ETH
Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

BIWI Computer Vision

Solving Absolute Pose Problem in Robocup using the Field Lines

Semester's Thesis

Jan Preisig
Department of Information Technology and Electrical Engineering
Computer Vision Lab

Advisor: Dr. Ajad Chhatkuli, Jan-Nico Zäch
Supervisor: Prof. Dr. Luc van Gool

December 1, 2019

Abstract

The ”Robot Soccer World Cup”, also known as RoboCup, is a student league where fully autonomous humanoid robots compete against each other in teams of five. One of the main challenges for the robots is to localize themselves which is done through extrinsic calibration of the cameras during the game. For the exact and efficient self-localization from the camera pose, it is important to determine the offsets between the camera's orientation and the robot's head orientation. The state-of-the-art method computes the orientation angle offsets in a controlled setup by marking lines and points with human input. We provide an alternative method that requires minimal user inputs by warping the images such that the field lines suffer from minimal perspective changes. We then detect lines and filter out false lines using various heuristics in order to establish accurate correspondences to the 3D field coordinates. Our method performs the calculations in a matter of seconds compared to 10 minutes from the currently used methods and much more accurately than the feature correspondence based method.

Acknowledgements

I would first like to thank my thesis advisors, Dr. Ajad Chhatkuli and Nico Zäch from the Computer Vison Lab at ETH Zurich who gave me invaluable support and advice during this thesis. They were always open and always took the time to discuss the problems thoroughly. I am grateful to Prof. Dr. Luc van Gool to give me this opportunity to improve my skill in this new field.

I would also like to thank all the team members of the RoboCup team NomadZ and especially the teaching assistants working for the team. They tried their best to make me feel welcomed from the beginning and it was always interesting to work with them.

Contents

Chapter 1

Introduction

RoboCup with its full name ”Robot Soccer World Cup” is an international robotics competition for student teams to compete against each other with standardized fully autonomous humanoid robots. The robot used by all the teams is the Nao depicted in figure 1.1.

One of the most important challenges during the game for the robot is to localize itself on the field. This can be done through analyzing the images of the two cameras whereas for self-localization mostly the images of the upper cameras located at the forehead is used. One way to solve this self-localization problem is by calculating the extrinsic calibration parameters of the camera. The extrinsic calibration parameters represent the absolute pose composing of a matrix for orientation and a vector for translation realitve to the origin of the object coordinate system. To be able to calculate these parameters one needs corresponding pairs of image points and object points where the 3D coordinates of the objects points in the object coordinate frame are known. There are different algorithms available for different number of corresponding pairs. In this thesis an optimization method is used taken into account more than the necessary minimum number of 4 points.

1.1 Focus of this Work

The internal robot control software can calculate the transformations of the head relative to its own coordinate system located between its feet. During calibration the robot is standing straight on the floor. Which means the head orientation is also known relative to the world coordinate system located at the center of the field. It is very important to note here that the robot needs user inputs to specify its position on the field for an absolute determination of the full pose of its head inside our controlled environment. During the game the robot will self-localize. This is the baseline to compare to the estimated camera pose received through solving the absolute pose problem. An illustration of the different coordinate systems used for this thesis can be found in figure 1.2. In theory the camera’s viewpoint is exactly aligned with the viewpoint of the head. This means if we were to get the extrinsic calibration parameters of the camera we would also know the exact location of the robot on the field. In practice the camera’s viewpoint always contains small offsets compared to

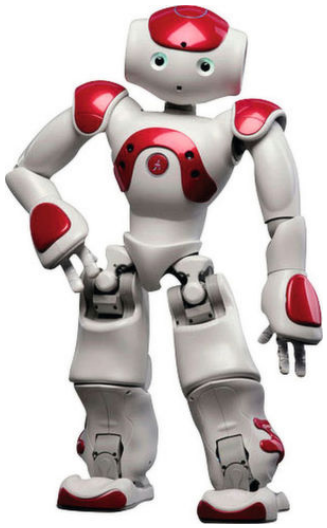


Figure 1.1: Softbank's robot NAO is used for the Standard Platform League in RoboCup

the robot's viewpoint. This is due to inaccurate positioning of the camera inside the robot's head and due to displacement of the camera caused by frequent falling of the robot during the game. In this thesis we solve the vision-based absolute pose problem in a controlled environment where the robots absolute position is known on the field. Therefore, it is possible to calculate the rotational and translational offsets between the camera viewpoint and the head viewpoint. These offsets can then be added to the robot transformation matrices and the robot can self-localize itself on the field during the game.

1.2 Organization of a RoboCup Tournament

The setup of every tournament in RoboCup is similar. The first few days(mostly two days) are for setting up the robots and getting them ready to play. The environment is always different and therefore, there is a lot of work which needs to be done before the games. When the tournaments starts there is only a limited amount of time between the games to configure the robots if something changed. The calculation of the described offsets between the camera viewpoint and the head viewpoint is absolutely crucial for the robot's self-localization and should be redone after every game due to frequent falling of the robot during the game. In the past the procedure took a huge amount of time during these setup days and because of the lengthy procedure is was often not possible to perform it for all robots between the games.

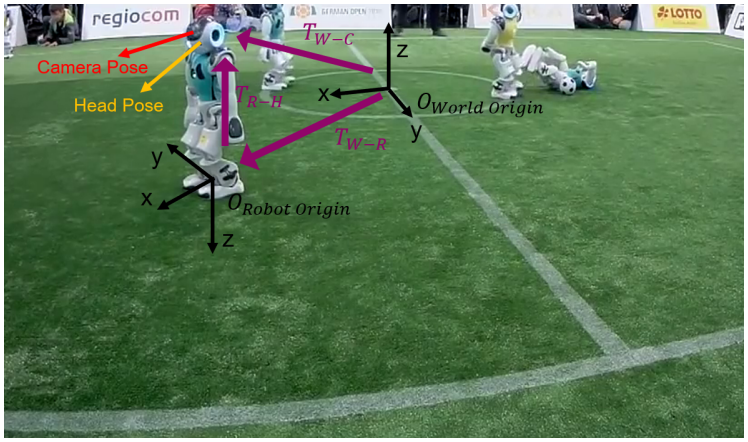


Figure 1.2: Different coordinate systems relevant for this thesis

1.3 Current Procedure

An image of the workflow is depicted in 1.3. First the robot is placed at a location on the field where the coordinates are known (either at the intersection of the center circle and middle line or at the intersection of the middle line and the side lines). These coordinates are given as input to the software. The user needs to click on the lines in the image to create samples for the algorithm. The algorithm works through a bundle adjustment optimizer which minimized the error of the samples to the closest line to iterate and optimize the parameter vector containing all the translational and rotational parameters. The lines to compare the samples to are not detected on the field but calculated through configuration files where the dimensions of the field is stored. The algorithm needs an initial camera configuration (in general it takes the current camera configuration) to calculate the error of the samples to the field lines. This means the accuracy of the result depend on the number of samples and the initial camera configuration. Practically for this method the algorithm needs around 300 samples per camera meaning 600 points on the lines of the whole field. Throughout the procedure the user has to rotate the robots head and choose samples through different viewpoints covering the whole field to get usable results. This means it includes a large part of manual work and it is very time intensive. The procedure for one robot takes at least 30 minutes and the full field is required to be mostly empty in order to be able to mark all the lines.

1.4 Thesis Organization

This thesis is organized in giving the reader a small introduction to RoboCup and the challenges of the work in the introduction as well as describing the current methods used. Furthermore, the methods section describes the newly implemented procedure and clarifies what has been done in this thesis. Finally the results and discussion section describes how

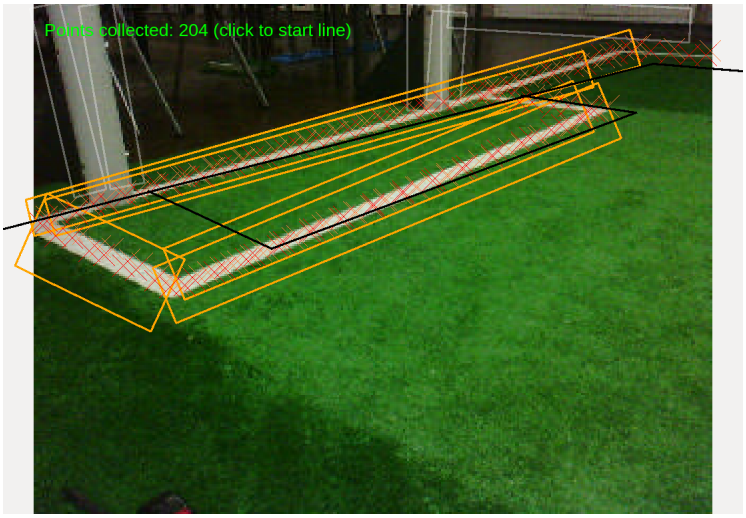


Figure 1.3: The user has to mark points on the line for the current method. For a single view more than 200 points are needed for the algorithm to converge. The lines in black are the projection of the field lines with the current values of the offset used as an initialization for the optimization algorithm.

the new algorithm fares against the current method as well as two additional different approaches to the problem.

Chapter 2

Related Work

The code for this thesis is written for the codebase of the RoboCup team NomadZ. It is written separately from the codebase during this thesis but will be included in the main codebase in a separate module. The implementation of the current method in the codebase helped to understand the input and output restrictions and gave an example on how solve the challenge.

The related work consists of different algorithms developed earlier and implemented in the opencv framework.

- *SIFT feature extraction and matching*
  The scale invariant feature transform (SIFT) developed by David Lowe in 1999 [6] is the most widely used feature detector and descriptor. It has a huge variety of application due to its invariant characteristics. It consists of different stages. The first stage is a scale-space extrema detection where the image is convoluted with Gaussian filters in different scales to create a scale space. Afterwards, keypoints are taken as the maxima/minima of the difference of gaussian (DOG). This step creates too many keypoints and some of them are unstable. So the second stage removes unstable features by checking location accuracy and edge responses. Stages three and four assign an orientation value to the features and create descriptors from the surrounding pixels intensity gradients.

- *Shi-Tomasi Good features to track*
  This is an extension to the Harris corner detection developed by J. Shi and C. Tomasi in 1994 [7]. The change they proposed concerned the scoring function used in the Harris corner detector to identify corners. They introduced a minimum function for the two λ which respresent edge direction intensities. This shows much more consistent result and the algorithm also takes a minimum distance between detected corners into account such that there is only one detected corner per image corner.

- *Canny Edge detection*
  John F. Canny developed in 1986 an algorithm to robustly detect a wide range of

edges in an image [1]. It uses a multistage algorithm by using a sobel kernel over the whole image and determining dominant edges with an upper threshold and uses hysteresis and a lower threshold to connect the dominant edges. Furthermore, a method was developed by Kerry D. Wong [8] for setting the lower and upper threshold of the canny edge detector automatically based on the mean intensity value of the image.

- *Hough Tranform*
  Developed to extract features from a binary image. The classical hough transform is used to extract lines from images but was later extended to arbitrary shapes and is today mostly used to extract lines, circles and ellipses from images. It uses a voting system in a parameter space to determine strong features. The method used for this thesis is most similar to the paper written by Duda and Hart [2].

- *Solving the Absolute Pose Problem*
  There are many different methods developed in the past to solve this problem of determining the absolute pose of the camera relative to the object coordinate system from image and object point pairs. To solve this problem one has to calculate the Rotation matrix and translation vector of the camera which projects the image points to the scene by solving a linear system of equations (See Richard Hartley's book on Multiview Geometry [3]). A.Penate-Sanchez et al. developed the universal Perspective-n-Point (PnP) method which also updates the camera matrix in the process [5].

Chapter 3

Materials and Tools

In general the whole software code base is written in c++ therefore, c++ is also used for this thesis. The main tool used for all computer vision related methods and functions is the opencv library with version 3.4.

The developed pipelines uses the Canny edge detector for edge detectin on the gray scale image and for line detection the Hough feature transform was used. For the two different approaches SIFT feature detection and matching was used and for corner detection the Shi-tomasi good features to track algorithm was used. All algorithm are available to use in opencv with all relevant parameters.

The canny edge detection and the hough line detection are crucial for main part of this thesis and therefore, a short overview over the workflow is given:

- *Canny Edge Detection*
  The canny edge detection is a multistage algorithm to robustly detect edges in a grayscale image developed by John. F. Canny [1]. There are five stages of the algorithm:
  - Apply Gaussion blur to reduce any noise and to smooth the image
  - Use a Sobel kernel to find the intensity gradient of the whole image
  - Apply non-maxima suppression
  - Apply upper threshold to find dominant edge points
  - Apply hysteresis and connect dominant edge points through edge points detected with the lower threshold

  In this thesis an addition to the canny edge detection is used developed by [8] where the upper and lower threshold is not set by the user but calculated automatically. The upper and lower thresholds are set with adding and subtracting respectively one standard deviation from the mean intensity value. This ensures robust detection of all field lines.

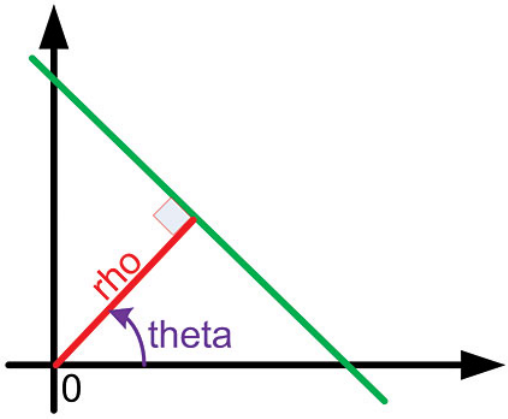


Figure 3.1: Parameter representation of a line in 2D with rho and theta. This parametrization is used in the Hough Transform.

- *Hough Line Detection*
  The hough transform is a robust method to extract shapes of an image through a voting procedure first developed by [2]. For this thesis the hough transform is used to detect lines, which means the votes are in the parameter space with rho and theta. Where rho is the distance to the origin and theta is the angle of the line. The parameters are visualized in figure 3.1. This method takes as an input normally an edge image and then each point votes for all possible lines going trough this point. If there are more votes than a certain threshold the direction is specified as a line in the image. For this thesis the voting threshold is kept quite low to make sure all line edges in the image are detected. A following filtering pipeline only keeps a single hough line corresponding to a field line edge.

Chapter 4

Pipeline Implementation

The main body of the pipeline is presented in figure 4.2. The inputs are the image from the robot's camera and an initial pose of the camera, additionally at the end the 3D coordinates of the field line corners are needed to solve the absolute pose problem. At the first run of the inner part of the pipeline the initial pose of the camera is taken from the head pose of the robot which can be calculated with user inputs and robot matrices. At the second run of the inner part the initial pose of the camera is the resulting pose from the first run. Which means this whole inner pipeline is used twice. Once to get a estimate of the camera pose out of the robot's head orientation and a second time to refine the resulting camera pose.

The first step in the inner pipeline is to create a warped image I^* with the homography transform of the image where the initial camera pose is used. The image is warped to reduce the perspective difficulties such as the scale of close and distance lines and the angle between the lines. The second step is to create a canny edge image and filter the edge point not corresponding to a field line edge. This filtered canny image is used to create the initial hough lines. The threshold for the voting of these hough lines is kept quite low such that all field lines are being detected. This leads to a large number of inaccurate lines close to the field lines but not exactly the field lines. Therefore, the main part of the whole pipeline is to remove the inaccurate hough lines to only end up with exactly one hough line representing one field line edge. There is a check at the end to see if the right number of lines is left at the end. If there are too few lines the algorithm reduces the threshold of the step filtering too many lines automatically and the pipeline is run again with the adjusted threshold. If there are too many lines detected a basic graphical user interface (GUI) is conferring with the user to check where the correct lines are.
If the filtering step was successful and the correct number of lines is left, the intersection points are calculated as well as additional points between the intersection points to improve the absolute PnP accuracy. The final step of the inner pipeline is to calculate the pose of the camera by solving the absolute PnP problem and using the 3D coordinates of all the points from a configuration file.

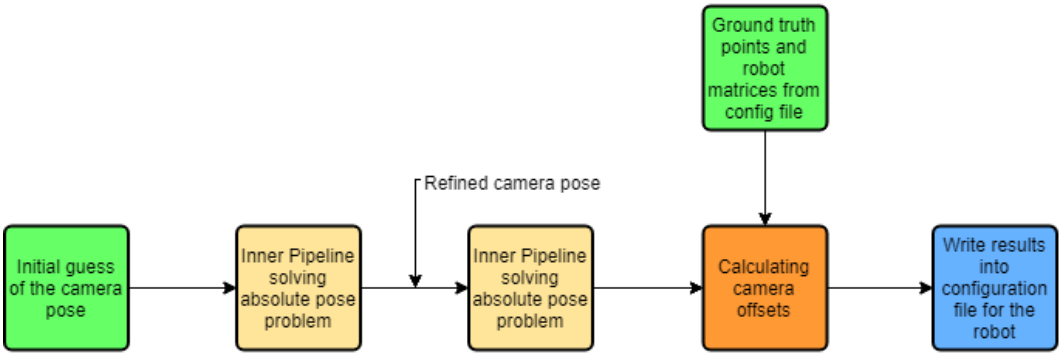


Figure 4.1: Pipeline highlevel overview

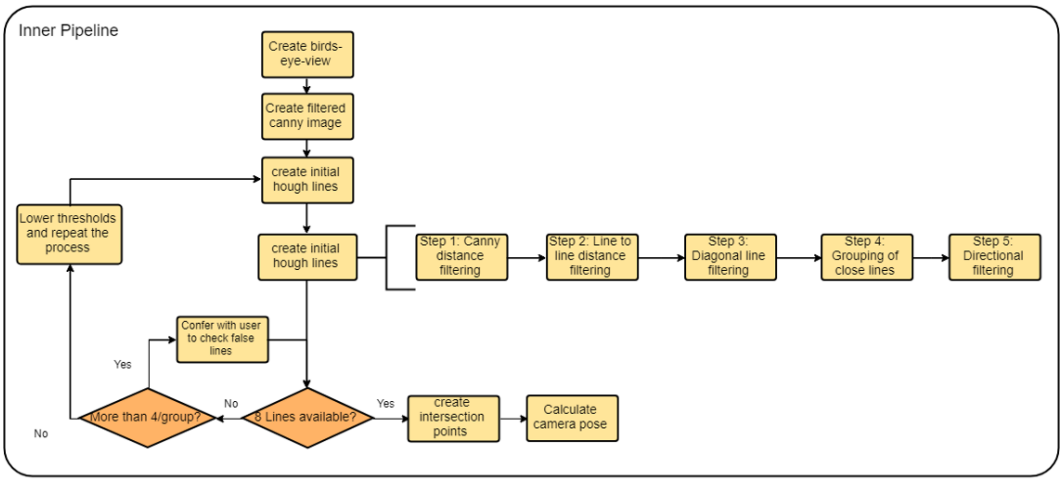


Figure 4.2: Detailed pipeline overview of the extraction of the camera absolute pose

4.1 Canny Edge Image Filtering

The general canny edge detector takes as input an upper and a lower threshold as well as a kernel size which is set to a standard value of three for this thesis. The two thresholds are set to a specific value depending on the mean intensity value of the image according to [8]. This leads to a very robust detection of all field lines, however there are also a lot of noisy small edges detected on the grass and background. This image needs to be filtered for a more stable detection of the hough lines later on. The filtering runs in two steps. The first step is a connected component analysis where edges are removed which are too short. A filter threshold is used such that the shortest field lines are not removed. The second filter step is a check through all edge pixels where the neighbourhood of each pixel is checked for multiple white pixels. If there are no white pixels around the edge it is probably part of the grass or the background and is removed. This leads to a extremely clean canny edge image mostly containing only the field lines.

4.2 Hough Lines Filtering

The filtering of the hough lines is the main piece of the pipeline. This part takes the filtered canny image, creates the initial hough lines with a low threshold to detect even the short lines and outputs the filtered lines. Ideally this step has 8 lines as output as we take only image of the penalty area into account. There is an error handling implemented for the cases where too many lines or not enough lines are detected. All of these steps are custom implemented and are specific for this application. This part of the pipeline can work as a base for improving the general line detection of the robot during the game.

4.2.1 Step 0: Initialization and Setup

The following filtering step depend on calculated distances from the lines to the edges in the canny image. Therefore, in the initialization step a voronoi diagram of the canny image is calculated. This diagram consists of a distance field where each pixel has the distance to the closest canny edge pixel as a value and a second matrix which gives each pixel a label corresponding to the closest connected component in the canny image.
The second step in the initialization phase is the truncation of the lines. The lines received from the hough detector are along the whole image. These lines are truncated with a generated canny edge mask. The canny image mask is a dilation of the canny edge image with a circular shape (see figure 4.3. An important step is to reduce the length of the lines even further on both sides for the same value as the dilation circles radius such that the lines are not longer than the field lines. This step is important because if the line is truncated longer than the actual canny edge, the distance values to the canny edge are wrong especially for the short lines. All calculations of the line to line and line to canny edge are done with the truncated lines.
The algorithm furthermore uses the line to line distance values. These values are calculated

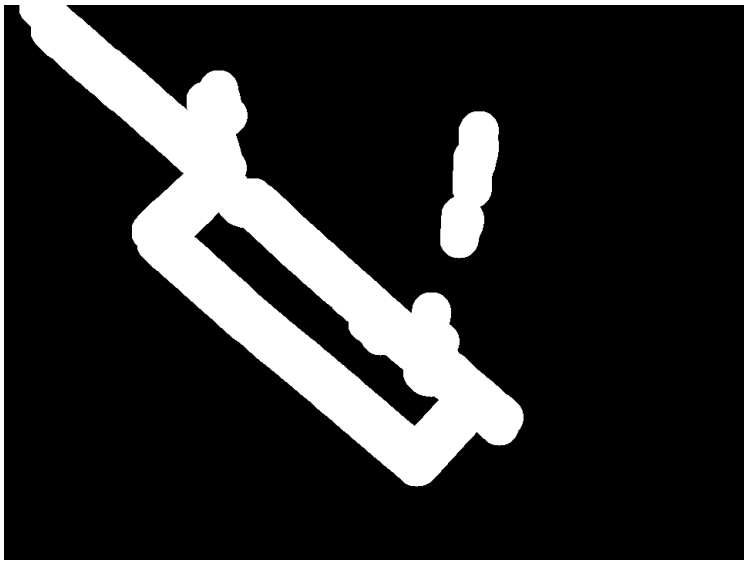


Figure 4.3: Example of a canny edge mask used to truncate the detected hough lines.

by creating a distance field for every line and then calculate the value of every line inside this distance field. However, it is computationally expensive to calculate the distance field for every line and calculate these line to line distances. That is why the line to line distances are only calculated after the first filter step was applied to reduce the number of lines and to reduce the computation time.

4.2.2 Step 1: Prefilter step by removing the lines with a high canny distance value

In this step the lines are removed which have a high canny distance value. This is a rough filter step and is used to reduce the computation time of the next steps. Only the lines are removed which are very skewed to the canny edges and therefore have a large distance to the canny edge compared to the lines parallel to the canny edge. On average this step removes 40% of the lines.

4.2.3 Step 2: Comparing line to line and line to canny distance values

The reason behind this step is that if there is a canny edge exactly represented a line then a second line is a wrong line if the distance to the canny edge is the same as the distance to the line exactly representing this canny edge.

$$d_{Line2-Line1} < threshold * d_{Line2-Canny} \quad \&\& \quad d_{Line2-Canny} > d_{Line1-Canny} \quad (4.1)$$

This means that for every line this step compares the distance to the canny edge to the distances to all the other lines. And if the canny edge distance matches the distance to

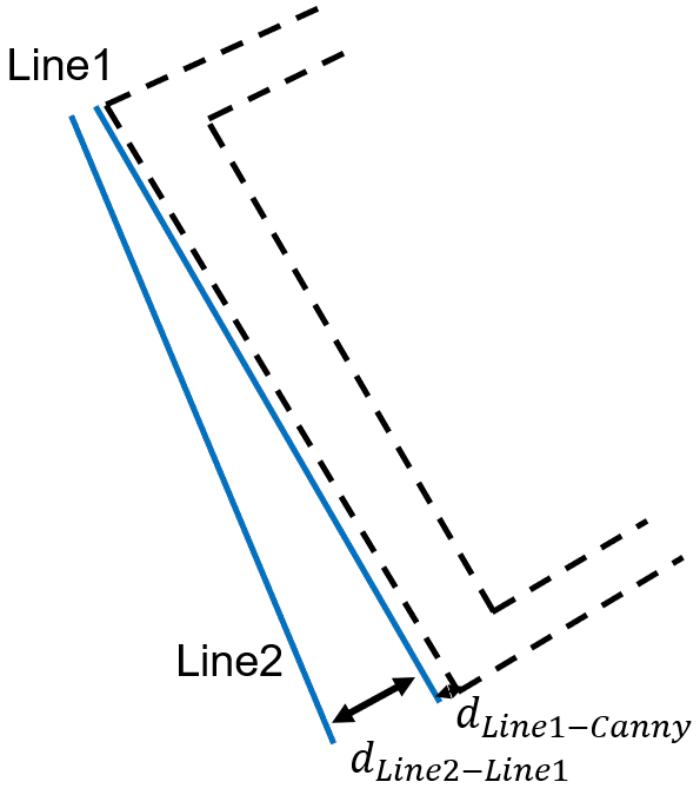
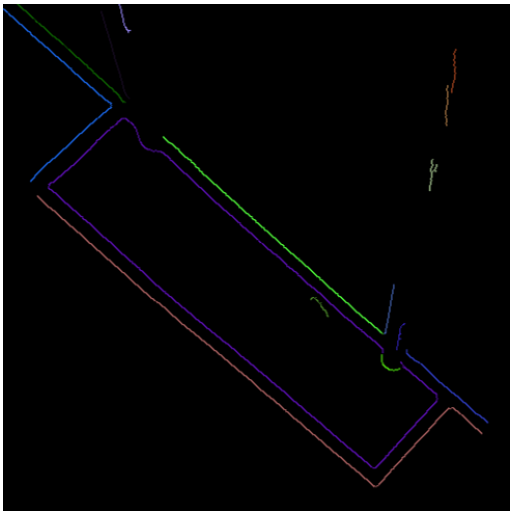


Figure 4.4: Second step in the line extraction method. Lines with a distance to a neighbour line close to the distance to the canny edge are removed.

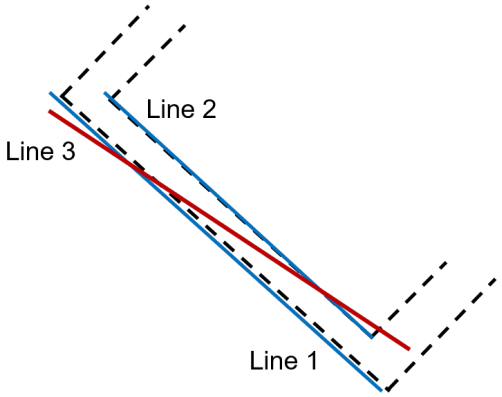
one of the other lines the line is removed. There is a second important check in this step where the line is only removed if the canny distance is larger than the canny distance of the compared line. This relationship is according to equation 4.1 and visualized in figure 4.4. Step 2 removes on average 90 percent of the available lines. This is the most important step, however, it is the computationally the most expensive at the same time.

4.2.4 Step 3: Checking for diagonal lines crossing two different canny edges

After the first two steps there are only a few wrong lines left. These are mostly the lines which do not belong to a single canny edge but kind of cross between two edges. This means they are very hard to filter out correctly. In this step this is realized with checking the closest connected components of the lines. First the algorithm searches for pairs of three lines close together. If one of such pairs is found then the connected canny edge components are listed for all three lines. The canny edges are only listed as a component of the line, if a large enough portion of the line is closest to this component. This is important such that if the line is crossing the edge perpendicularly, then this edge is not part of the connected



((a)) Connected Component analysis of a filtered Canny edge image. Every separate connected components is colored in a different color.



((b)) Pairs of 3 lines are checked in this step. An example of a interesting pair of 3 lines in visualized here. The blue lines represent the field line edges well and the third red line should be removed in this step.

Figure 4.5: Fourth step of the line extraction method. Removal of diagonal lines by checking shared connected components.

canny edge component of this respective line.
If one of those lines shares a different connected component with the other two, the line is filtered out. Say there are two parallel lines and a crossing line. The two parallel lines do not share any connected component, meaning they will not be filtered out. The crossing line shares at least one connected component with each of the parallel lines. Meaning that it will get removed in this step. A visualization is shown in figure 4.5 where an example of the connected components of a canny image is shown and a visualization of an pair of 3 lines where one line 3 should be removed by this step.

4.2.5 Step 4: Grouping lines at the same canny edge and reducing them to one line

After removing the diagonal lines there should only be lines left corresponding to a single canny edge. But it is possible that there are several lines for a single canny edge left. This means in this step the lines close together and sharing the same canny edge connected components are grouped together. For all these groups the weighted average is begin calculated with the weighs being the distance to the canny edge. This means that the closer the line is to the canny edge the more impact it has on the direction of this group.

4.2.6 Step 5: Removing lines not inside the main direction threshold

All the images are from the penalty area. This means that all the images contain the goal posts. These posts are white which means that they are also edges in the canny image. Therefore, the goal posts are also detected as lines by the hough line detector. We do need to filter out these lines, otherwise they will generate wrong intersection points at the final step. The idea behind this step is that all the field lines are nearly parallel or perpendicular to each other because we created a warped image with the homography transform in the beginning of the algorithm. Which means that there is a principal direction containing one group of lines and a direction 90 degrees to the principal direction containing the second group of the correct lines. The algorithm then calculates this principal direction and groups the lines into parallel and perpendicular groups. All lines are checked separately. For each line outliers and inliers are calculated separated by a angle threshold. From these inliers the principal directions can be calculated by grouping the common inliers of all the lines together. With a final check through all the lines with this calculated principal direction, the outliers outside a certain threshold can be removed, this would be for example lines from the goal posts.

4.2.7 Step 6: Checking the final lines

In the optimal case the thresholds of all the steps were set correctly and the algorithm delivers exactly 8 lines with two groups of 4 parallel lines. If this is the case then all the intersection points can be calculated and the algorithm can continue. If there was a mistake in filtering the lines, the algorithm performs some error handling the following way.

If one of the two parallel group of lines contain less than 4 lines the whole pipeline is repeated. First the threshold of the hough line detection is being reduced given that maybe the line was not detected from the start. If the line was detected, but removed in a later step, the general hough line threshold is reduced and the threshold corresponding to the step which removed the line is reduced automatically to repeat the process and hopefully not remove the line in a second run. Most of the times the algorithm results in the correct number of lines at least after the second run.

If one of the two parallel groups contains more than 4 lines the user is taken into account. A basic GUI is conferring with the user for each line if it is correct or not.

With these two error handling steps the resulting 8 lines can be found robustly with either automatic reduction of threshold or with conferring with the user.
It is possible that even after reducing the threshold several times or conferring with the user that not all of the 8 lines are being detected. Then this image is not suitable for this procedure and another one has to be taken into account. However, with the images taken as test image this happened only to a few images.

4.3 Absolute Pose and Offset Calculation

At this step it is guaranteed that there are exactly 8 lines left. To achieve the final offset calculations 3 more steps have to be made.

4.3.1 Find all intersection Points

With all the correct lines present and a suitable heuristics, the lines can be matched to the corresponding field lines. In this thesis this is done via the rho values of the line. The lines are already grouped into two groups of parallel lines. These groups can be sorted along the value of the rho parameter to match them to the field lines. After this is done it is possible to intersect the correct lines with each other to create the intersection points in same order as the ground truth points are given as input in a configuration file. To increase the accuracy of the PnP method afterwards, we generate more points homogeneously along the lines between the intersection points.

4.3.2 Calculate the Camera Pose

With the help of the opencv solvePnPRansac method, the camera position and orientation relative to the world origin can be calculated. Different methods have been checked covering P3P, iterative and more advance methods like a direct least square method proposed in this paper proposed by Hesch et al [4] or the UPnP where the camera matrix is iteratively updated at the same time proposed by Kneip [5]. The solvePnPRansac method gives as output the orientation in angle axis representation and the translation vector.

4.4 High Level Pipeline

The high level flow chard is illustrated in figure 4.1. It starts with calculating an initial estimate of the camera pose through the pose of the head. For this initial estimation it is assumed that the camera is perfectly aligned with the head. After performing the whole absolute pose pipeline described above once to get a refined camera pose the refined intersection points are calculated again by repeating the whole inner pipeline with the refined camera pose. This is necessary because the first initial estimate of the camera is not exact. With the refinement step the intersection points can be determined more exactly and therefore the absolute pose problem will have a higher accuracy.

With this refined camera pose estimation the offsets can be calculated.

4.4.1 Calculate offsets comparing Camera Pose and Head Pose

The final step of the pipeline is calculating the offsets needed to update the robots configuration file. The translation offset can be directly calculated through the translation vector of the head pose and the camera pose relative to the world coordinate system. However, the orientation offset is more difficult to calculate and one has to be very careful about the

rotation convention. The angle offsets have to be given in yaw, pitch and roll angles in the robots coordinate frame visualized in figure 1.2. There is a difference in coordinate system definitions in computer vision and robotics. The camera coordinate system follows the computer vision definition and has the z axis along the view points or the cameras principal axis, the x-axis is to the right of the camera and the y-axis is downwards. The robot has the robotics or aerial definitions with x-axis forward, y-axis to the right and z-axis facing downwards. Due to the fact that the offsets in orientation have to given in the Euler angles ZYX following the aerial conventions the corresponding Euler angles for the camera orientation were calculated following the same convention but a different succession of rotation to define the same angles. Which means that the rotation sequence corresponding to the ZYX Euler angle sequence is the YXZ sequence in to get from the world coordinate system to the camera coordinate system. With these corresponding angles the camera orientation can be expressed as a rotation matrix in the world coordinate system with the same axis definitions as the robot's head which means that the rotation matrix can be calculated representing the orientation difference between the robot's head and the camera and therefore, the Euler angles can be derived from the resulting rotation matrix.

4.5 Final Procedure to perform the Offset Calculations

First the robot has to be placed at a position where the whole penalty area is visible in the image. This step is important for calculating the offsets afterwards, that is why a position plate has been constructed to be able to position the robot correctly with the right angle. The robot needs to stand upright and the user should make sure that it is looking straight and should note the current pitch of the head as an input later for the pipeline. The images taken with the robot has to be given as input to the pipeline, with the configuration file determining the robot ground truth pose, robot matrices, the camera intrinsic matrix and the scene coordinates of the field line intersection points as additional inputs. The pipelines runs fully automatically to give the translation and orientation offsets as outputs in a configuration file.

Chapter 5

Experimental Results

The evaluation of the proposed method is split into three different sections. First is the evaluation of the inner pipeline representing the evaluation of the line detection and intersection point determination. Second is the result of solving the absolute pose problem in three different ways containing the proposed new pipeline, a general SIFT matching alternative and corner matching. Last is the comparison between the proposed new method and the current method used.

5.1 Inner Pipeline Evaluation

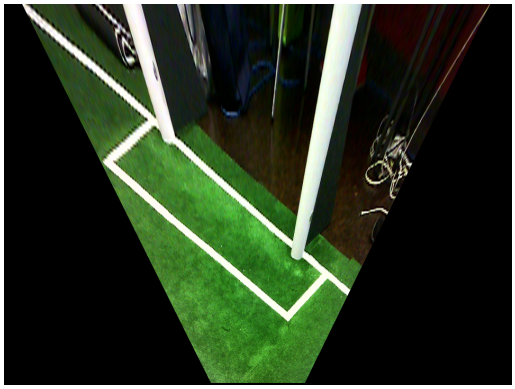
The inner pipeline is the most important part of the method. Due to the constrained nature of the problem we are only interested if the inner pipeline results in eight lines.This evaluation is, therefore, done qualitatively.
The threshold for the first step where lines with a large distance to the canny edge are removed is 2.4 pixels in average. This means that lines with an average distance larger than 2.4 pixels are removed from the image. The threshold for the second step is 1.1. Which means that if the line to line distance is within 10% of the line to canny edge distance then the line is removed. For the last step where the lines are grouped into parallel and perpendicular groups the threshold is at plus/minus 10 degrees.

5.1.1 Results of the different filter steps

The first step in the inner pipeline is to transform the image into a image transformed with the homography. The result of this transformation can be seen in figure 5.1(b). It can be seen that the short lines further away from the camera are much better separated and the field lines intersect each other almost perpendicular.

The following figures represent examples for the different filter steps in the inner pipeline for one example image. Figure 5.2(b) shows the result of the canny image filtering. The result of this step is a very clear canny image without the noisy edges of the initial image. Figure 5.3(a) shows an example of the initial hough line detection. It can be seen that there

((a)) Original image taken with the robot from a perspective of the penalty area



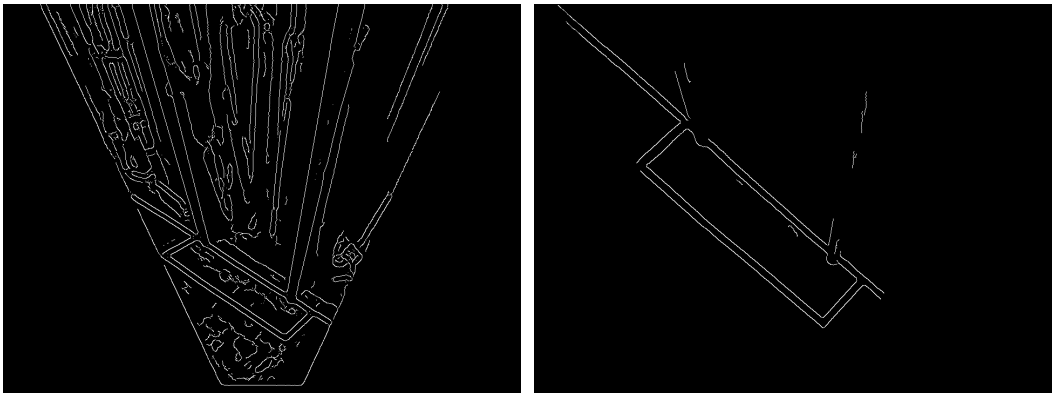
((b)) Resulting birds-eye-view of the image used for the filter process.

Figure 5.1: First step of the inner pipeline

are many obviously wrong lines mostly at the long lines. The threshold is set low such that the short lines are detected and the figures shows that at the short lines there aren't that many detected lines. If the threshold is set higher then these lines won't get detected at all. The first step is a rough filtering of the lines. It can be seen in figure 5.3(b) that this step is removing a lot of lines even though it is a simple check. It is used to reduce the number of lines before the more computationally expensive steps and would otherwise not be necessary. A more complex step is the second step. Figure 5.3(c) illustrates the result after removing the lines where a close line is at the same distance than the canny edge. It is clear that this step removes the most lines. The results after the first two filtering steps are in general very promising. These two steps filter out the majority of the lines. The next three steps take care of single lines where special heuristics has to be taken into account. The result of the third step is shown in figure 5.3(d) where it can be seen very nicely how the two diagonal lines are removed at both long field lines. The fourth step where close lines belonging to the same canny edge are grouped together often doesn't remove any lines because the lines are already removed at the second step. The results of the last step can again be seen nicely in figure 5.3(e). The line originating from the goal post was successfully removed.

Lastly figure 5.3(f) shows the result of the inner pipeline including the create intersection points and additional points in between. It can be seen that the lines are not perfectly accurate. But it can also be seen that the field lines are not exactly straight. That is why the lines seem to fit the field lines and then start to drift away. This is caused by the fact that the actual field lines are bent leading to curved lines in the image. Which makes it impossible for the lines to fit the field lines perfectly.

Figure 5.4 shows the results of the inner pipeline from both sides of the penalty area. It can be seen that only the required correct lines are left and all the required lines are being detected. The resulting hough lines fit the field lines accurately. The result on the truncation



((a)) Initial canny edge image from the canny edge detector implemented in opencv with automated lower and upper threshold calculations

((b)) Resulting canny edge image after filtering for connected component lengths and filtering for close field lines

Figure 5.2

of the lines can also be seen here in the lines in blue as the result of the masking with the dilated canny edge image.

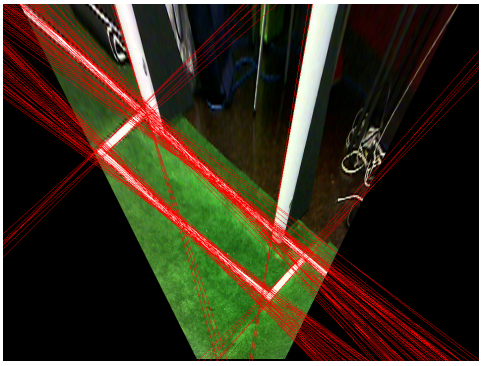
5.2 Different Approaches to Solve the Absolute Pose Problem

5.2.1 Direct SIFT feature extraction and matching

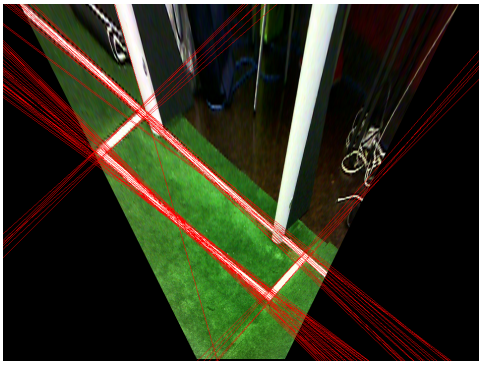
In general the absolute pose problem is solved via feature matching, thus, tested as reference in the project. The reference consists of SIFT feature extraction and matching of a test image with a template image which contains annotated 3D coordinates of the features. Figure 5.5 shows the result of the matching. The SIFT matching was tested on the normal image and on the warped image. It can be seen that the warped image contains better matches but clearly still not enough for a robust matching. It can be clearly seen that even with a nearest neighbour optimization approach of the matches that the direct SIFT feature matching will not give satisfying results and was not further taken into account.

5.2.2 Matching of the Extracted and filtered Corners

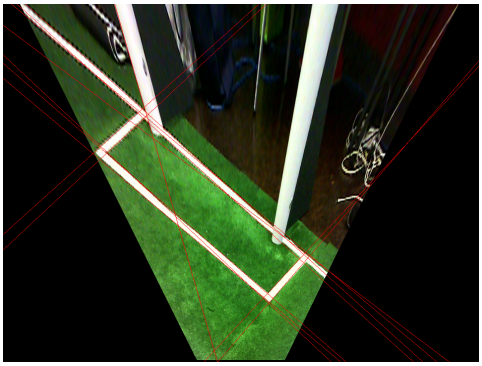
The second approach was to use the SIFT matching on certain points on the image, namely on extracted corners. The corners were detected with the shi-tomasi good features to track algorithm based on their paper [7]. This lead to lot of corners along the edges and all over the image. With extracting the field lines the corners were filtered in order to only match the corners of the field lines. One can see in figure 5.6 that some of the matches are correct but not enough to generate a robust matching pipeline and solve the absolute pose problem. It can also be seen that for both approaches the reprojection error was so high that the



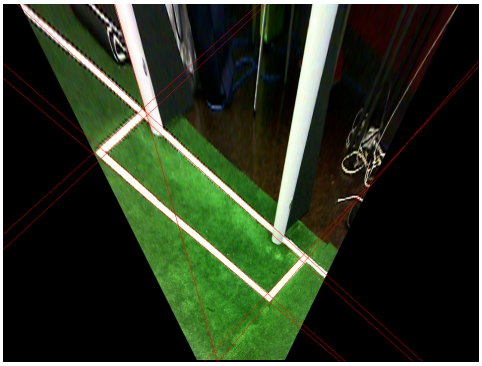
((a)) Initial hough lines with a low threshold to detect the short lines. The hough lines are focused on the field lines but there are a lot of obviously incorrect lines.



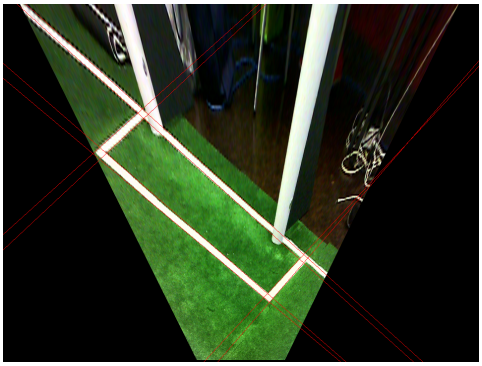
((b)) Thresholding the lines with the distance to the canny edges removes a lot of the obviously wrong lines and reduces computation time for the next steps.



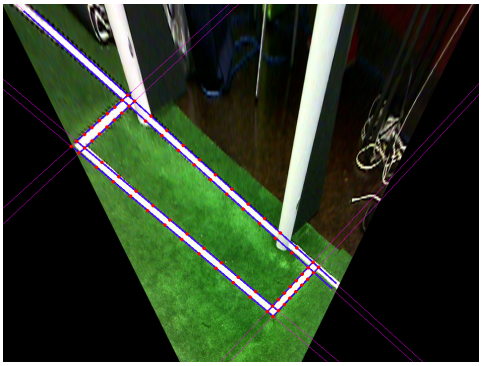
((c)) This is a more complex filter step where the line to line distance is compared to line to canny distance. If these two are equal for a line pair, then the line is removed.



((d)) The third filtering step in the pipeline takes care of single lines. It compares pairs of three lines to check if one of the lines intersects the other two.

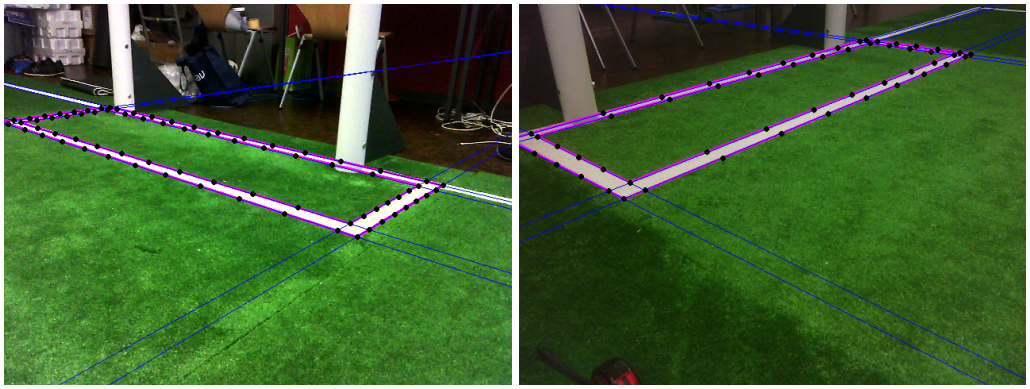


((e)) The last step removes lines resulting from the goal posts by grouping the lines into a principal direction and a direction perpendicular to this one. The lines from the goal post normally varies a lot form this direction and can be filtered out.



((f)) Shows the result of the inner pipeline with the created intersection points and additional points in between. The thicker lines represented the truncated lines with the dilated canny edge image.

Figure 5.3



((a)) Image take from the right of the penalty area ((b)) Image take from the left of the penalty area

Figure 5.4: Two examples results of the inner pipeline from both sides of the penalty area. Blue lines represent the hough lines, purple lines represent the truncated lines and the black points are the intersection points with additional points in between.

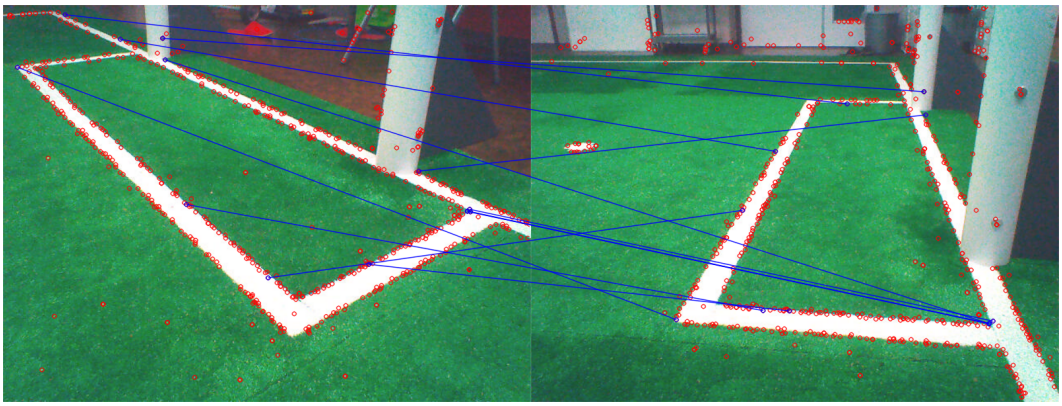
reprojected points weren't even on the image.

5.2.3 The Implemented Final Pipeline

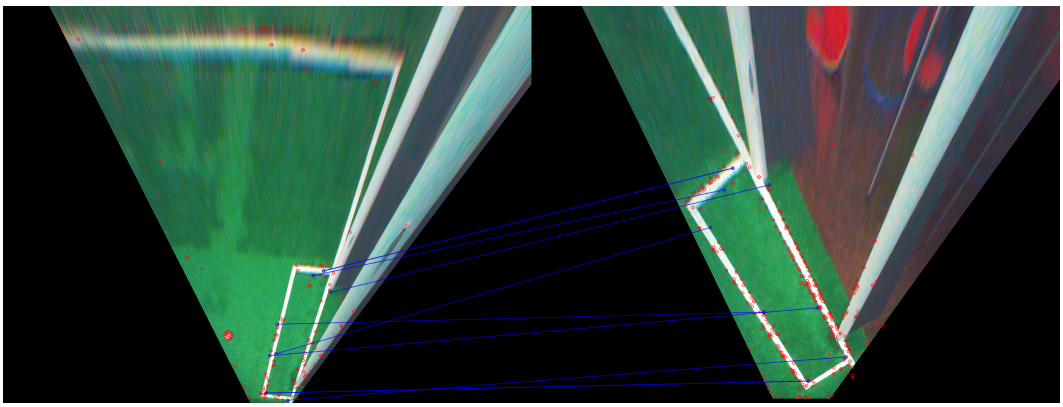
The result of solving the absolute pose problem with the proposed line filtering pipeline and intersection point calculation achieved average reprojection error of three to four pixels. The result are visible in figure 5.7. The black dots represent the created intersection points and the blue dots represent the reprojected points. The reprojection has an error of 3-4 pixels on different images from both sides of the penalty area.

5.3 Offset Calculations of the Current and the Proposed Method

In general it is difficult to measure or calculate the ground truth of the two methods, therefore, the results of the two methods will be shown separately and compared to each other. To be able to compare the two methods the setup should be as identical as possible. Meaning that for this evaluation the current method has been tested on the same view like the proposed method. Figure 5.8 shows the result of the current method and our method evaluated in the same way with the same restrictions. One can see that the projected field lines differ from the actual field lines. However the order of the error is similar in both views. For the current method to converge to a solution needs at least 200 to converge and 600 for satisfying results. In this test setup the number of points had to be at least 100 which meant that it took around 5min to input the points and run the procedure. For the new method only the image need to be taken with the robot and given as input to the software. The pipeline finished in around 20 seconds and without manual input and with similar values as a result. Which leads to significant reduction of effort.



((a)) SIFT feature matching with nearest neighbour filtering.



((b)) SIFT feature matching with nearest neighbour filtering on the warped images

Figure 5.5: First approach to the problem with a SIFT feature extraction on a test image on the left to match with a template image on the right.

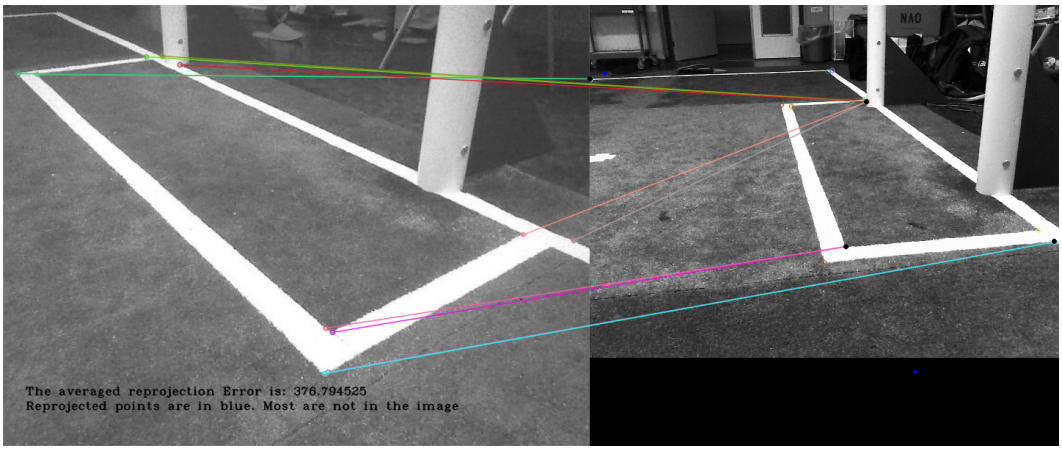


Figure 5.6: SIFT matching of the extracted and filtered corners also did not lead to satisfying results.

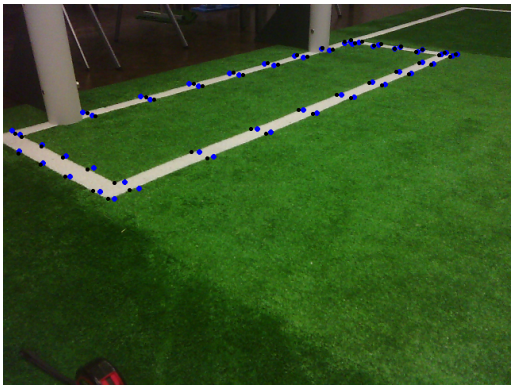
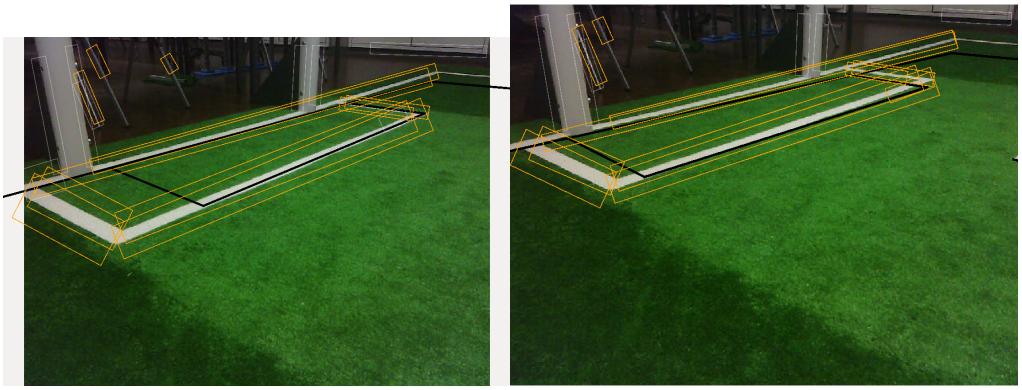


Figure 5.7: Final result of the finished pipeline with the detected points in black and the reprojected points in blue.



((a)) Result of the proposed method for the calculation of the offset.

((b)) Result of the current method on a single view with more than 140 points as input, otherwise the algorithm cannot converge.

Figure 5.8: Qualitative result of the two methods. The lines in black represent the projected field lines. It can be seen that on both images the projected lines do not match exactly the field lines. But on a qualitative basis the two methods result in similar results.

Table 5.1 shows the results of the two methods on the same image and the same setup.

Parameter Offset	Current Method	Proposed Method
Translation[mm]		
x	21	51
y	-133	59
Orientation[deg]		
Yaw	0.9	-3.7
Pitch	7.6	4.6
Roll	5.2	-0.5

Table 5.1: Result of offset calculation of the current method and the proposed method.

Chapter 6

Discussion

The proposed pipeline performs well on the controlled environment. The results from the same setup are similar or even better than the current method with a lot less sample points, almost no manual work and it can be performed a lot faster with less effort. However, if one performs the current method exactly and with enough time the results are more accurate. This means that the pipeline presents a nice base to improve the procedure further to take also take more than one position into account and to further automate the whole procedure to remove the manual input in the beginning.

There are also possibilities to combine the two methods. One can choose the time consuming but exact manual method, if there is enough time available e.g during the setup days of the tournament. At a later stage during the game days when there is not a lot of time available the team can switch to the faster version or even adapt the new procedure to perform the whole pipeline during the game.

The developed inner pipeline for the line detection can be extended into other methods for example the line detection during the game and therefore improve the self-localization of the robot. Especially if the robot is standing close to the penalty area the implemented line detection works perfectly, also detecting the short lines even if the current implemented line detection in the robot misses them.

Furthermore, the proposed new procedure works robustly inside the constraints that the camera needs to see majority of the penalty area to make sure it can detect all the necessary lines. The placement of the robots in the position inside these constraints is not optimal and difficult to perform exactly because the robot cannot be placed exactly at a corner. The provided placement plates decrease the margin for error but it is still prone to inaccurate placement. Additionally, the official field during the tournaments is a different one than the team has in the lab available. The officially field is larger and has different sprayed lines which are much more difficult to detect.

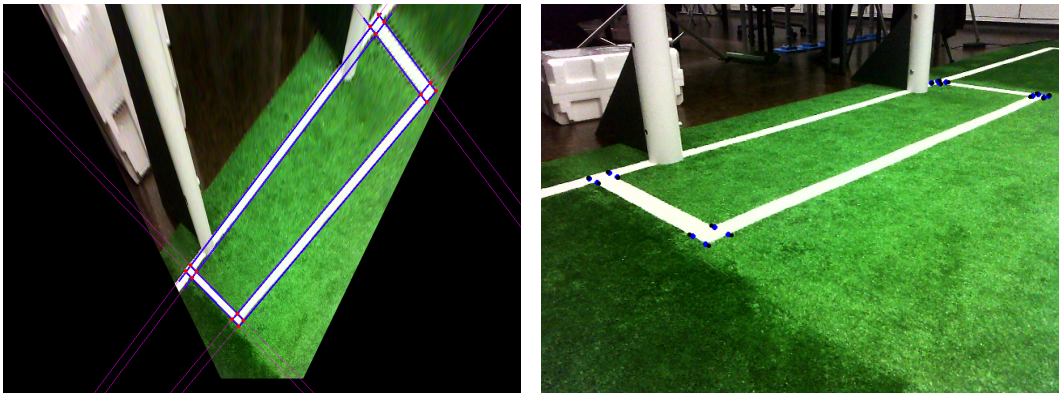
Chapter 7

Conclusion

As a conclusion I can say that the algorithm works in giving the user the required offset values for the robot to use during the game. The procedure should be made more robust to be able to work in a larger variety of places on the field to give the user more freedom to place the robot exactly. The developed algorithms and procedures have a lot of potential in improving the game-play in different fields. The usefulness of the proposed new pipeline has to be further tested and improved to be able to use in on the official field and maybe even online during the game.

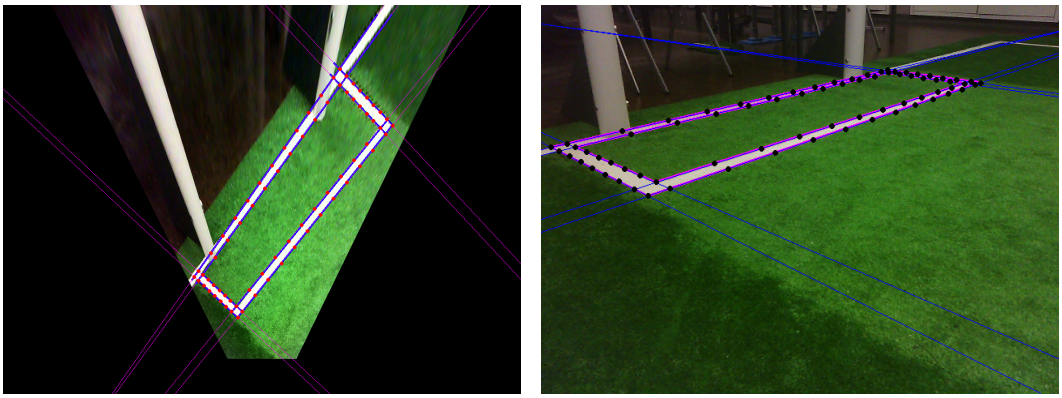
Appendix A

The First Appendix



((a)) ((b))

Figure A.1: Result of the inner pipeline on the left with the created intersection points and the result of reprojecting the ground truth point on the image after solving the absolute pose problem and calculating the camera pose.



((a)) ((b))

Figure A.2: Two additional examples of the result of the inner pipeline. Inside the constraints the field lines and intersection points are detected robustly and accurately

Bibliography

[1] John Canny. A computational approach to edge detection. In *Readings in computer vision*, pages 184–203. Elsevier, 1987.

[2] Richard O Duda and Peter E Hart. Use of the hough transformation to detect lines and curves in pictures. Technical report, SRI INTERNATIONAL MENLO PARK CA ARTIFICIAL INTELLIGENCE CENTER, 1971.

[3] Richard Hartley and Andrew Zisserman. New York, NY, USA, 2 edition.

[4] Joel A Hesch and Stergios I Roumeliotis. A direct least-squares (dls) method for pnp. In *2011 International Conference on Computer Vision*, pages 383–390. IEEE, 2011.

[5] Laurent Kneip, Hongdong Li, and Yongduek Seo. Upnp: An optimal o (n) solution to the absolute pose problem with universal applicability. In *European Conference on Computer Vision*, pages 127–142. Springer, 2014.

[6] David G Lowe et al. Object recognition from local scale-invariant features. In *iccv*, volume 99, pages 1150–1157, 1999.

[7] Jianbo Shi and Carlo Tomasi. Good features to track. Technical report, Cornell University, 1993.

[8] Kerry Wong. Canny edge detection auto thresholding, 2009.