point correspondences between the observations and the 2D projected field model. For solving the Perspective-n-Point (PnP) problem, two optimization techniques, hill climbing method and EPnP are introduced to find a optimal 6-DoF camera pose. To make our tracking process more robust, we also introduce a multi-hypothesis approach which enables the robot to recover the optimal camera pose after losing its track. In the end, to stabilize the pose estimate, the Kalman filter is applied for state fusion.

5 Experiments

Our visual tracking system is built on Robotic Operating System (ROS) (Saito, 2016) which provides libraries and tools to facilitate the development of robot applications. ROS provides an environment where developer can easily integrate different modules and visualize the simulation scene on a Linux platform.

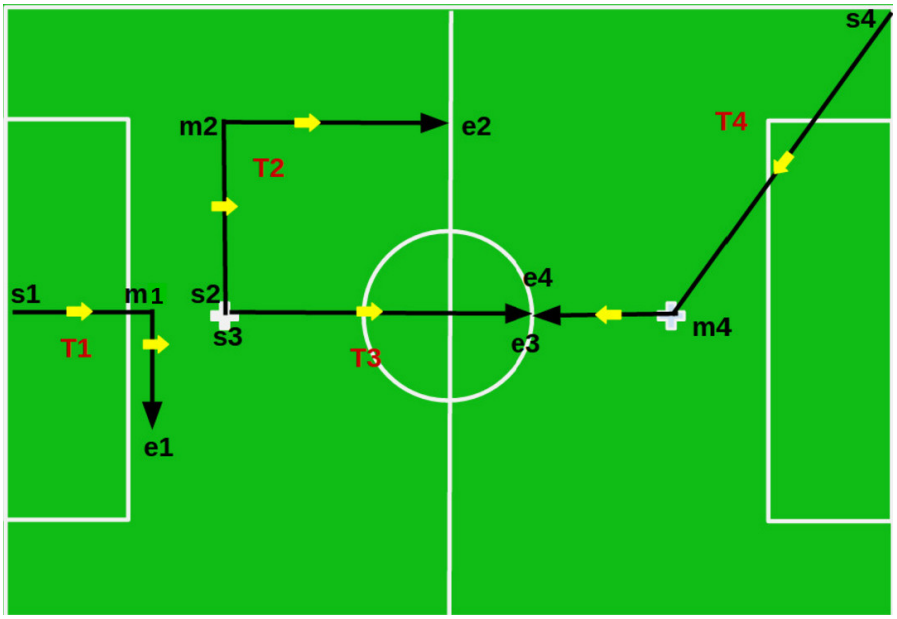
We conduct several experiments to evaluate our visual tracking system from three main aspects, computational performance, object detection performance, and visual tracking accuracy. Due to time constraints, it is not possible to experiment on our system in a real RoboCup environment. Thus all experiments were performed in a simulated soccer field in our lab, the field dimensions of which can be found in Table 4.1.

5.1 Apparatus

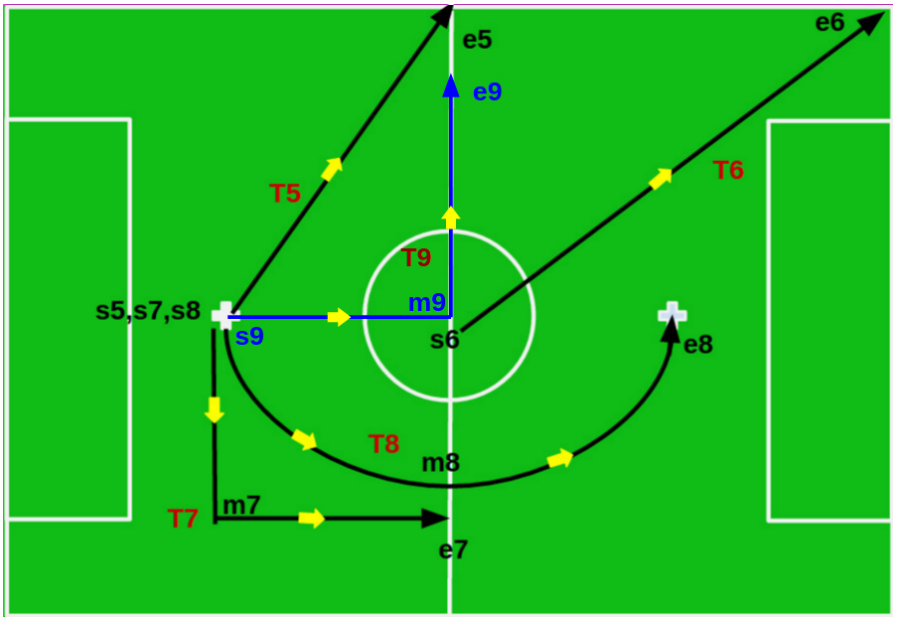
We conducted all experiments using igus® Humanoid Open Platform (Allgeuer et al., 2015). This robot is nominally equipped with two 720p Logitech C905 USB cameras. Each camera is fitted with a 150° Field of view(FOV) wide-angle lens. We used one of the cameras to capture images in RGB format at a resolution of 640×480 by using the Video4Linux2 library. The igus® Humanoid Open Platform was only used to capture images and IMU data. Further image processing and pose estimation were performed on a PC, which has a computer with a Intel® Core^{TM} i5-4210U CPU @ 1.70GHz $\times$ 4 processor and 8GB of memory.

5.2 Experiment Design

We designed nine different tasks to evaluate the system from the three aspects. In particular, in each of the nine tasks, we recorded the images, dead reckoning and IMU data published by the robot while walking along the pre-defined trajectories (Figure 5.1) controlled by joystick. Most of the time, the head of robot was straight to the front, which is identified by the yellow arrows in Figure 5.1. Thereby it ensures that the captured scenes are roughly as what we expected.



(a)



(b)

Figure 5.1: Nine pre-defined tasks: (a) T1 $\sim$ T4; (b) T5 $\sim$ T9. Each task contains a trajectory which has a starting point (s) and an ending point (e). Some of the trajectories also have a midpoint (m). The black/blue arrows indicate the direction of trajectory while the yellow arrows indicates the front direction of the trunk.

It is common to employ pre-defined trajectories as the ground truth for evaluating localization accuracy (Strasdat et al., 2006). However, the error of controlling the robot along the trajectories by joystick may lead to moving errors. Hence we marked the trajectories according to robot moving on the soccer field in videos recorded by a second camera outside the field. Consequently, we used the marked trajectories as the ground truth in our experiments.

The datasets includes about 8,000 images captured in different field locations and orientations. Many captured images were affected by motion blur due to walking and head panning motions. We analyzed these nine recorded datasets on the aforementioned PC.

Firstly we examined computational performance by measuring average execution time of each part of the proposed visual tracking systems in nine tasks. For each task, the initial estimated pose was set to a pose within 0.2 meter to the actual start pose. Each task was repeatedly performed with three different optimization methods: hill climbing method, EPnP and EPnP+RANSAC. All the three methods used the Kalman filter (KF) for state fusion by default.

In addition, we also evaluated the performance of the object detection of our system both qualitatively and quantitatively. In order to test the object detection system, 200 frames of images were selected randomly from all recorded datasets. We logged all extracted field contours, obstacle contours and clustered lines of these 200 frames. The data was evaluated frame by frame to calculate the true positive rate (TPR) and positive predictive value (PPV).

$$\begin{cases} TPR = TP/(TP + FN) \\ PPV = TP/(TP + FP) \end{cases} \tag{5.1}$$

where TP, FP, FN denote the number of true positives, false positives and false negatives respectively. Below shows the criteria for calculating the TP, FP and FN, where *Detected_object* indicates an output feature(field boundary, obstacle, or lines) of our vision system and *True_object* indicates a actual feature identified by us.

TP : The objects that are correctly identified. For a *True_object*, it is correctly identified if at least 90% of it has been detected.

FP : The objects that are incorrectly identified. For a *Detected_object*, it is incorrectly identified if at least 90% of it doesn't belong to any *True_object*.

FN : The objects that are incorrectly rejected. For a *True_object*, it is incorrectly rejected if at most 10% of it has been detected.

Task group	Tasks	Task difficulty level	Interpretation
NT group	T1 - T4	normal	visible field pct. $> 25\%$
DT group	T5 - T8	difficult	visible field pct. $< 25\%$

Table 5.1: Two groups of experiments

Finally, we evaluate the visual tracking accuracy of our system. The visual tracking accuracy is measured by the distance error, which is the Euclidean distance between the estimated position (x, y) and the actual position. However, since it is difficult to generate a ground truth for the height z and the orientations of the camera without motion capture system, we only evaluated the distance error in x and y coordinates. The ground truth for x and y coordinates are marked by us after checking the videos recorded by the camera outside the field.

We evaluated the visual tracking accuracy from the flowing three aspects:

- Comparing three different optimization methods tested in our system.
- Comparing the system using Kalman filter with the system without using Kalman filter.
- Comparing our system with the system proposed by Farazi, Allgeuer, Ficht, et al. (2016) which optimizes 3D of the pose $(x, y, \alpha)^T$ while the other 3D of the pose state (z, θ, ϕ) are retrieved from Dead Reckoning.

To compare the visual tracking accuracy of three different optimization methods tested in our system, the first eight tasks (T1-T8) were performed. Prior to the evaluation, according to task difficulty level we categorized the first eight tasks into two groups, normal task (NT) group, and difficult task (DT) group. The difficulty level is defined by the visibility of soccer field in the images captured by the camera of robot. NT group has at least 25% of the soccer field which is visible; while DT group has at most 25% of the soccer field which is visible Table 5.1. The two groups of taks are demonstrated in Figure 5.1, where the trajectory and facing directions of the robot's trunk for each tasks are shown.

Additionally, to investigate the effects of the Kalman filter on the tracking accuracy, we performed two experiments on T9 test data by using the visual tracking system with EPnP+RANSAC for pose optimization. We did not use the Kalman filter for state fusion in the first experiment, while we used it in the second experiment.

Finally, we still use T9 test data to compare the accuracy of our system with that of the system proposed by Farazi, Allgeuer, Ficht, et al. (2016). In this

experiment, task T9 was performed once again with using Farazi's approach for 3D pose localization.

5.3 Result

5.3.1 Computational Performance

Figure 5.2 shows the average processing time for each algorithm for pose optimization. The execution time of line detection and pose optimization are significantly longer than the other parts. In fact, our line detection method takes longer time than the method proposed by Schulz and Behnke (2012). One of the reasons is that unlike their method performing the line detection on a sub sampled images ($\frac{1}{8}$ size of the original images), our approach performs line detection on the original images to prevent losing essential information in sub-sampling process. The other reason be that convoluting line filters on the image is more expensive than their color-table based approach.

In pose optimization step, the non-iterative approach EPnP (average time: 18.84 ms) and the approach of EPnP within RANSAC schema (average time: 25.28 ms) are much faster than an iterative approach, hill climbing method (average time: 50.01 ms). Hill climbing method is slow because the current 6-DoF pose optimization generates a great number of neighborhoods and the evaluation of each neighborhood takes time; moreover, the data registration is performed iteratively according to the updated estimated pose during each iteration of this algorithm.

5.3.2 Object Detection Performance

Table 5.2 shows true positive rate (TPR), positive predictive value (PPV) and the number of features (N) that should have been recognized. The PPV of field boundary and obstacle detection is 1.0, which indicates that there is no false positives in the detection results. The TPR of field boundary and obstacle detection is very high, which means that most of the field boundaries and obstacles are detected correctly. The PPV of field line detection is 0.90, which indicates that the detected results contain some false positives. The overall TPR of line detection(0.52) is low because the lines are marked as reference even though they are barely perceivable for a human. However, our line detection still gains higher TPR than the TPR (0.45) of the system proposed by Härtl et al. (2013).

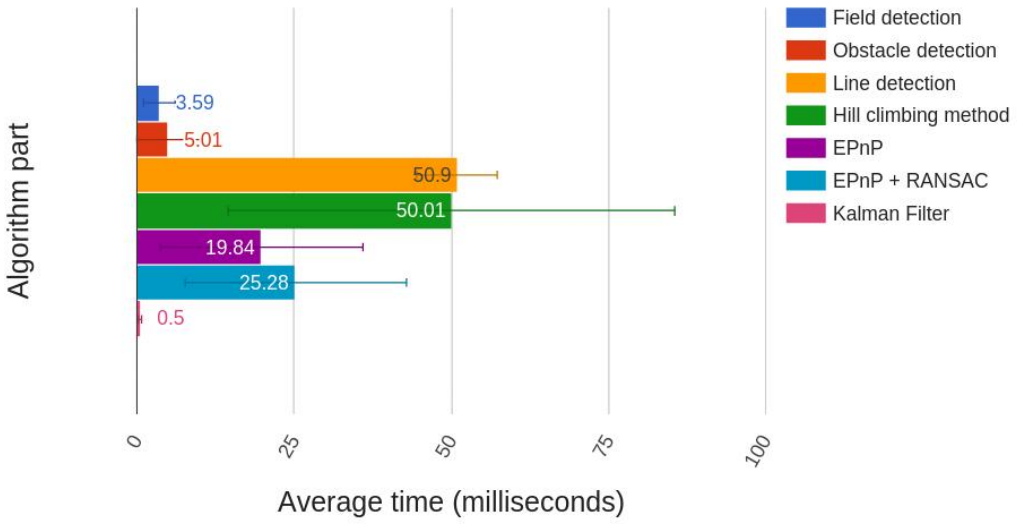


Figure 5.2: Execution time of each part of visual tracking system. The error bars in the figures indicate the standard deviations.

Features	**N**	**TPR**	**PPV**
Field boundary	200	0.94	1.0
Obstacles	30	0.95	1.0
Field lines	1038	0.52	0.90

Table 5.2: TPR and PPV results for three types of features.

5.3.3 Visual Tracking Accuracy

Regarding visual tracking accuracy, we first analyzed the results from NT group (T1-T4) and DT group (T5-T8) mainly for evaluating the three different optimization methods tested in our system. And then we analyzed the two experiments results of T9 to investigate the effects of the Kalman filter on tracking accuracy. Lastly we compared the tracking accuracy of our system to that of Farazi's system.

1. Test results of NT group and DT group

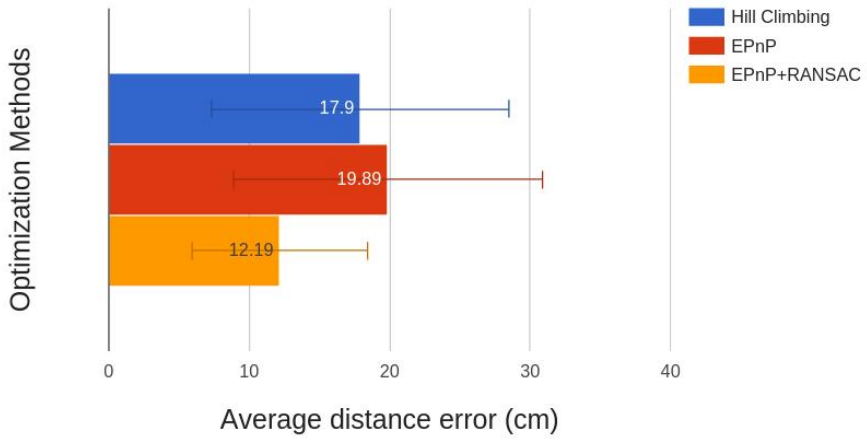
   We analyzed the evaluation result of three different optimization methods. The distance error in x and y coordinates between estimated pose and ground truth is calculated every four seconds. The average distance errors of three optimization methods of NT group and DT group are shown in Figure 5.3. As we can see from Figure 5.3, in terms of average distance errors, both hill climbing method and EPnP are larger than EPnP+RANSAC in experiments of two task groups. As already shown in computational evaluation result, the hill climbing method is the most time consuming approach among all three tested optimization methods. Moreover, the results indicate that EPnP output will be affected by the number of outliers.

   The output of EPnP+RANSAC approach shows smallest average error, which indicates that using RANSAC makes the EPnP method more robust. In NT group tests, the average distance error of EPnP+RANSAC approach is 12 cm which is better than the average error (14 cm) of the method proposed by Schulz and Behnke (2012). It is noteworthy that they tested their vision algorithms in a more color-coded environment which is relatively easier for object detection than in the environment we are using. Based on the results of three approaches, we recommend the EPnP+RANSAC as the most effective algorithm for our system. The trajectories generated by EPnP+RANSAC approach and the corresponding ground truth (in dark brown) are visualized in Figure 5.4:

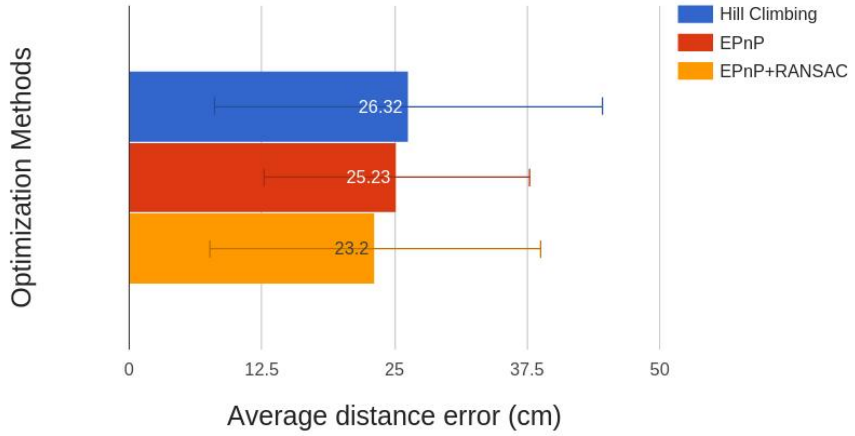
   To examine the visual tracking accuracy of our system while performing difficult tasks, we compare the accuracy results of DT group to that of NT group. As we can see from Figure 5.3 and Figure 5.4, the tracking accuracy of NT group is higher than that of DT group. Furthermore, we computed the probability of being lost $Prob_{lost}$ using the following formula:

$$Prob_{lost} = \frac{N(P_{lost}(s_t) > 0.8)}{N} \times 100\% \tag{5.2}$$

   Where $P_{lost}(s_t)$ is a parameter defined in equation (4.27) that evaluates



(a)



(b)

Figure 5.3: Tracking accuracy: the average distance error of three types of optimization approaches of (a) NT group and (b) DT group. The error bars in the figures indicate the standard deviations.

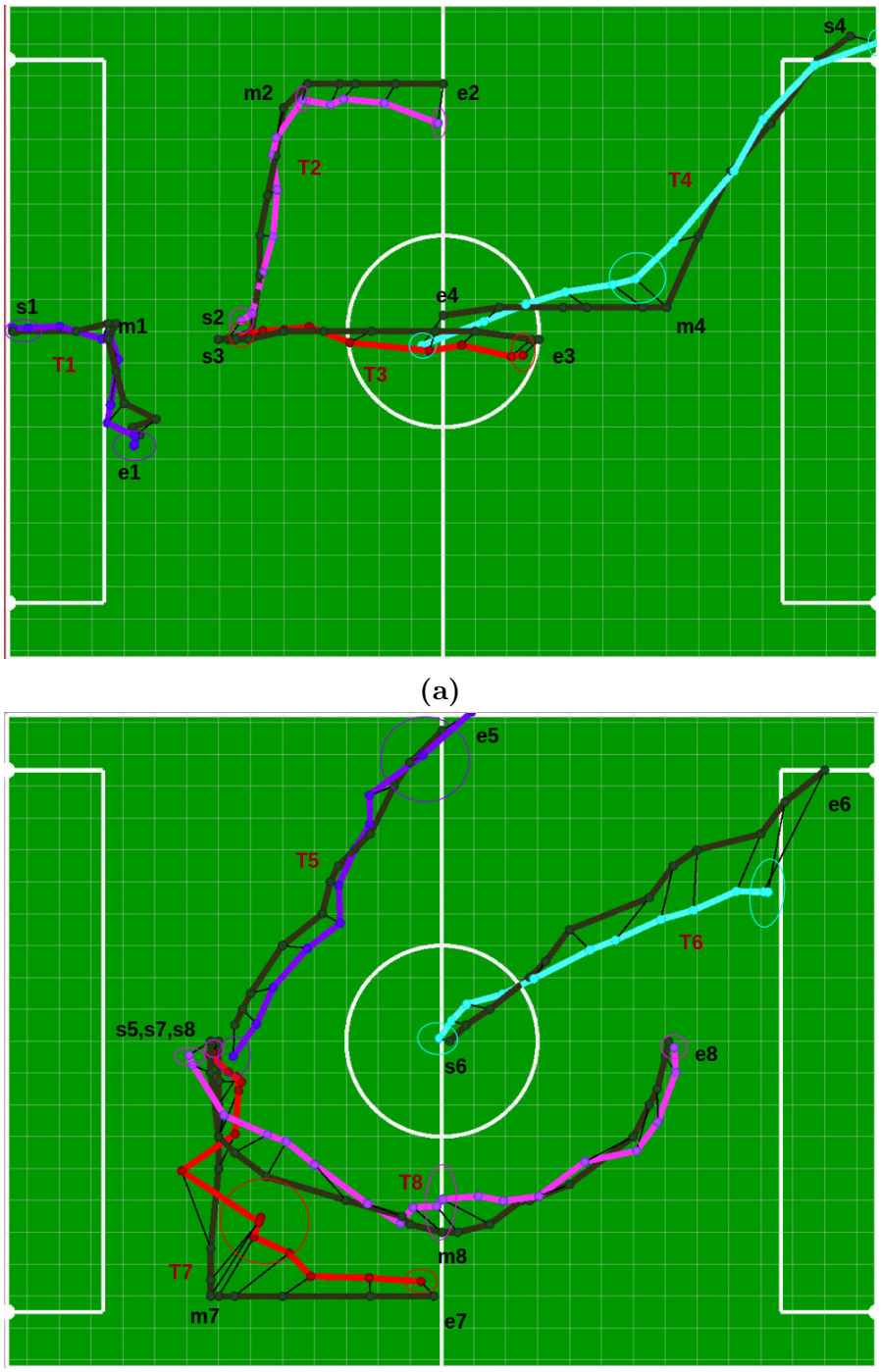


Figure 5.4: Pose tracking output using EPnP+RANSAC approach. The ground truth is illustrated by dark brown trajectories. (a) Trajectories generated from NT group tests. (b)Trajectories generated from DT group tests. The ellipses visualize the covariance of the corresponding estimated pose in x and y directions

the estimate s_t. The higher the $P_{lost}(s_t)$ is, the worse the estimate s_t is. $N(P_{lost}(s_t) > 0.8)$ is the number of bad estimates and N is the total number of estimates.

Before using multi-hypothesis approach, the $Prob_{lost}$ of NT group is 5% while DT group is 20%. Such high probability of being lost in DT group may be resulted from two aspects:

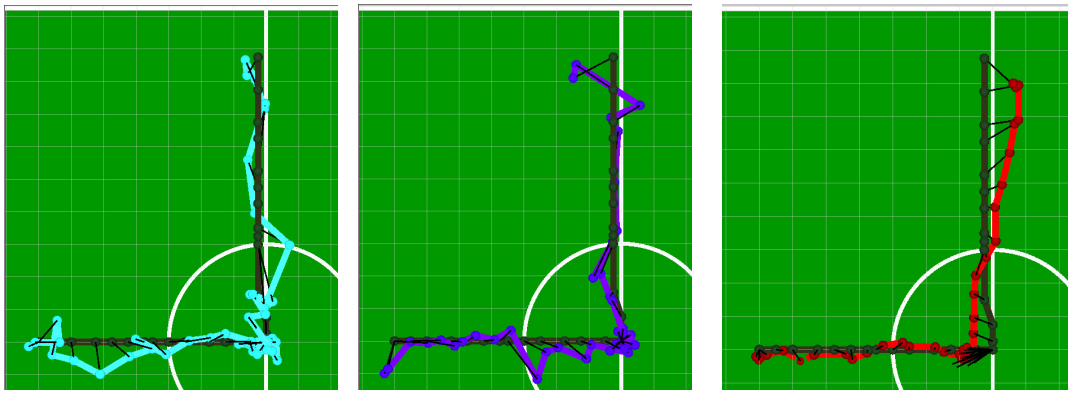
- There are many optimal solutions for 6-DoF pose optimization problem when the number of observed features is low; however, only one from these optimal solutions is the true pose. Hence, the optimization algorithm tends to converge to a false optimal solution rather than the true one. For example, when the robot moves from the starting point to the middle point along the trajectory of T7, the output (red trajectory) deviates in left and right directions from the ground truth trajectory.
- It may not be sufficient to update estimated pose only according to odometry input in case of no observations. For examples, at the end of T5, the estimated pose moves too fast; and at the end of T6 the robot moves too slowly.

However if the multi-hypothesis approach (see Section 4.6) is used then the $Prob_{lost}$ of NT group decreases to 3.5% while DT group decreases to 15%. The decrease of $Prob_{lost}$ shows that the multi-hypothesis approach helps the system to recover from a false pose estimation as long as the number of observations is enough. For example, in T7, when the camera moves from s7 to m7, the uncertainty of estimated state is increased with the decreasing of the number of observed features. However, when the robot turns around and moves to e7, the system helps the robot recover better estimated states.

2. Test results of T9

   Figure 5.5 shows the resulting trajectory from three localization approaches. As we can see from the Figure 5.5a and Figure 5.5b, using Kalman filter helps system smooth the trajectory and reduce the average distance error from 16.0 cm to 12.90 cm (see Figure 5.6). The mechanism of Kalman filte is to correct the predicted pose according to the value and covariance of new measurements. A bad measurement tends to have relatively larger covariance, thus it will have less influence on the pose correction step. Therefore the system become more robust after applying Kalman filter .

   To compared our approach (EPnP+RANSAC+KF) with Farazi's approach, we first transform our output camera pose to the trunk pose, so that the



(a) (b) (c)

Figure 5.5: Evaluation on T9 test: (a) EPnP+RANSAC for pose optimization; (b) EPnP+RANSAC for pose optimization and Kalman filter for state fusion; (c) using Farazi's approach. The unit of grid is 20 cm * 20 cm.

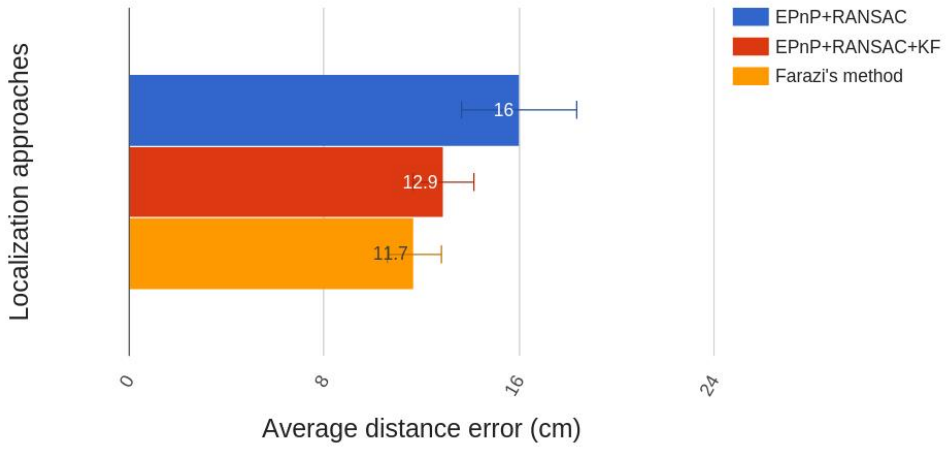


Figure 5.6: Average distance error of three types of localization approaches. The error bars in the figure indicates the standard deviations.

comparison is in the same coordinate system. The average error of our approach (12.9 cm) is slightly larger than that of Farazi's apprpach (11.8 cm). This may result from the following four reasons. First, our approach aims to optimize the 6-DoF pose, which has larger state space than 3D localization approach. Thus, the problem of finding the optimal state is more difficult and may take longer time. Further improving the optimization speed may improve our tracking accuracy. Second, currently in our visual tracking system, only the field lines painted on the soccer field are used for tracking, however, in Farazi's system, the goal posts which are important field landmarks are detected and used for localization. In fact, the errors in IMU data may also effect our result because we use transformation from IMU data to compare the two system results in a same coordinate system.

5.4 Summary

In this chapter, we start from describing the structure and design of our experiments. Then we analyze the results of the empirical experiments conducted with the visual tracking system. In general, EPnP+RANSAC+KF is regarded as the most favorable approach among all tested approaches in our system. By considering 6-DoF camera pose, this new localization approach obtains a good tracking accuracy which can be comparable to the traditional 3D pose localization approaches such as Farazi's approach. Thereby, a good overview of the characteristics of this approach could be established:

- Extracting field lines in the original full size images has higher accuracy in object detection than using the subsampled images. However, it makes line detection process time longer than other algorithm part in our vision system.
- Our object detection system detects field boundary and obstacles with very high accuracy and there is almost no false detections. The generated node graph represents the field line structure well.
- The execution time of hill climbing method is influenced by the large number of neighborhood states of the current 6D state to be exploited. This makes it the most time consuming among all the three tested optimization methods.
- Using EPnP in RANSAC schema makes the optimization more robust.
- Multi-hypothesis approach reduces the probability of being lost in the system.

- Using Kalman filter for state fusion makes the tracking approach more robust to noise measurements.
- The output of our system can be used as a source for correcting the dead reckoning messages. Better dead reckoning messages can further help to improve the visual tracking accuracy.

6 Conclusion

In this chapter, the content of the thesis will be summarized and discussed, followed by several interesting directions for future work of this work.

Improving the tracking accuracy so that the robot in Humanoid league competition has better performance in motion planning is the main motivation for this thesis. However, tracking a 6-DoF camera pose is much more challenging than tracking a 3-DoF pose, since the solution space for a pose optimization problem increases exponentially with the state dimension thus more efficient search strategies are required to explore all interesting regions within a given time budget.

In this work, we present a visual tracking system which leverages the field lines to track the 6-DoF camera pose of the soccer robot that has less dependence on IMU and dead reckoning data than traditional 3-DoF localization approaches. The system can be adapted quickly for different applications since it is built on a Linux platform with ROS, an open source software.

In visual detection part, we propose an effective approach for detecting line segments and constructing node graph. Using line kernels for detecting relative brighter pixels makes this approach less sensitive to light changes than thresholding-based approaches. By processing the local optimal values, a purified skeleton is constructed, form which a better node graph can be obtained.

In general, our system achieves high accuracy in the data registration and it is robust to outliers. This is due to applying following rules in data registration process: i) nodes in a same cluster should be assigned to a same model element. ii) the registration can only be formed when the angle difference between a node and a model element is small enough.

In the optimization step, we evaluate the performance of three optimization approaches. The evaluation results indicate that hill climbing method is not preferable in our 6-DoF pose tracking task due to long execution time, while EPnP+RANSAC is the most favorable one because it's fast and robust to outliers. Moreover, the Kalman filter makes the system more robust to noise measurement, thus the tacking accuracy is further improved.

Regarding the accuracy of our system, the evaluation result shows that our tracking systems generate a better result while performing tasks of NT group (Table 5.1) than the method proposed by Schulz and Behnke (2012). Whereas, when

our system performs difficult tasks, the tracking accuracy is decreased since the probability of being lost is increased. This is because that most of the captured images from difficult tasks only have a few observations, which makes the tracking process more challenging. However, the multi-hypothesis approach is able to reduce the probability of being lost because it enables the system to recover from a false estimated pose after losing its track to some extend.

Additionally, in order to investigate the utility of 6-DoF pose tracking in our system, we therefore compare the tracking accuracy of our system with that of Farazi's system. In terms of tracking accuracy, although the results do not show the superiority of 6-DoF pose tracking over traditional 3-DoF pose tracking, they still maintain the accuracy in a same level. In addition, since our system maintain a 6-DoF pose in 3D space, the output of 6-DoF pose tracking can be used in variety ways to improve the performance of the soccer robot. For example, it can be served as a source for correcting the IMU or dead reckoning data. Thereby, better IMU or dead reckoning can further improve the visual tracking accuracy.

However, we also noticed some drawbacks of our visual tracking system which can be summarized as follows: 1) Convolution of line filters for detecting high response pixels takes longer time than what we expect thus the overall frame rate is low. 2) The multi-hypothesis approach is not always able to recover from a false estimated pose, and its result depends on both the maximum number of hypothesis and the quality of following observations.

Further work on this system may include improving the processing speed in line detection thereby the frame rate can be increased. Moreover, the multi-hypothesis approach can be improved in several possible ways. For instance, instead of only keeping the best hypothesis in each run, it would be better to track a set of candidate best hypotheses. In addition, it is possible to test another iterative optimization method that minimizes the error by using iteratively re-weighted least squares (IRLS) (Drummond and Cipolla, 2002) which is able to achieve higher accuracy than EPnP approach as they claimed.

Appendix A

To make the cluster extraction algorithm easier to understand, we show an example in Figure 1. The input is a set of nodes i, l, m, k, j, o, n, q. The steps are as follows:

- Traversing from the first node v_i. Because $num\_connected_i > 0$ and $v_i \notin VisitedList$, adding v_i to $cluster\_1$ and $VisitedList$.
- Because $ang(v_j, v_i, v_k)$ is the max angle of v_i, and this angle is larger than β_{min_ang}, $cluster\_1$ can be expanded in two directions:
  - In v_j direction: adding v_j to $cluster\_1$ and $VisitedList$; $disconnect(v_i, v_j)$. Stop expanding in this direction because $ang(v_l, v_j, v_i) < \beta_{min\_ang}$.
  - In v_k direction: first reversing the order of elements in $cluster\_1$ then adding k to $cluster\_1$ and $VisitedList$; $disconnect(v_i, v_k)$. Because $ang(v_i, v_k, v_o) > ang(v_i, v_k, v_n)$ and $ang(v_i, v_k, v_n) > \beta_{min\_ang}$, adding v_o to $cluster\_1$ and $VisitedList$; $disconnect(v_o, v_k)$. Stop expanding in this direction because $num\_connected_o = 0$.
- Going to node v_l, because $num\_connected_l = 1$ and $v_l \notin VisitedList$, adding v_l and its connected node v_j to $cluster\_2$ and $VisitedList$; $disconnect(v_l, v_j)$. Stop expanding because $num\_connected_j = 0$.
- Going to node v_m,, similar to previous step, adding v_m, and v_i, to $cluster\_3$.
- Going to node v_k,, adding v_k, and v_n, to $cluster\_4$.
- skipping node v_j, and v_o, because both of them are in $VisitedList$ and have no connection.
- Going to node v_q,, which is not visited yet, adding v_q, to $cluster\_5$. Stop expanding because $num\_connected_q = 0$. The resulting clusters can be found on the right side of Figure 1.

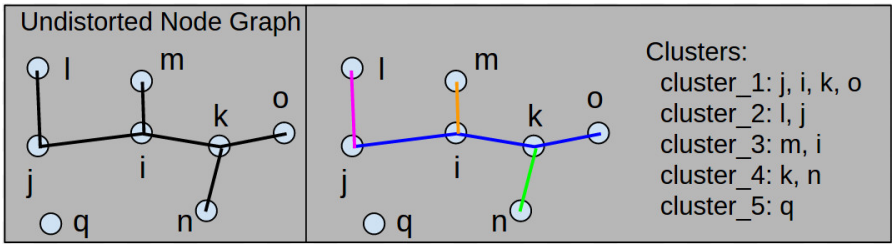


Figure 1: An example for extracting clusters from a node graph. Five clusters: four of which are in red, orange, blue, green respectively, and the fifth cluster contains a single node are extracted from the left node graph.

Appendix B

Here shows the pseudo code of fitting squares algorithm:

Algorithm 4: Fitting Squares Algorithm

Data: S_{skel_pixels}
Result: $S_{squares}$
for $P \in S_{skel\_pixels}$ **do**
  if $P \notin VisitedList$ **then**
    $current\_square = square(k = 0; c = P)$;
    while *True* **do**
      k++;
      if $Num\_of\_corners\_that\_has\_no\_neighborhood\_pixel \geq 2$ *or* $k > K_{MAX}$ **then**
        break;
      end
      if $overlap\_with\_other\_squares$ **then**
        $k = k - 1$;
        break;
      end
    end
    $c_{max} = P$;
    $current\_skel\_num = Skel\_Num[k](P_x, P_y)$;
    for P' *inside* $current\_square$ **do**
      $new\_square = square(k = 0; c = P')$;
      $new\_skel\_num = Skel\_Num[k]((P')_x, (P')_y)$;
      if new_square *overlap with other squares or* P *is not inside* new_square **then**
        continue;
      end
      if $new\_skel\_num > current\_skel\_num$ **then**
        $c_{max} = p'$; $current\_skel\_num = new\_skel\_num$;
      end
    end
    $Optimal\_square = square(k = 0; c = c_{max})$;
    $S_{squares} = S_{squares} \cup Optimal\_square$;
    for $P_o \in Optimal\_square$ **do**
      $VisitedList = VisitedList \cup P_o$;
    end
  end
end
return $S_{squares}$

Bibliography

Allgeuer, Philipp, Hafez Farazi, Michael Schreiber, and Sven Behnke (2015). "Child-sized 3D printed igus humanoid open platform". In: *Humanoid Robots (Humanoids), 2015 IEEE-RAS 15th International Conference on*. IEEE, pp. 33–40 (cit. on p. 61).

Baist, Abdul, Robert Sablatnig, and Gregor Novak (2005). "Line-based landmark recognition for self-localization of soccer robots". In: *Emerging Technologies, 2005. Proceedings of the IEEE Symposium on*. IEEE, pp. 132–137 (cit. on p. 4).

Behnke, Sven, Marcus Pfister, and Raul Rojas (1997). "Recognition of handwritten digits using structural information". In: *Neural Networks, 1997., International Conference on*. Vol. 3. IEEE, pp. 1391–1396 (cit. on p. 38).

Bengtsson, Ola and Albert-Jan Baerveldt (2003). "Robot localization based on scan-matching—estimating the covariance matrix for the IDC algorithm". In: *Robotics and Autonomous Systems* 44.1, pp. 29–40 (cit. on p. 58).

Bollobás, Béla (2013). *Modern graph theory*. Vol. 184. Springer Science & Business Media (cit. on p. 41).

Bradknox (2015). *How to Calibrate a Monocular Camera*. `http://wiki.ros.org/camera_calibration/Tutorials/MonocularCalibration`. Accessed: 2016 (cit. on p. 13).

Brown, Duane C (1966). "Decentering distortion of lenses". In: *Photometric Engineering* 32.3, pp. 444–462 (cit. on p. 14).

Bujnak, Martin, Zuzana Kukelova, and Tomas Pajdla (2008). "A general solution to the P4P problem for camera with unknown focal length". In: *Computer Vision and Pattern Recognition, 2008. CVPR 2008. IEEE Conference on*. IEEE, pp. 1–8 (cit. on pp. 8, 18).

Canny, John (1986). "A computational approach to edge detection". In: *Pattern Analysis and Machine Intelligence, IEEE Transactions on* 6, pp. 679–698 (cit. on p. 3).

Cheriyadat, Anil M, Budhendra L Bhaduri, and Richard J Radke (2008). "Detecting multiple moving objects in crowded environments with coherent motion regions". In: *Computer Vision and Pattern Recognition Workshops, 2008. CVPRW'08. IEEE Computer Society Conference on*. IEEE, pp. 1–8 (cit. on p. 6).

Chiang, Jen-Shiun, Chih-Hsien Hsia, Shih-Hung Chang, Wei-Hsuan Chang, Hung-Wei Hsu, Yi-Che Tai, Chun-Yi Li, and Meng-Hsuan Ho (2010). "An efficient object recognition and self-localization system for humanoid soccer robot". In:

SICE Annual Conference 2010, Proceedings of. IEEE, pp. 2269–2278 (cit. on pp. 3, 6).

Conrady, Alexander Eugen (1919). "Decentred lens-systems". In: *Monthly notices of the royal astronomical society* 79.5, pp. 384–390 (cit. on p. 14).

Dementhon, Daniel F and Larry S Davis (1995). "Model-based object pose in 25 lines of code". In: *International journal of computer vision* 15.1-2, pp. 123–141 (cit. on p. 8).

Derpanis, Konstantinos G (2010). "Overview of the RANSAC Algorithm". In: *York University, Toronto, Canada* (cit. on p. 20).

Drummond, Tom and Roberto Cipolla (2002). "Real-time visual tracking of complex structures". In: *Pattern Analysis and Machine Intelligence, IEEE Transactions on* 24.7, pp. 932–946 (cit. on pp. 6, 76).

Eberly, David (2002). "Rotation representations and performance issues". In: *Magic Software, Inc., Chapel Hill, NC* (cit. on p. 16).

Edmonds, Alan Robert (1996). *Angular momentum in quantum mechanics*. Princeton University Press (cit. on p. 16).

Enderle, Stefan, Marcus Ritter, Dieter Fox, Stefan Sablatnög, Gerhard Kraetzschmar, and Günther Palm (2001). "Vision-based localization in robocup environments". In: *RoboCup 2000: Robot Soccer World Cup IV*. Springer, pp. 291–296 (cit. on p. 6).

Farazi, Hafez, Philipp Allgeuer, and Sven Behnke (2015). "A Monocular Vision System for Playing Soccer in Low Color Information Environments". In: *Proceedings of 10th Workshop on Humanoid Soccer Robots, 2015*. IEEE-RAS Int (cit. on pp. 4, 33).

Farazi, Hafez, Philipp Allgeuer, Grzegorz Ficht, and Sven Behnke (2016). "NimbRo TeenSize Team Description 2016". In: pp. 1–7 (cit. on p. 64).

Ferraz, Luis, Xavier Binefa, and Francesc Moreno-Noguer (2014). "Very fast solution to the PnP problem with algebraic outlier rejection". In: *Computer Vision and Pattern Recognition (CVPR), 2014 IEEE Conference on*. IEEE, pp. 501–508 (cit. on pp. 8, 18).

Fischler, Martin A and Robert C Bolles (1981). "Random sample consensus: a paradigm for model fitting with applications to image analysis and automated cartography". In: *Communications of the ACM* 24.6, pp. 381–395 (cit. on p. 20).

Gevers, Theo and Arnold WM Smeulders (1999). "Color-based object recognition". In: *Pattern recognition* 32.3, pp. 453–464 (cit. on p. 3).

Gonzalez, Rafael C, Richard Eugene Woods, and Steven L Eddins (2004). *Digital image processing using MATLAB*. Pearson Education India (cit. on pp. 23–25).

Graham, Ronald L and F Frances Yao (1983). "Finding the convex hull of a simple polygon". In: *Journal of Algorithms* 4.4, pp. 324–331 (cit. on p. 34).

Grassia, F Sebastian (1998). "Practical parameterization of rotations using the exponential map". In: *Journal of graphics tools* 3.3, pp. 29–48 (cit. on p. 16).

Gudi, Amogh, Patrick de Kok, Georgios K Methenitis, and Nikolaas Steenbergen (2013). "Feature detection and localization for the RoboCup Soccer SPL". In: *Project report, Universiteit van Amsterdam (February 2013)* (cit. on p. 6).

Härtl, Alexander, Ubbo Visser, and Thomas Röfer (2013). "Robust and efficient object recognition for a humanoid soccer robot". In: *RoboCup 2013: Robot World Cup XVII*. Springer, pp. 396–407 (cit. on p. 65).

— (2014). "Robust and efficient object recognition for a humanoid soccer robot". In: *RoboCup 2013: Robot World Cup XVII*. Springer, pp. 396–407 (cit. on p. 3).

Hartley, Richard and Andrew Zisserman (2003). *Multiple view geometry in computer vision*. Cambridge university press (cit. on pp. 11, 13).

Hatze, Herbert (1988). "High-precision three-dimensional photogrammetric calibration and object space reconstruction using a modified DLT-approach". In: *Journal of biomechanics* 21.7, pp. 533–538 (cit. on p. 13).

Hugemann, Wolfgang (2010). "Correcting lens distortions in digital photographs". In: *Ingenieurbüro Morawski+ Hugemann: Leverkusen, Germany* (cit. on p. 14).

Hughes, John F, Andries Van Dam, James D Foley, and Steven K Feiner (2013). *Computer graphics: principles and practice*. Pearson Education (cit. on p. 16).

Julier, Simon J and Jeffrey K Uhlmann (1997). "New extension of the Kalman filter to nonlinear systems". In: *AeroSense'97*. International Society for Optics and Photonics, pp. 182–193 (cit. on p. 27).

Kendoul, Farid, Isabelle Fantoni, and Kenzo Nonami (2009). "Optic flow-based vision system for autonomous 3D localization and control of small aerial vehicles". In: *Robotics and Autonomous Systems* 57.6, pp. 591–602 (cit. on p. 5).

Khan, Waseem (2013). "Image Segmentation Techniques: A Survey". In: *Journal of Image and Graphics* 1.4, pp. 166–170 (cit. on p. 24).

Kim, Sungho, Inso Kweon, and Incheol Kim (2003). "Robust model-based 3d object recognition by combining feature matching with tracking". In: *Robotics and Automation, 2003. Proceedings. ICRA'03. IEEE International Conference on*. Vol. 2. IEEE, pp. 2123–2128 (cit. on p. 5).

Laue, Tim, Thijs Jeffry De Haas, Armin Burchardt, Colin Graf, Thomas Röfer, Alexander Härtl, and Andrik Rieskamp (2009). "Efficient and reliable sensor models for humanoid soccer robot self-localization". In: *Proceedings of the Fourth Workshop on Humanoid Soccer Robots in conjunction with the*, pp. 22–29 (cit. on pp. 1, 3).

Lepetit, Vincent and Pascal Fua (2005). *Monocular model-based 3D tracking of rigid objects*. Now Publishers Inc (cit. on pp. 11, 12).

Lepetit, Vincent, Francesc Moreno-Noguer, and Pascal Fua (2009). "Epnp: An accurate o (n) solution to the pnp problem". In: *International journal of computer vision* 81.2, pp. 155–166 (cit. on pp. 8, 18, 19).

Lowe, David G. (1991). "Fitting parameterized three-dimensional models to images". In: *IEEE Transactions on Pattern Analysis & Machine Intelligence* 5, pp. 441–450 (cit. on p. 8).

Lu, Chien-Ping, Gregory D Hager, and Eric Mjolsness (2000). "Fast and globally convergent pose estimation from video images". In: *Pattern Analysis and Machine Intelligence, IEEE Transactions on* 22.6, pp. 610–622 (cit. on p. 8).

Marr, David and Ellen Hildreth (1980). "Theory of edge detection". In: *Proceedings of the Royal Society of London B: Biological Sciences* 207.1167, pp. 187–217 (cit. on pp. 24, 25, 36).

Mehrotra, Rajiv, Kameswara Rao Namuduri, and Nagarajan Ranganathan (1992). "Gabor filter-based edge detection". In: *Pattern Recognition* 25.12, pp. 1479–1494 (cit. on p. 4).

Melen, Trond (1994). *Geometrical modelling and calibration of video cameras for underwater navigation*. Institutt for Teknisk Kybernetikk, Universitetet i Trondheim, Norges Tekniske Høgskole (cit. on p. 13).

Minakata, Hideaki, Yasuo Hayashibara, Katsuhiro Ichizawa, Takehito Horiuchi, Masahiro Fukuta, Shohei Fujita, Hiroshi Kaminaga, Kiyoshi Irie, and Hajime Sakamoto (2008). "A method of single camera robocup humanoid robot localization using cooperation with walking control". In: *Advanced Motion Control, 2008. AMC'08. 10th IEEE International Workshop on*. IEEE, pp. 50–55 (cit. on p. 6).

Mukundan, R (2002). "Quaternions: From classical mechanics to computer graphics, and beyond". In: *Proceedings of the 7th Asian Technology conference in Mathematics*, pp. 97–105 (cit. on p. 17).

Olsson, Carl, Fredrik Kahl, and Magnus Oskarsson (2009). "Branch-and-bound methods for Euclidean registration problems". In: *Pattern Analysis and Machine Intelligence, IEEE Transactions on* 31.5, pp. 783–794 (cit. on p. 8).

Papanikolopoulos, Nikolaos P, Pradeep K Khosla, and Takeo Kanade (1993). "Visual tracking of a moving target by a camera mounted on a robot: a combination of control and vision". In: *Robotics and Automation, IEEE Transactions on* 9.1, pp. 14–35 (cit. on p. 5).

Pollefeys, Marc, Reinhard Koch, and Luc Van Gool (1999). "Self-calibration and metric reconstruction inspite of varying and unknown intrinsic camera parameters". In: *International Journal of Computer Vision* 32.1, pp. 7–25 (cit. on p. 13).

Yu-qian, Zhao, Gui Wei-hua, Chen Zhen-cheng, Tang Jing-tian, and Li Ling-Yun (2006). "Medical images edge detection based on mathematical morphology". In: *Engineering in Medicine and Biology Society, 2005. IEEE-EMBS 2005. 27th Annual International Conference of the*. IEEE, pp. 6492–6495 (cit. on p. 4).

Quan, Long and Zhongdan Lan (1999). "Linear n-point camera pose determination". In: *Pattern Analysis and Machine Intelligence, IEEE Transactions on* 21.8, pp. 774–780 (cit. on p. 8).

Röfer, Thomas (2008). "Region-based segmentation with ambiguous color classes and 2-D motion compensation". In: *RoboCup 2007: Robot Soccer World Cup XI*. Springer, pp. 369–376 (cit. on p. 3).

Röfer, Thomas and Matthias Jüngel (2003). "Vision-based fast and reactive monte-carlo localization". In: *ICRA*, pp. 856–861 (cit. on p. 8).

Rowenhorst, David, AD Rollett, GS Rohrer, Mike Groeber, Michael Jackson, Peter Joachim Konijnenberg, and Marc De Graef (2015). "Consistent representations of and conversions between 3D rotations". In: *Modelling and Simulation in Materials Science and Engineering* 23.8, p. 083501 (cit. on p. 15).

Russell, Stuart, Peter Norvig, and Artificial Intelligence (1995). "A modern approach". In: *Artificial Intelligence. Prentice-Hall, Egnlewood Cliffs* 25, pp. 111–114 (cit. on p. 20).

Saito, Isaac (2016). *ROS documentation*. `http://ros.org/wiki`. Accessed: 2016 (cit. on p. 61).

Schulz, Hannes and Sven Behnke (2012). "Utilizing the structure of field lines for efficient soccer robot localization". In: *Advanced Robotics* 26.14, pp. 1603–1621 (cit. on pp. 5–7, 39, 65, 67, 75).

Slama, Chester C, Charles Theurer, Soren W Henriksen, et al. (1980). *Manual of photogrammetry*. Ed. 4. American Society of photogrammetry (cit. on p. 13).

Soccer, RoboCup (2015). "RoboCup Soccer Humanoid League Rules and Setup For the 2015 Competition in Hefei". In: p. 2 (cit. on p. 49).

Strasdat, Hauke, Maren Bennewitz, and Sven Behnke (2006). "Multi-cue localization for soccer playing humanoid robots". In: *RoboCup 2006: Robot Soccer World Cup X*. Springer, pp. 245–257 (cit. on p. 63).

— (2007). "Multi-cue localization for soccer playing humanoid robots". In: *RoboCup 2006: Robot Soccer World Cup X*. Springer, pp. 245–257 (cit. on pp. 3, 4, 7).

Suzuki, Satoshi et al. (1985). "Topological structural analysis of digitized binary images by border following". In: *Computer Vision, Graphics, and Image Processing* 30.1, pp. 32–46 (cit. on pp. 33, 34).

Thrun, Sebastian, Wolfram Burgard, and Dieter Fox (2005). *Probabilistic robotics*. MIT press (cit. on pp. 26–28).

Tsai, Roger Y (1987). "A versatile camera calibration technique for high-accuracy 3D machine vision metrology using off-the-shelf TV cameras and lenses". In: *Robotics and Automation, IEEE Journal of* 3.4, pp. 323–344 (cit. on p. 13).

Viola, Paul and Michael Jones (2001). "Rapid object detection using a boosted cascade of simple features". In: *Computer Vision and Pattern Recognition, 2001. CVPR 2001. Proceedings of the 2001 IEEE Computer Society Conference on*. Vol. 1. IEEE, pp. I–511 (cit. on p. 41).

Wan, Eric, Ronell Van Der Merwe, et al. (2000). "The unscented Kalman filter for nonlinear estimation". In: *Adaptive Systems for Signal Processing, Communications, and Control Symposium 2000. AS-SPCC. The IEEE 2000*. IEEE, pp. 153–158 (cit. on p. 27).

Warren, William H, Michael W Morris, and Michael Kalish (1988). "Perception of translational heading from optical flow." In: *Journal of Experimental Psychology: Human Perception and Performance* 14.4, p. 646 (cit. on p. 5).

Whelan, Thomas, Sonja Stüdli, John McDonald, and Richard H Middleton (2012). "Line point registration: A technique for enhancing robot localization in a soccer environment". In: *RoboCup 2011: Robot Soccer World Cup XV*. Springer, pp. 258–269 (cit. on p. 8).

Wu, Yihong and Zhanyi Hu (2006). "PnP problem revisited". In: *Journal of Mathematical Imaging and Vision* 24.1, pp. 131–141 (cit. on pp. 8, 18).

Yang, Tianwu, Changjiu Zhou, and Mohan Rajesh (2012). "A fast vision system for soccer robot". In: *Applied Bionics and Biomechanics* 9.4, pp. 399–407 (cit. on pp. 3, 6).

Yi, Seung-Joon, Steve McGill, Qin He, Larry Vadakedathu, Hak Yi, Sanghyun Cho, Dennis Hong, and Daniel D Lee (2015). "RoboCup 2014 Humanoid AdultSize League Winner". In: *RoboCup 2014: Robot World Cup XVIII*. Springer, pp. 94–105 (cit. on p. 8).

Zhang, Qilong and Robert Pless (2004). "Extrinsic calibration of a camera and laser range finder (improves camera calibration)". In: *Intelligent Robots and Systems, 2004.(IROS 2004). Proceedings. 2004 IEEE/RSJ International Conference on*. Vol. 3. IEEE, pp. 2301–2306 (cit. on p. 13).

Zhang, Zhengyou (2000). "A flexible new technique for camera calibration". In: *Pattern Analysis and Machine Intelligence, IEEE Transactions on* 22.11, pp. 1330–1334 (cit. on p. 13).