ISSN 0280-5316
ISRN LUTFD2/TFRT--5830--SE

Vision Based Tracker for Dart Catching Robot

Magnus Linderoth

Department of Automatic Control
Lund University
December 2008

Lund University
Department of Automatic Control
Box 118
SE-221 00 Lund Sweden

Document name
MASTER THESIS

Date of issue
August 2008

Document Number
ISRN LUTFD2/TFRT--5830--SE

Author(s)
Magnus Linderoth

Supervisor
Anders Robertsson Automatic Control, Lund
Kalle Åström Mathematics, Lund
Rolf Johansson Automatic Control, Lund (Examiner)

Sponsoring organization

Title and subtitle
Vision Based Tracker for Dart Catching Robot (Datorseendebaserad banföljning för pilfångande robot)

Abstract
The objective of this thesis has been to develop the foundation for a robot system that catches darts. A dart board is to be mounted on a robot. When a dart is thrown at the board, it is detected by cameras, and an algorithm predicts where the dart will hit the board. The goal is then to move the board in such a way that the dart hits at the desired coordinate, typically the bull's eye. This report describes the different components developed to realize this system, including image acquisition, camera calibration, image analysis and modeling and estimation of the dart trajectory.

Keywords
Dart catching, computer vision, camera calibration, state estimation.

Classification system and/or index terms (if any)

Supplementary bibliographical information

ISSN and key title
0280-5316

ISBN

Language
English

Number of pages
44

Recipient's notes

Security classification

Acknowledgments

The work resulting in this thesis was carried out at the Department of Automatic Control, LTH, Lund University. I would like to thank my supervisors, Anders Robertsson at the Department of Automatic Control and Kalle Åström at the Department of Mathematics for their guidance throughout the project. I would also like to thank Anders Blomdell and Leif Andersson at the Department of Automatic Control and Sven Gestegård Robertz and Mathias Haage at the Department of Computer Science for helping me with practical issues during the implementation. Last, but not least, I want to thank all the people at the department for making my time there pleasant.

Contents

1. Introduction

This chapter describes the problem to be solved and analyzes its feasibility. Some guidelines for the solution are also listed.

1.1 Background

In practical robot applications it is often desirable to have systems that respond quickly to input from the robot environment. The most common input is position, velocity etc. of the robot or an object to be tracked.

Using cameras as sensors has several advantages. They can make touch free measurements of very generic types, e.g. position, shape or color. The limitation for what can be done often lies in the algorithms that analyze the images. Some image properties can be very difficult to extract and the execution time is often significant. Hence the problem gets even harder when the camera is used as a sensor in a real-time system.

1.2 Problem formulation

The objective of this thesis has been to evaluate the use of high speed computer vision in a motion control application. The specific problem chosen was to develop the foundation for a dart catching robot. A sketch describing the problem can be seen in Figure 1.1. A dart board is mounted on a robot and when a dart is thrown toward the board, the robot should move the board so the dart always hits the bull's eye. The detection of the dart is performed with cameras that provide

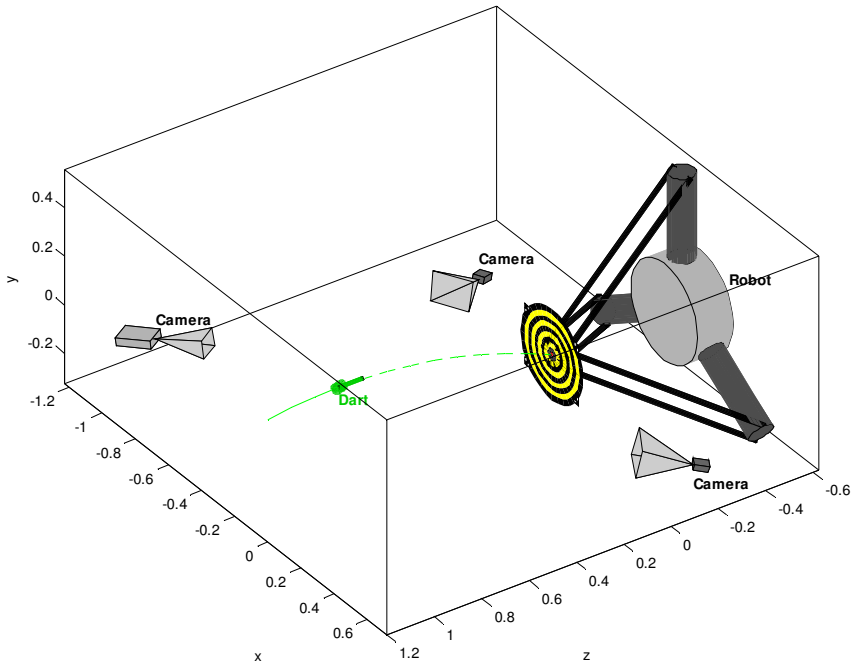


Figure 1.1 Sketch of setup.

data to the algorithms that estimate the position and future trajectory of the dart. This information should in turn be used to move the dart board to the correct position.

1.3 Feasibility Analysis

A typical dart throw is performed 2-3 m away from the dart board and the dart gets a forward velocity of 5-8 m/s. Consequently the duration of the throw is approximately 0.25-0.6 s. In this time the trajectory of the dart should be estimated with sufficient precision and the dart board should be moved to the predicted hit point.

Assume that the robot has the maximum acceleration capacity a. The fastest way to move the dart board between two points, having zero speed at both end points, is to first give full acceleration towards the end point, and half way give full acceleration in the opposite direction. If this movement has the duration T, the distance traveled is $d = 2 \cdot \frac{a(T/2)^2}{2} = a(T/2)^2$. Assuming $a = 6g$ (the capacity of the Flexpicker described in Section 2.3) and that we want to be able to move the dart board $d = 0.2$ m, this movement can hence be done in $T = 2\sqrt{d/a} \approx 0.12\,\text{s}$. Considering that the duration of the dart throw was previously estimated to 0.25-0.6 s there seems to be a good margin. This margin will be reduced by delays from data transmission and computations. Another limiting factor is that good estimates of the hit point will not be available until the dart has traversed a part of its trajectory.

1.4 Guidelines

It is desirable that the solution conforms to some guidelines to optimize the utilization of the hardware and make maintenance easy.

- **Modularity.** The software should be split into modules that can be developed independently or be replaced without having to make changes in the rest of the system. Examples of such modules can be image acquisition, image analysis, triangulation, dart trajectory estimation and dart board trajectory generation.
- **Easy exchange of hardware components.** It should be easy to e.g. switch to other cameras or another kind of robot.
- **Restrictive movement of the dart board.** Every time a measurement of the dart position is acquired during its flight, there is an update of the estimate of where the dart is predicted to hit the dart board. If the estimate moves around a lot and there is a long time left to the impact, it is desirable that the robot does not shake the dart board too much. Still the robot should try to move the board to the approximate position where the dart is expected to hit, so the board will have a shorter way to go when the estimate improves. When the dart is close to the board, the robot should use its full capacity to move the dart to the estimated hit point if it is necessary to get there before the impact.

2. Hardware and Platform

This chapter describes the different hardware components and tools used to develop and realize the system.

2.1 Setup

An overview of the setup can be seen in Figure 2.1. The dart board is mounted on a robot. On each side of the dart board there is a camera (denoted FWA and FWB) pointing toward the dart thrower. These are used to get images of the dart's flight and provide data to the prediction of where the dart will hit the dart board. Another camera (denoted NW) is pointed toward the dart board and used to evaluate where the dart actually hit the board.

2.2 Cameras

The cameras used to photograph the flight of the darts (FWA and FWB) have to be able to capture images at a high frame rate, and fine control of the image acquisition is desirable. The cameras used in this project were two Basler A602fc

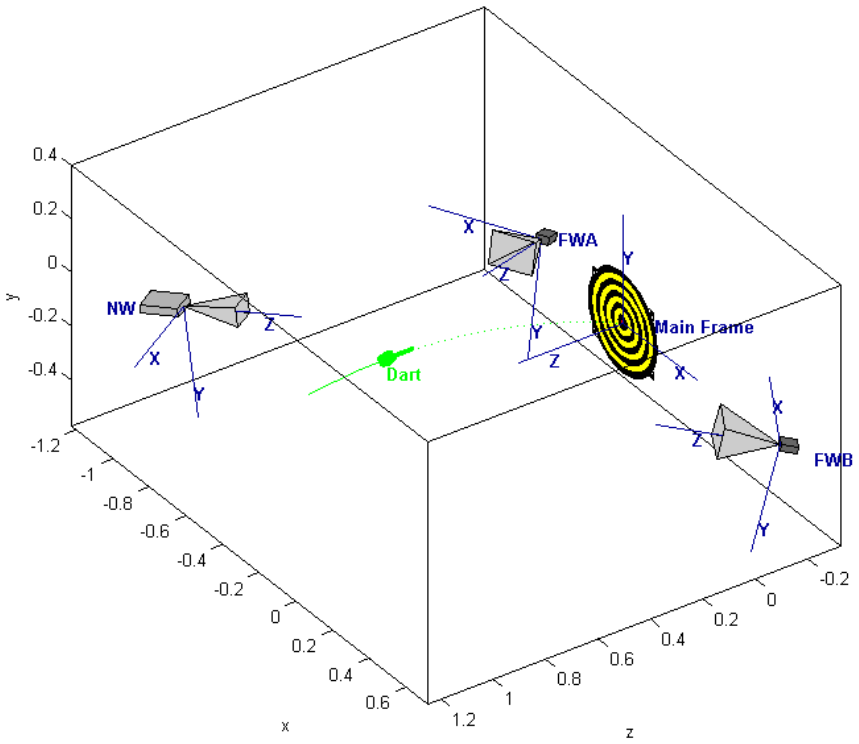


Figure 2.1 Overview of the setup. The main frame is used to measure the position of the dart. A possible position and trajectory of the dart are marked in green. The cameras are denoted NW, FWA and FWB. Their coordinate axes are marked in blue. The cones in front of the cameras illustrate their fields of view.

(a) Basler A602fc (b) Axis 211A

Figure 2.2 The cameras used in the project.

(Figure 2.2 (a)), which can supply color images at resolutions up to 656×490 pixels with a maximum frame rate of 100 fps (frames per second) at full resolution. The communication with the cameras is done with the IEEE1394 serial bus commercially known as Firewire or iLink. The cameras conform to the IIDC [10] standard for digital cameras

The camera pointed toward the dartboard does not have to fulfill the same real-time requirements, since it is used mainly to get still images of the dart on the dart board to measure where it hit. For this an Axis 211A (Figure 2.2 (b)), connected through Ethernet, was used. It can supply 640×480 pixel color images at rates up to 30 fps, but with latencies up to 1 s in the current setup.

2.3 Robot

The implementation has not been specialized for any special kind of robot. The robot that is planned to be used is a Flexpicker IRB 340 from ABB. Its truss-like structure makes it stiff, though light-weight parts are used. This in turn allows it to effect large accelerations, something that is essential for the dart catcher application. The robot can be seen in Figure 2.3.

Figure 2.3 Flexpicker IRB 340

2.4 Tools

Matlab

Most algorithms have been developed and evaluated using Matlab, which facilitates fast development through a large set of functions for calculations and visualization. Most algorithms have then been implemented in C or Java for better real-time performance. Some scripts that are run offline (e.g. for camera calibration) are implemented only in Matlab.

Programming Languages

The first steps in the data flow, including image acquisition, image analysis, state estimation and prediction, are implemented in C, running on a Linux platform. Given an estimation of where the dart will hit the board, the Java part generates a trajectory for the robot and actuates it. The reason for splitting the implementation into two languages is that the Firewire cameras are interfaced through a C driver and the interface to the robot is written in Java.

The C and Java parts of the program communicate through a socket, making it easy to either run the two parts on the same computer or on different computers.

A schematic overview of the components and how they communicate can be seen in Figure 2.4.

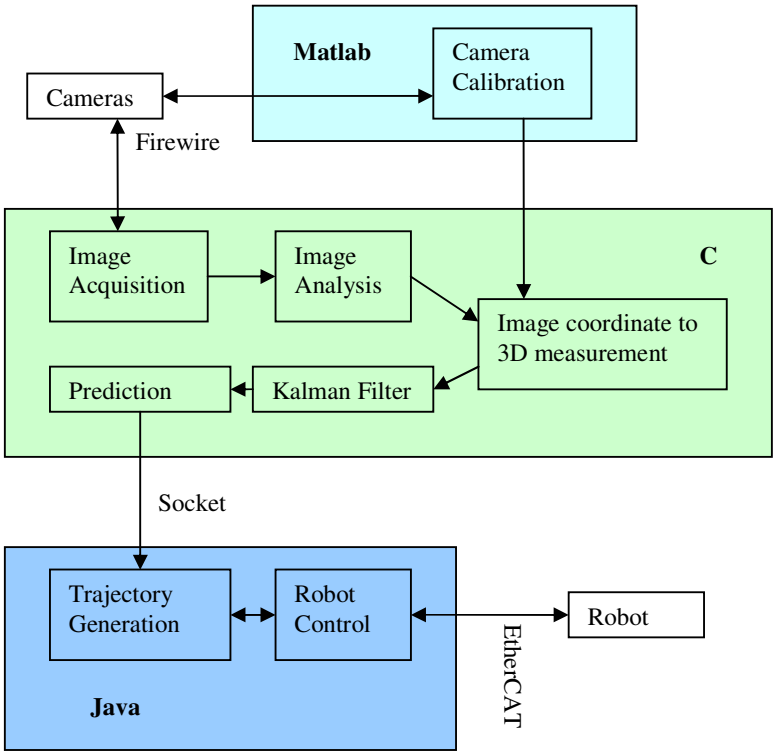


Figure 2.4 A schematic overview of the components in the system and how they communicate.

libdc1394

The communication with the Firewire cameras is done with libdc1394 [2]. It provides a high level API in C for cameras conforming to the IIDC standard [10].

OpenCV (Open Source Computer Vision)

OpenCV [3] is a C/C++ library mainly aimed at real-time computer vision. In this project it has mostly been used for matrix operations and to display images on the computer screen in real time.

Robot interface

The robot is interfaced through a Java control system developed at the Department of Computer Science at Lund University [15]. The Java controller takes position references with velocity feed forward and communicates with the robot through an EtherCAT interface [16].

Camera Calibration

For camera calibration the Camera Calibration Toolbox for Matlab [4] by Jean-Yves Bouguet was used. It uses series of images of a checkerboard pattern to extract both intrinsic camera parameters and the relative position of the cameras in a stereo pair. It has a relatively intuitive GUI with convenient functions for detecting and eliminating errors. The localization of grid corners is done semi-automatically; the user has to click the approximate position of four corners in each image. Then the remaining corners in the grid are identified and their positions are calculated at sub pixel level.

As alternative calibration tools Multi-Camera Self-Calibration by Tomas Svoboda [5] and EasyCal Camera Calibration Toolbox from University of Pennsylvania [6] were considered. They both use a point light source being moved around in front of the camera in a dark room. This strategy facilitates easy image analysis at the cost of requiring the possibility to make the room dark. Another important limitation of the toolbox by Svoboda, related to the fact that only one point is detected in each image, is that it requires at least three cameras with common field of view. EasyCal on the other hand requires that the intrinsic parameters of at least two of the cameras are determined with a checkerboard and the toolbox by Bouguet.

Taking all these facts into account, the toolbox by Bouguet was considered to be the most appropriate. A desirable future extension would be to make the detection of the checkerboards completely automatic, maybe using [13] or [14].

3. Methods

3.1 Positioning of Cameras

Many different ways of positioning the cameras are possible. The chosen positions of the high speed cameras are on each side of the dart board, pointing toward the thrower at an angle. The positioning and the stereo coverage are illustrated in Figure 3.1.

This positioning allows the cameras to see the dart at a big distance if the thrower is straight in front of the board. The dart catcher application has a higher requirement on the accuracy of the estimated dart trajectory when the dart is close to the dart board than when it is far away, since there then is little time left to the impact and the dart board trajectory generator needs to know exactly where to send the dart board. This requirement is fulfilled by the chosen camera positioning, since the dart is close to the cameras when it is close to the board. The fact that the cameras observe the dart from very different angles (optimally 90° difference) when the dart is close to the board, further increases the accuracy of the position estimate after the triangulation.

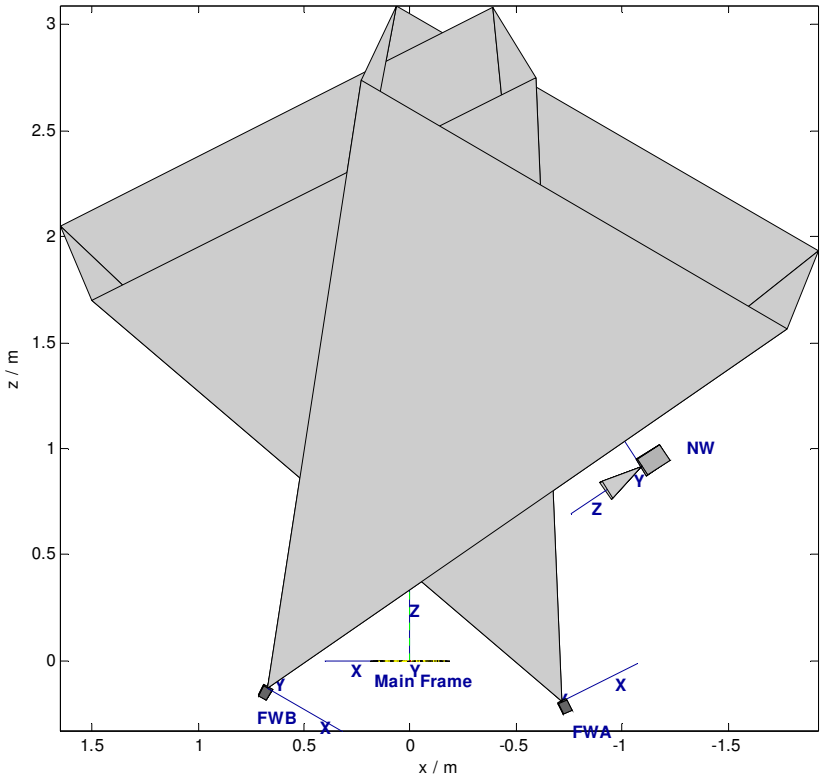


Figure 3.1 Top view of the setup illustrating the stereo coverage. The cones show the field of view for the respective cameras.

The last possible point on the trajectory, where the dart can be seen by both cameras simultaneously, is straight in front of the board, approximately 0.45 m away from the board.

Another consideration in the placement of the cameras is that it should be easy to calibrate the setup. This is described in more detail in the following section.

3.2 Camera Calibration

To be able to calculate the position in space of a dart captured in an image, it is necessary to know the camera parameters. The *extrinsic parameters* tell the position and orientation of the camera with respect to some coordinate system. The *intrinsic parameters* tell how a point in space, measured in a coordinate system fixed to the camera, is projected onto the image.

Homogeneous Coordinates

In computer vision it is often convenient to represent points with homogeneous coordinates, since it allows projections and coordinate transformations to be performed as matrix multiplications. For the purposes of this report they can be described by the following paragraphs.

To describe a point $(x \quad y \quad z)^T$ in 3D space with homogeneous coordinates, a "1" is augmented:

$$(x \quad y \quad z)^T \rightarrow (x \quad y \quad z \quad 1)^T \tag{3.1}$$

To find out which point in space is represented by an arbitrary homogeneous coordinate vector, it is rescaled so that the last element is equal to one:

$$\begin{pmatrix} a \\ b \\ c \\ d \end{pmatrix} = d \begin{pmatrix} a/d \\ b/d \\ c/d \\ 1 \end{pmatrix} \rightarrow \begin{pmatrix} x \\ y \\ z \end{pmatrix} = \begin{pmatrix} a/d \\ b/d \\ c/d \end{pmatrix} \tag{3.2}$$

Camera Parameter Representation

A point $\mathbf{X} = (X \quad Y \quad Z \quad 1)^T$ in some reference frame can be transformed to a coordinate $\mathbf{X_c} = (X_c \quad Y_c \quad Z_c \quad 1)$ in the camera reference frame with some translation, $\mathbf{t}_{3\times1}$, and some rotation, $R_{3\times3}$.

$$\mathbf{X_c} = \begin{pmatrix} R_{3\times3} & \mathbf{t}_{3\times1} \\ \mathbf{0}_{1\times3} & 1 \end{pmatrix} \mathbf{X} \tag{3.3}$$

Let $\mathbf{x_n}$ be the normalized projection of $\mathbf{X_c}$ defined by

$$\mathbf{x_n} = \begin{pmatrix} x_n \\ y_n \\ 1 \end{pmatrix} = \begin{pmatrix} X_c / Z_c \\ Y_c / Z_c \\ 1 \end{pmatrix} \sim \begin{pmatrix} X_c \\ Y_c \\ Z_c \end{pmatrix} \tag{3.4}$$

It corresponds to the coordinate where a line between $\mathbf{X_c}$ and the camera's focal point intersects the plane $Z_c = 1$ (cf. Figure 3.2). This projection can be obtained by removing the last row from (3.3):

$$\mathbf{x_n} = \begin{pmatrix} R_{3\times3} & \mathbf{t_{3\times1}} \end{pmatrix} \mathbf{X} = P\mathbf{X} \tag{3.5}$$

The matrix *P* is the output from the extrinsic parameter estimation process.

Since the camera lenses are not ideal, they exhibit nonlinear behavior. Consequently the normalized coordinates are distorted: $\mathbf{x_n} \xrightarrow{distortion} \mathbf{x_d}$. Three coefficients ($k_1$, k_2 and k_5) represent the radial distortion (along lines through the principal point). Two coefficients (k_3, and k_4) represent the tangential distortion (along circles with their centers at the principal point). For more details, consult [4].

Finally $\mathbf{x_d}$ is converted to pixel coordinates, $\mathbf{x_p}$, through

$$\mathbf{x_p} = K\mathbf{x_d} = \begin{pmatrix} f_x & \alpha f_x & c_x \\ 0 & f_y & c_y \\ 0 & 0 & 1 \end{pmatrix} \mathbf{x_d} \tag{3.6}$$

where f_x and f_y are the focal lengths, (c_x, c_y) is the principal point and α is the skew coefficient.

For more details on camera parameterization, read [4, 9].

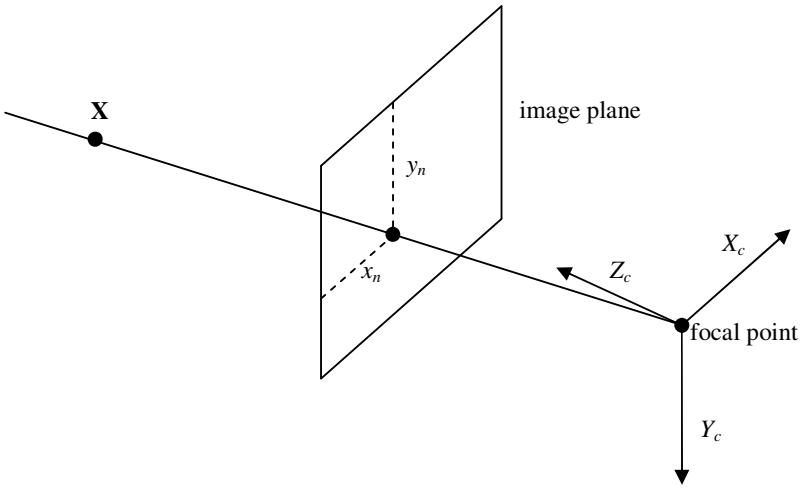


Figure 3.2 Illustration of projection of 3D-points onto an image plane. The projection of **X**, some point in space, is determined by finding the point where the line between **X** and the focal point intersects the image plane.

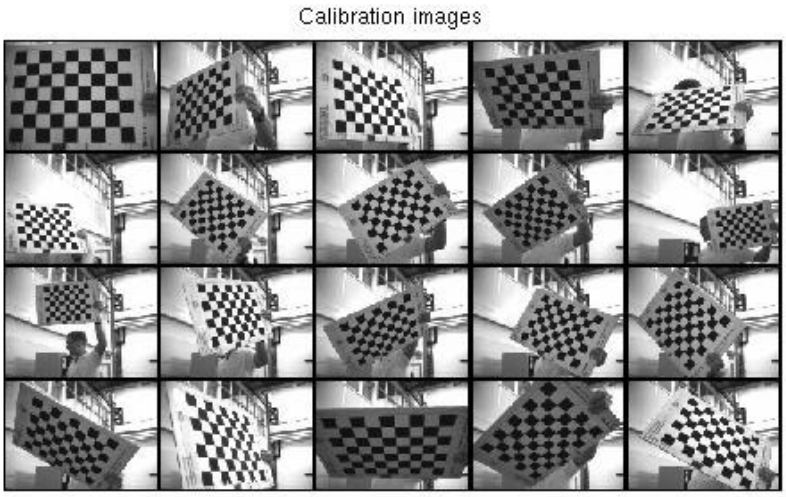


Figure 3.3 Set of images of a checkerboard used to calibrate the intrinsic parameters of a camera.

Calibration of Intrinsic Parameters

In order to measure the intrinsic parameters, a checkerboard pattern with known square sizes is used. A set of images of the checkerboard in different position is captured. An example can be seen in Figure 3.3. The camera calibration toolbox by Bouguet has a function to extract the intrinsic parameters from these images, including lens distortion.

The result of a calibration of the intrinsic parameters is shown in Table 3.1. The presented uncertainties are approximately three times the standard deviations. The parameters α and k_5 are assumed to be negligible and thus set to zero to make the estimation process more robust.

Parameter	**NW**	**FWA**	**FWB**
f_x / mm	1394 ± 8.52	874.9 ± 4.48	877.8 ± 4.37
f_y / mm	1396 ± 8.53	875.8 ± 4.81	879.0 ± 4.46
c_x / mm	318.9 ± 19.0	317.9 ± 8.36	314.1 ± 8.59
c_y / mm	260.8 ± 14.8	294.3 ± 7.26	271.7 ± 7.08
α	0 ± 0	0 ± 0	0 ± 0
k_1 / mm^{-2}	-0.3245 ± 0.0804	-0.03927 ± 0.0383	-0.1608 ± 0.0359
k_2 / mm^{-4}	0.5830 ± 1.53	-0.2954 ± 0.371	0.1121 ± 0.279
k_3 / mm^{-1}	0.004018 ± 0.00179	0.007887 ± 0.00255	0.004027 ± 0.00205
k_4 / mm^{-1}	0.002602 ± 0.00170	-0.002283 ± 0.00287	-0.001657 ± 0.00219
k_5 / mm^{-6}	0 ± 0	0 ± 0	0 ± 0

Table 3.1 Results from a calibration of the intrinsic parameters. The errors are approximately three times the standard deviations.

Calibration of Extrinsic Parameters

To measure the relative position of two cameras, a set of synchronized images are captured, showing the checkerboard in the same positions from the different viewpoints of the cameras.

The calibration toolbox comes with a function that extracts the relative position of two cameras from this kind of data, and at the same time enhances the estimate of the intrinsic parameters. An inconvenient limitation is that the exact same part of the grid has to be selected in both image sets when clicking corners, though different portions of the grid may be visible in the different images. More importantly, the intrinsic parameters used in the calculation of the extrinsic parameters are extracted from the images where the grid is visible in both cameras, with no option to use predetermined measurements of the intrinsic parameters. Since the cameras may have a limited common field of view, it may then be impossible to get images where the grid covers a large part of the image, resulting in poor estimates of the nonlinearities.

Considering the arguments in the above paragraph, a new script for calibration of the extrinsic parameters was written. It uses the same set of images as input data, where the grids are identified by manually clicking at the approximate position of the grid corners in the images. The largest grid portion visible in two corresponding images is detected automatically. The 3D position of each grid, with respect to the camera used to capture the image, is then calculated using the `extrinsic_computation` function in the calibration toolbox.

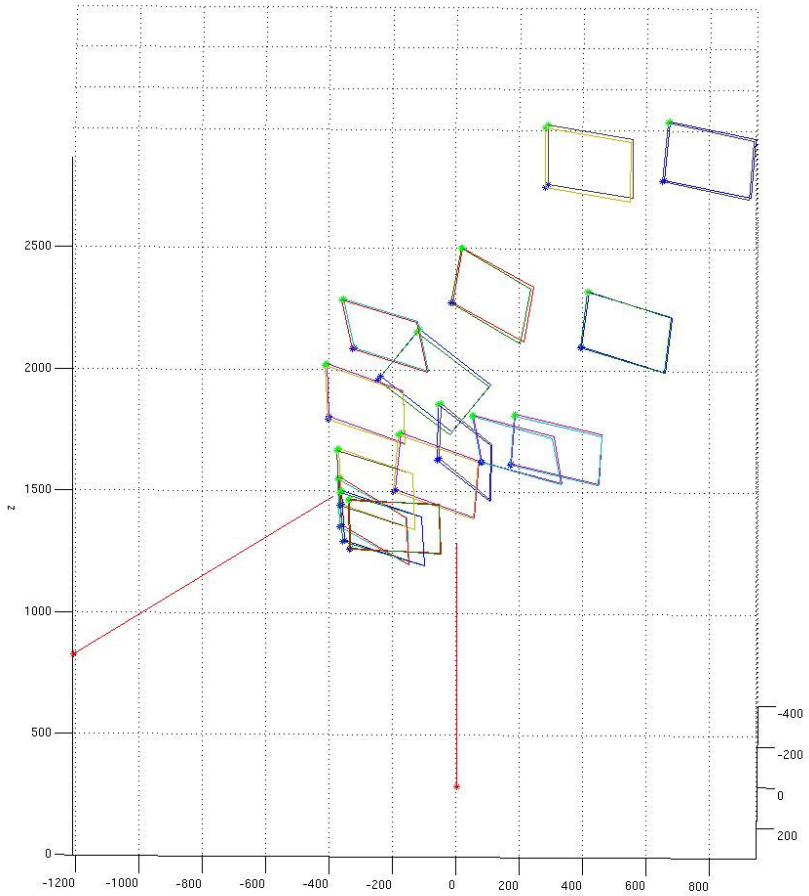


Figure 3.4 Illustration of calibration of the extrinsic camera parameters. The red lines are the forward direction of the cameras. The rectangles are the perimeters of the used checkerboard grids. Small displacements can be seen between the corresponding boards.

We now have two sets of corresponding 3D points, each belonging to one camera. The relative position of the cameras can be obtained by matching these points. To do this a weight function is defined as the root-mean-square value of the distances between the corresponding points. Its minimum is then found numerically using the Nelder-Mead simplex method [7]. Occasionally the minimization gets stuck in local minima. Therefore it is checked whether the weight function is below some threshold (a few mm is sufficient), otherwise a random value is added to the result and the minimization process is resumed.

To perform the minimization, a parameterization of the transform between the camera frames is needed. The translation has three *DOF*s (degrees of freedom) and is naturally represented as the position of one of the cameras in the coordinate system of the other camera. The representation of the 3-DOF orientation serves a few alternatives. One alternative would be to use a 3-by-3 rotation matrix, but it uses nine elements to describe a 3-DOF quantity. Since the nine elements are not independent the minimization algorithm can not simply vary all elements in the search for a minimum of the weight function. The yaw-pitch-roll representation uses three parameters, but has the drawback of singularities at pitches equal to $\pm\pi/2$. Instead a representation related to Rodrigues' rotation formula [8] was chosen, avoiding both problems of the other parameterizations. The direction of the rotation vector $\mathbf{r} = (r_1 \quad r_2 \quad r_3)^T$ defines the axis around which to make the rotation. The 2-norm of $\mathbf{r}$ defines the angle to rotate in radians.

This algorithm could quite easily be extended to calibration of several cameras simultaneously if they partly share a field of view.

Figure 3.4 visualizes the result of the calibration of the extrinsic parameters. Figure 3.5 shows the distribution of the errors. The errors are mainly along the forward directions of the cameras.

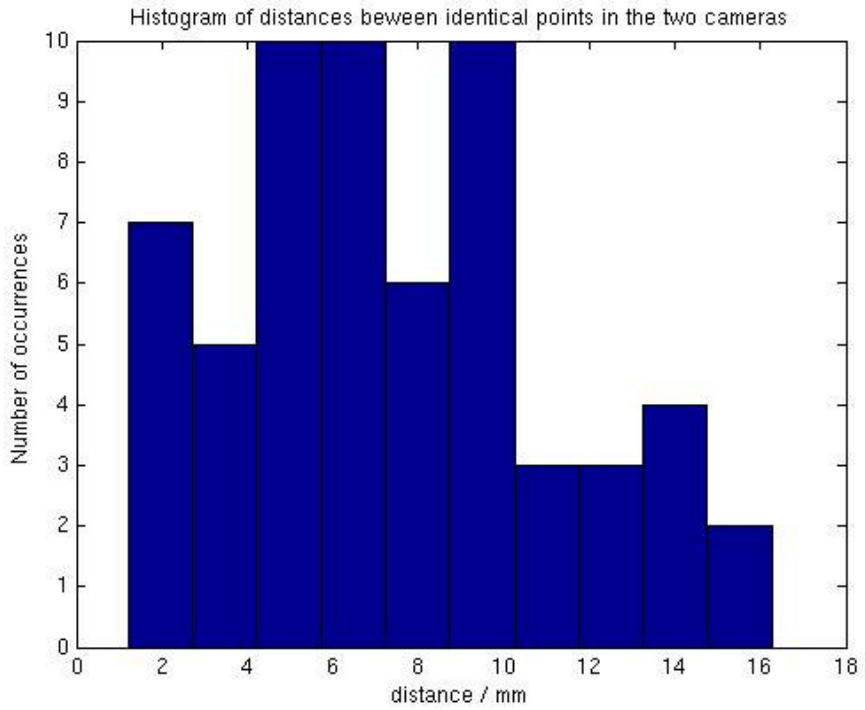


Figure 3.5 Distribution of distances between corresponding points in the calibration of the extrinsic parameters illustrated in Figure 3.4.

Figure 3.6 Dart board with markers

Calibration of the Whole Setup

When calibrating the whole setup, the first step is to measure the intrinsic parameters of each camera. The origin of the main reference frame (cf. Figure 2.1) is defined by the center of the dart board. To align the reference frame with the gravity field, the orientation of the dart board should be measured. This can be done e.g. with a plummet and a ruler. The pose of camera NW can then be determined by taking pictures of the dart board and identifying the markers, see Figure 3.6. Since cameras NW and FWA share a common field of view, their relative positions can be determined with the method described in the previous section. Similarly the relative positions of FWA and FWB can be determined. A potential problem with this procedure is accumulation of errors in the absolute pose of FWB, since it is determined through a series of relative poses. In practice, however, the errors do not seem to cause any problems.

Alternative calibration procedures were considered, where markers with known positions would be put in front of the dartboard on the floor or on tripods. If the markers are put on the floor, that puts constraints on the direction of the cameras. If the markers are put on a tripod in front of the board, it has to be removed before any darts can be thrown. In either case it would be hard to measure the position of the markers accurately.

The chosen calibration procedure could be improved by adding more cameras, so more cameras share a field of view. With a proper setup this could reduce the accumulation of errors described above.

3.3 Image acquisition

Triggering and synchronization

Two strategies for triggering of the cameras FWA and FWB were considered:

1. Let the cameras stream images continuously.
2. Trig every image individually.

A potential problem with strategy 1 is that the cameras eventually get out of synch, since they do not use the same clock. If strategy 2 is used, the program has to wait for the camera to be ready with the previous frame before sending a new trigger. This can be achieved either by reading back the status from the camera or sending the triggers with a period that is slightly longer than the time it takes for the camera to process one frame, in either case achieving a frame rate slightly lower than the bandwidth of the data transmission allows. Strategy 2 would also give a more irregular frame rate, since there would be jitter on the trigger due to the scheduling of processes on the host computer.

Considering the discussion above, strategy 1 was believed to be the best. Due to problems getting the single trig functionality of the cameras to work, the choice of strategy 1 was even more obvious.

To handle the problem of the cameras drifting out of synch, they are restarted periodically. If the cameras are restarted while there is a dart flying towards the dart board, the resulting interruption of the image flow disturbs the filter that estimates the position and velocity of the dart. Thus the restart gets postponed until there is no dart detected.

To find an appropriate period for the restart, the drift was measured. It is hard to measure the actual time the frames are received on the computer, due to the jitter induced by the scheduler. Thus the strategy used was to let the cameras stream data and count the received images, to see when one camera has sent more images than the other. When streaming images at 50 fps, the number of sent images would still be the same after one hour. This means that the drift is less than 1/50 s in 3600 s. Under the simple assumption that the cameras have different constant frame rates, a restart period of 20 s would give frames that are captured with a time difference less than $20/(50 \cdot 3600)\,\mathrm{s} \approx 0.11\,\mathrm{ms}$. This time difference is well within the limits of what is acceptable, since the used exposure time typically is 1-4 ms. A quite hard dart throw has a speed of approximately 8 m/s, resulting in that the dart moves less than 0.9 mm during the possible time difference of 0.11 ms between the time instants for exposure start.

Bayer Decoding

On most digital color cameras, including Basler A602fc, each pixel of the image sensor is covered by a red, green or blue color filter. Consequently each pixel only registers the amount of light in one part of the color spectrum. The filters are arranged in a so called Bayer pattern, depicted in Figure 3.7. There are a large number of algorithms available that use some kind of interpolation to convert a Bayer coded image to an image with red, green and blue intensities for every pixel.

The Basler A602fc has a few options for the format of the images it transfers to the computer. When using the *Raw 8* format, Bayer coded images are transferred using 8 bits per pixel. If this format is used the Bayer decoding has do

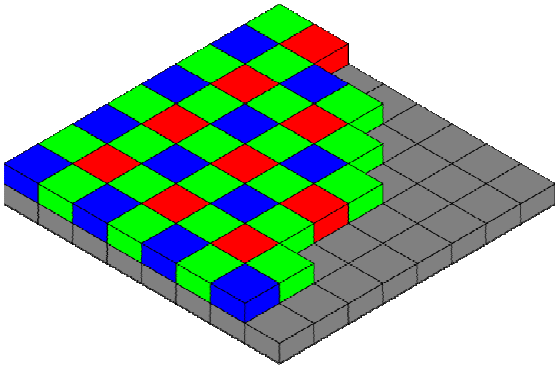


Figure 3.7 Bayer pattern.

be performed on the computer. When using the *YUV 4:2:2* format, the Bayer decoding is performed on the camera and an average of 16 bits per pixel are used in the transmission. See [11] for more information on the formats.

Using the *YUV 4:2:2* format it takes 20 ms to transfer one frame. Using the *Raw 8* format it only takes 10 ms to transfer one frame, but the computer gets some extra work decoding the Bayer pattern. The Bayer decoding takes approximately 2 ms for one frame and the rate limiting factor is the transfer of frames from the cameras, not computer load. Thus the *Raw 8* format was chosen.

3.4 Image Analysis

This section describes the process of analyzing images and extracting the image coordinates, at which there are darts. In the development of the algorithm an important consideration has been to make it simple and computationally efficient due to the high real-time requirements on the implementation

Pixel Classification

As a first step the color of each pixel is examined in order to determine whether it is likely to be part of a dart in the image. To do this the RGB (red, green, blue) image is converted to the HSV (hue, saturation, value) color space defined by (3.7) [1]. Assuming $r,g,b \in [0,1]$ this yields $h \in [0°, 360°[$ and $s,v \in [0,1]$.

$$
\begin{aligned}
h &= \begin{cases} 0, & \text{if } M = m \\ \left(60° \cdot \dfrac{g-b}{M-m} + 0°\right) \bmod 360°, & \text{if } M = r \\ 60° \cdot \dfrac{g-b}{M-m} + 120°, & \text{if } M = g \\ 60° \cdot \dfrac{g-b}{M-m} + 240°, & \text{if } M = b \end{cases} \\
s &= \begin{cases} 0, & \text{if } M = 0 \\ \dfrac{M-m}{M} = 1 - \dfrac{m}{M}, & \text{if } M \neq 0 \end{cases} \\
v &= M \\
&\text{where} \\
m &= \min(r,g,b) \\
M &= \max(r,g,b)
\end{aligned}
\tag{3.7}
$$

Intuitively h can be interpreted as the tone of the color. Large values of s give intense colors, while small values of s give shades close to gray. The value of v indicates the brightness.

To determine if a pixel is part of a dart, the criterion

$$0.55 < h/360° < 0.65 \wedge 0.2 < v < 0.7 \tag{3.8}$$

was found to work well empirically for the dart depicted in Figure 3.10. The basic idea was to use mostly the hue, since it does not vary much with different lighting conditions. Initially a minimum value of the saturation was also used, since the darts have an intense green color. This criterion was however removed later, allowing detection of darts in darker conditions without introducing much false positives. The lower limit of the brightness is needed because the calculation of the hue is very sensitive to noise at low intensities and then generates hue values in the entire range, including the hue range used to discriminate dart pixels. Similarly incorrect hue values are generated at high brightness, when some pixel intensities saturate. Consequently a maximum value of the brightness was introduced.

Using (3.8), a new image, D, can be generated, where the value of each pixel is 1 if it is likely to be part of a dart, 0 otherwise. This is expressed more formally in (3.9). An example of D can be seen in Figure 3.9 (c).

$$D(r,c) = \begin{cases} 1 & \text{if (3.8) is satisfied} \\ 0 & \text{otherwise} \end{cases}$$

for

$$r, c \in \mathbf{Z}, \quad 0 \le r \le H-1, \quad 0 \le c \le W-1 \tag{3.9}$$

where

H = image height

W = image width

Calculating the Number of Dart Pixels in a Rectangle

In the process of finding darts in an image, the number of dart pixels in a rectangle is calculated a large number of times. A naïve way to do this would be to use D from (3.9) each time and simply sum all the pixel values in the rectangle. If the rectangle is h-by-w pixels this would require $h \cdot w - 1$ additions. As an example, p in Figure 3.8 would be computed with the formula $p = \sum_{i=t}^{t+h-1} \sum_{j=l}^{l+w-1} D(i,j)$.

When the number of dart pixels in a rectangle is to be computed for a large number of rectangles in the same image, a computationally more efficient method is to use the *integral image*, I, defined by (3.10).

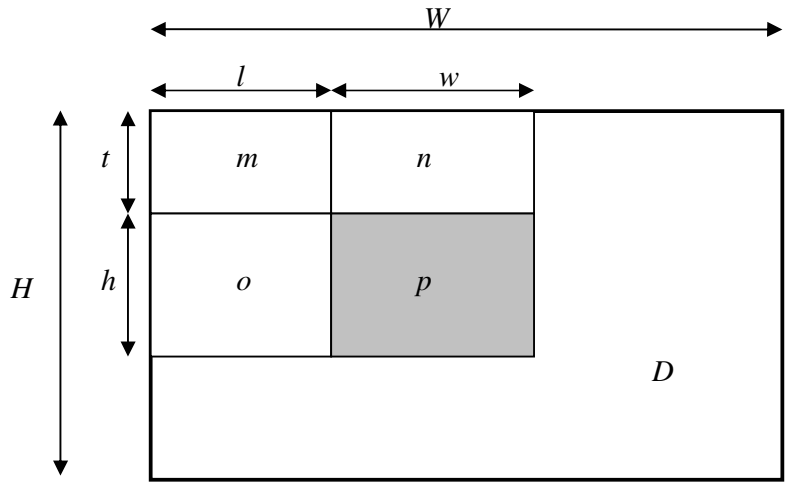


Figure 3.8 Illustration of how to compute the pixel values in a rectangle by means of the integral image, defined in (3.10). The variables *m*, *n*, *o* and *p* denote the sum of the pixel values in the respective rectangles of *D*.

$$I(r,c) = \sum_{i=0}^{r}\sum_{j=0}^{c} D(i,j)$$

for

$$r, c \in \mathbf{Z}, \quad 0 \le r \le H-1, \quad 0 \le c \le W-1 \tag{3.10}$$

where

H = image height

W = image width

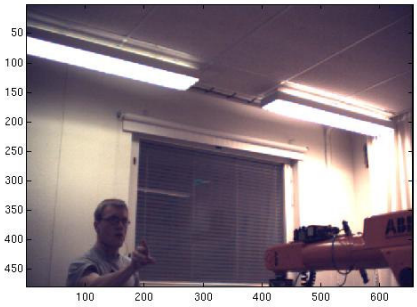
This is quite a heavy computation, but once it is done the number of dart pixels in any rectangle can be computed by only summing 4 values. This can be understood by looking at Figure 3.8. The sum of the pixel values in *D* over the different rectangles are denoted by *m*, *n*, *o* and *p*. A formula for calculating *p* and a derivation of the formula are given in (3.11).

$$\begin{aligned} &I(t+h,l+w) + I(t,l) - I(t,l+w) - I(t+h,l) = \\ &= (m+n+o+p) + (m) - (m+n) - (m+o) = \\ &= p \end{aligned} \tag{3.11}$$

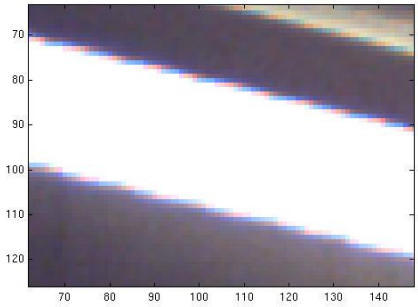
Finding Blobs

The criterion (3.8) for detecting dart pixels generates quite a large number of false positives, as can be seen in Figure 3.9 (c). A large number of these are caused by the Bayer pattern, which can be seen in Figure 3.9 (b). Because of the displacement of the color filters a large number of colors are generated at sharp edges in the images. These artifacts occur in lines that are one, or possibly two, pixels wide. Thus they can be filtered out by keeping only the pixels that are the centers of 3-by-3 pixel boxes where at least 7 out of the 9 pixels satisfy (3.8). This results in Figure 3.9 (d).

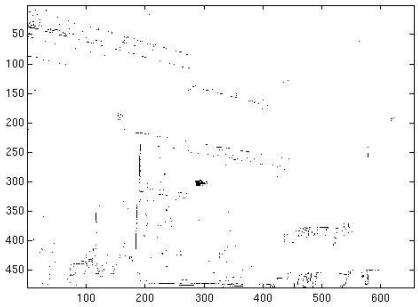
Still there may be some outliers. To filter these out, the pixel that is the center of the 5-by-5 pixel box with the largest number of pixels satisfying (3.8) is used as the estimate of the position of the dart. If several pixels share the same number of neighbors satisfying (3.8), their center of gravity is used. If the dart is close to the camera there are generally many pixels having 25 neighboring dart pixels in the 5-by-5 box. By using the maximum number of measured neighbors



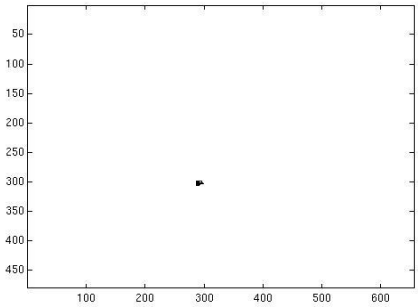
(a) Original image. There is a dart at (row, column) = (303, 290)



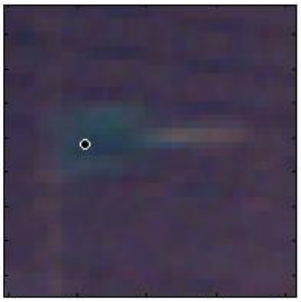
(b) Close-up showing artifacts of Bayer pattern.



(c) Pixels with colors similar to a dart.



(d) Pixels left after blob detection.



(e) Close-up of dart with estimated position marked with a white circle.

Figure 3.9 Illustration of image analysis process.

in the image instead of always using 25, it is possible to detect darts that are far away and do not cover many pixels of the image.

By means of (3.11) and the integral image (3.10) the blobs can be found efficiently with a binary search over the image. The basic idea is to discard a rectangular part of the image if it does not contain enough dart pixels to contain a dart. Otherwise the rectangle is split in half and each half is examined for the possibility of containing a dart. In this way the position of the dart is narrowed down through recursion.

Depending on the background there may still be a significant number of false positives. These can however be filtered out at a later stage by analyzing if they correspond to the expected behavior of a flying dart.

3.5 Kalman Filter

A Kalman filter is used to estimate the state of the dart and to predict its future trajectory. The general form of the filter is derived in Chapter 11 of [12] and the results are repeated here.

The process is described by the discrete time state-space model

$$\begin{aligned} x(kh+h) &= \Phi x(kh) + \Gamma u(kh) + v(kh) \\ y(kh) &= C(kh)x(kh) + e(kh) \end{aligned} \tag{3.12}$$

where h is the time step and v and e are discrete time Gaussian white noise processes with zero mean value and

$$\begin{aligned} \mathrm{E}\left[v(kh)\ v^T(kh)\right] &= R_1 \\ \mathrm{E}\left[v(kh)\ e^T(kh)\right] &= R_{12} \\ \mathrm{E}\left[e(kh)\ e^T(kh)\right] &= R_2 \end{aligned} \tag{3.13}$$

Throughout this report all elements of R_{12} are assumed to be zero, i.e. the load disturbance, v, and the measurement noise, e, are assumed to be uncorrelated. The initial state is assumed to be Gaussian distributed with

$$\mathrm{E}\left[x(0)\right] = m_0 \text{ and } \mathrm{E}\left[x(0)x^T(0)\right] = R_0 \tag{3.14}$$

Assuming that h is used as time unit, the update laws for the Kalman filter are then described by (3.15) – (3.17). Here $\hat{x}(k+1\,|\,k)$ denotes the estimate of x at sample k+1 based on measurements up to sample k.

$$\hat{x}(k+1\,|\,k) = \Phi\hat{x}(k\,|\,k-1) + \Gamma u(k) + K(k)\left(y(k) - C(k)\hat{x}(k\,|\,k-1)\right) \tag{3.15}$$

$$K(k) = \left(\Phi P(k)C^T(k) + R_{12}(k)\right)\left(R_2(k) + C(k)P(k)C^T(k)\right)^{-1} \tag{3.16}$$

$$P(k+1) = \Phi P(k)\Phi^T + R_1 - K(k)\left(\Phi P(k)C^T(k) + R_{12}(k)\right)^T$$
$$P(0) = R_0 \tag{3.17}$$

Process Model

The model used for the free-flying dart dynamics is very simple. It is assumed to fly through the air with negligible air friction. Since the image analysis does not yet have the functionality to extract the dart's orientation, it does not make much sense including the orientation in the model. Thus the dart is modeled as a particle.

A problem with this simple model is that the point that conforms well to the model is the center of mass, but only the position of the tail is measured, and the point of interest is at the tip, since this determines where the dart hits the board. For the model to be valid it is thus required that the throws are "friendly" with small rotation rates, so that the three points traverse approximately the same trajectory in space. The different parts of the dart are shown in Figure 3.10.

One problem with a wobbling dart is the displacement, that the tip and tail do not follow the same trajectory. Moreover, if the rotation of the dart gives the tale a supplementary downward velocity, the Kalman filter will think that the center of mass is moving downwards faster than it is. This error gets amplified more the longer time there is left to the impact, possibly giving hit point estimates that are far too low.

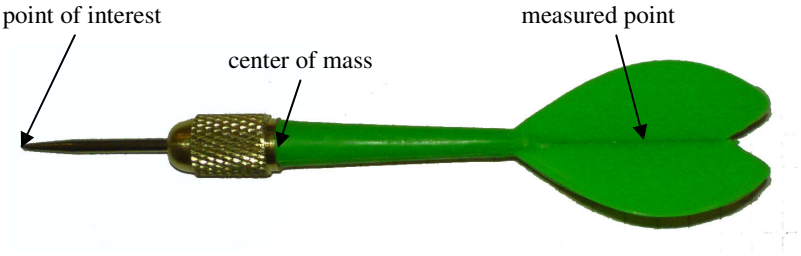


Figure 3.10 Image of dart.

Computation of Process Model Matrices

The state vector used to describe the model contains first the positions and then the velocities as described in (3.18). The positions are measured with respect to the main reference frame in Figure 2.1.

$$x = \begin{pmatrix} x_d & y_d & z_d & \dot{x}_d & \dot{y}_d & \dot{z}_d \end{pmatrix}^T \tag{3.18}$$

The continuous time description of the model is then given by (3.19).

$$\dot{x} = Ax + B \cdot 1 = \begin{pmatrix} 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \end{pmatrix} x + \begin{pmatrix} 0 \\ 0 \\ 0 \\ 0 \\ -g \\ 0 \end{pmatrix} \cdot 1 \tag{3.19}$$

One way to compute the discrete time system is described in (3.20) [12]. Note that $A^2 = 0$ in our example.

$$\begin{aligned} \Psi &= \int_0^h e^{As} ds = Ih + \frac{Ah^2}{2!} + \frac{A^2 h^3}{3!} + \ldots + \frac{A^i h^{i+1}}{(i+1)!} + \ldots \\ \Phi &= I + A\Psi \\ \Gamma &= \Psi B \end{aligned} \tag{3.20}$$

Consequently (3.12) can be used with system matrices described by

$$\Phi = \begin{pmatrix} 1 & 0 & 0 & h & 0 & 0 \\ 0 & 1 & 0 & 0 & h & 0 \\ 0 & 0 & 1 & 0 & 0 & h \\ 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \end{pmatrix}$$

$$\Gamma = \begin{pmatrix} 0 \\ -gh^2/2 \\ 0 \\ 0 \\ -gh \\ 0 \end{pmatrix} \tag{3.21}$$

$$u(kh) = 1 \quad \text{for all } k$$

Prediction

In each filtering iteration the dart trajectory is extrapolated to predict where the dart will hit the board. This is done by simply running more iterations of the Kalman filter assuming that no measurements are available. The filtering is continued until $\hat{z}_d$ is equal to the z-coordinate of the dart board. The values of $\hat{x}_d$ and $\hat{y}_d$ at that instant are used as the hit point estimate.

As the dart approaches the board, the uncertainty of the estimate gets smaller and smaller. One reason for this is that the uncertainty of the dart state estimate gets smaller with every successful measurement. Another reason is that the uncertainty in the velocity estimate contributes less to the uncertainty of the hit point, the less time there is left to the impact.

Outlier Detection

Before a measurement of the dart position is used in the Kalman filter, it is checked that it is close to what is expected based on the state estimate. Otherwise the measurement is discarded.

Each measurement is associated with a measurement vector, y, a measurement covariance, R_2, and a matrix, C, relating the measurement to the state according to $y(k) = C(k)x(k) + e(k)$, cf. (3.12). Based on the state estimate the expected measurement and its variance become

$$\begin{aligned} \hat{y} &\equiv C\hat{x} \\ m_y &\equiv \mathrm{E}[\hat{y}] = \mathrm{E}[C\hat{x}] = C\,\mathrm{E}[\hat{x}] = Cm_x \\ \hat{R}_2 &\equiv \mathrm{E}\left[(\hat{y} - m_y)(\hat{y} - m_y)^T\right] = \mathrm{E}\left[(C\hat{x} - Cm_x)(C\hat{x} - Cm_x)^T\right] = \\ &= C\,\mathrm{E}\left[(\hat{x} - m_x)(\hat{x} - m_x)^T\right]C^T = CPC^T \end{aligned} \tag{3.22}$$

Now let

$$d = y - \hat{y} \tag{3.23}$$

be the difference between the actual measurement and the expected measurement. Since the errors in y and $\hat{x}$ are assumed to be uncorrelated the covariance of d is

$$R \equiv \mathrm{cov}(d) = R_2 + \hat{R}_2 \tag{3.24}$$

The variance in the direction of d is

$$\sigma^2 = \frac{d^T}{\|d\|} R \frac{d}{\|d\|} = \frac{d^T R d}{d^T d} \tag{3.25}$$

where $\|d\|$ is the 2-norm of d. The measurement is considered to be an outlier if

$$\|d\| > a\sigma \tag{3.26}$$

i.e. if the distance between y and $\hat{y}$ is larger than a standard deviations, where a is a tuning parameter. (3.26) can be transformed to a computationally more efficient form:

$$\begin{aligned} &\|d\| > a\sigma \Leftrightarrow \|d\|^2 > a^2\sigma^2 \Leftrightarrow d^T d > a^2 \frac{d^T R d}{d^T d} \Leftrightarrow \\ &\left(\frac{d^T d}{a}\right)^2 > d^T R d \end{aligned} \tag{3.27}$$

Managing of Multiple Trajectories Simultaneously

On top of the Kalman filter is a layer that can keep track of several state vectors, each representing the trajectory of one dart. The obvious advantage of this is that

the trajectories of several darts can be tracked simultaneously. A more important advantage has to do with outlier detection.

In each iteration of the Kalman filter, the measurements are discarded if they are not close to where they are expected to be based on the current state estimate (cf. Outlier Detection). If the measurement initiating the trajectory is a false positive, successive correct measurements will be discarded, since they are not close to what would be expected after the incorrect measurement. This means that one incorrect measurement would block the system. The algorithm used to handle this situation is described by the following pseudo code:

```
for all trajectories
  if the measurement is not an outlier to this trajectory
    perform iteration of Kalman filter
  if the trajectory has not received a measurement for a while
    discard it
if the measurement did not match any existing trajectory
  create a new trajectory
```

3.6 Conversion from Image Measurements to 3D Space Measurements

Approach 1

In the first approach a pair of stereo images acquired simultaneously can only be used if the dart is detected properly in both images. The *x*, *y* and *z* coordinates of the dart at that instant are then calculated and used as input to the Kalman filter that estimates the state of the dart. The C matrix is then always $(I_{3\times3} \quad 0_{3\times3})$.

From the position of the dart in a single image it is possible to determine a line in 3D-space along which the dart must be. In this report this line will be referred to as the *viewline* (cf. Figure 3.11). If a viewline is known for the same dart from two different cameras, the position of the dart can be calculated as the point where the viewlines intersect. In practice measurement noise will cause the lines not to intersect. Instead the midpoint of the shortest possible line connecting the two viewlines can be used.

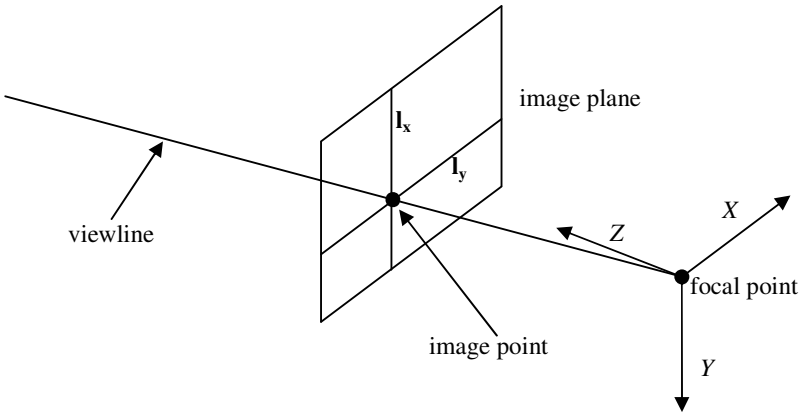


Figure 3.11 Illustration of viewline. An object projected onto some image point, can be located anywhere along the corresponding viewline.

Extracting Viewlines

To determine the viewline for a point in an image, first two lines through the point in the image are chosen. The easiest way to do this is to take one horizontal and one vertical line. Each of these lines corresponds to a plane through the focal point and the projection of this line. Finally the viewline can be determined as the intersection of these planes.

A line **l** in the images can be represented as the points fulfilling the equation

$$\mathbf{l}^{\mathbf{T}}\mathbf{x} = 0, \text{ where } \mathbf{l} = (l_1 \quad l_2 \quad l_3)^T \text{ and } \mathbf{x} = (x \quad y \quad 1)^T \tag{3.28}$$

A plane $\boldsymbol{\pi}$ in space can similarly be represented by the points fulfilling

$$\boldsymbol{\pi}^{\mathbf{T}}\mathbf{X} = 0, \text{ where } \boldsymbol{\pi} = (\pi_1 \quad \pi_2 \quad \pi_3 \quad \pi_4)^T \text{ and } \mathbf{X} = (x \quad y \quad z \quad 1)^T \tag{3.29}$$

Moreover if the plane $\boldsymbol{\pi}$ is projected onto the line **l**, then

$$\mathbf{x} \sim P\mathbf{X} \tag{3.30}$$

where P is the projection matrix of the camera, cf. (3.5). Inserting (3.30) into (3.28) we get

$$0 = \mathbf{l}^{\mathbf{T}}\mathbf{x} \sim \mathbf{l}^{\mathbf{T}}P\mathbf{X} = (P^T\mathbf{l})^T\mathbf{X} \tag{3.31}$$

Hence it can be can concluded that the points that project onto the line **l** reside in the plane

$$\boldsymbol{\pi} = P^T\mathbf{l} \tag{3.32}$$

If the coordinate of the dart in an image is (x, y), a convenient choice of lines through this point are $\mathbf{l}_\mathbf{x} = (-1 \quad 0 \quad x)^T$ and $\mathbf{l}_\mathbf{y} = (0 \quad -1 \quad y)^T$ (cf. Figure 3.11) corresponding to the planes

$$\begin{aligned} \boldsymbol{\pi}_\mathbf{x} &= P^T\mathbf{l}_\mathbf{x} \\ \boldsymbol{\pi}_\mathbf{y} &= P^T\mathbf{l}_\mathbf{y} \end{aligned} \tag{3.33}$$

The direction **v** of the viewline can then be calculated as the cross product of the normal directions of the planes:

$$\mathbf{v} = (\pi_{x1} \quad \pi_{x2} \quad \pi_{x3}) \times (\pi_{y1} \quad \pi_{y2} \quad \pi_{y3}) \tag{3.34}$$

To get a complete description of the line, some point on the line is also needed. Since all viewlines go through the focal point, **c**, it is convenient to calculate this point once for every camera and then use it for all viewlines. The focal point fulfills the relationship

$$P\mathbf{c} = \mathbf{0} \tag{3.35}$$

Setting $\mathbf{X} = \left((-R^T\mathbf{t})^T \quad 1\right)^T$ in (3.5) and recalling that R is orthonormal we get

$$P\mathbf{X} = \begin{pmatrix} R & \mathbf{t} \end{pmatrix} \begin{pmatrix} -R^T\mathbf{t} \\ 1 \end{pmatrix} = -RR^T\mathbf{t} + \mathbf{t} = \mathbf{0} \tag{3.36}$$

Consequently the focal point of P in (3.5) can be calculated as

$$\mathbf{c} = -R^T\mathbf{t} \tag{3.37}$$

To sum up, points on the viewline satisfy (3.38) for some scalar s.

$$\mathbf{X} = \mathbf{c} + s\mathbf{v} \tag{3.38}$$

Finding Line Intersection

Having two viewlines for the same dart from different cameras, indexed a and b, the position of the dart should be at the point satisfying

$$\mathbf{c_a} + s_a\mathbf{v_a} = \mathbf{c_b} + s_b\mathbf{v_b} \tag{3.39}$$

This equation is however not likely to have a solution due to measurement error. Instead we can find the solution that minimizes the sum of the squared errors of the three components. This value is also the square of the minimum distance between the viewlines. The process of estimating the dart position is illustrated in Figure 3.12. Using the definitions

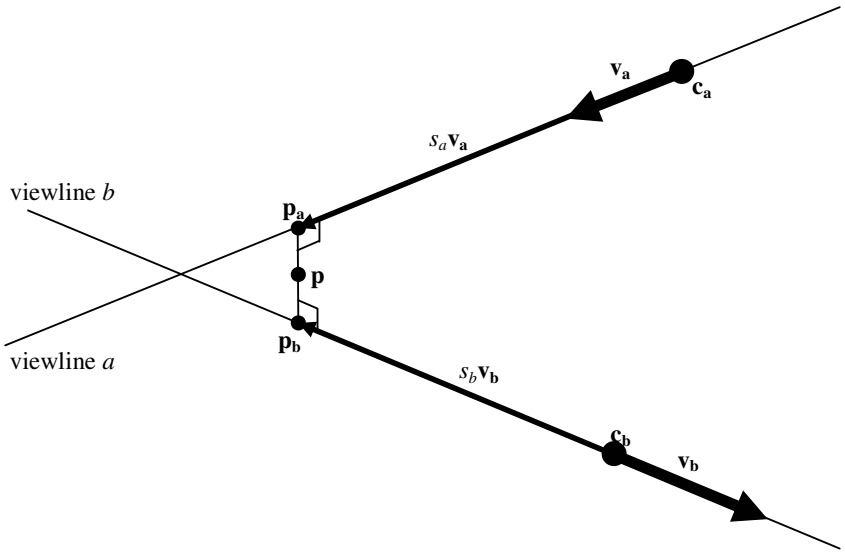


Figure 3.12 Illustration of triangulation process.

$$\mathbf{c} = -\mathbf{c_a} + \mathbf{c_b}$$
$$\mathbf{s} = \begin{pmatrix} s_a \\ s_b \end{pmatrix} \tag{3.40}$$
$$V = \begin{pmatrix} \mathbf{v_a} & -\mathbf{v_b} \end{pmatrix}$$

we can rewrite (3.39) as

$$V\mathbf{s} = \mathbf{c} + \mathbf{e} \tag{3.41}$$

where $\mathbf{e}$ are the residuals. The parameters minimizing the error $\|\mathbf{e}\|_2$ can then be solved as

$$\mathbf{s} = (V^T V)^{-1} V^T \mathbf{c} \tag{3.42}$$

The point on each line closest to the other line can now be calculated as

$$\mathbf{p_a} = \mathbf{c_a} + s_a \mathbf{v_a} \text{ and } \mathbf{p_b} = \mathbf{c_b} + s_b \mathbf{v_b} \tag{3.43}$$

It is reasonable to use the average of these points as the estimate of the dart position:

$$\mathbf{p} = (\mathbf{p_a} + \mathbf{p_b})/2 \tag{3.44}$$

Outlier detection

A simple way to detect outliers during triangulation is to examine the shortest distance $d = \|\mathbf{p_a} - \mathbf{p_b}\|$ between the viewlines. If d is above some threshold it is considered that in at least one of the images something else than the dart was found, and the triangulation returns a failure. The distribution of d can be seen in Figure 3.13. A threshold of 25 mm was found to be appropriate to discriminate outliers.

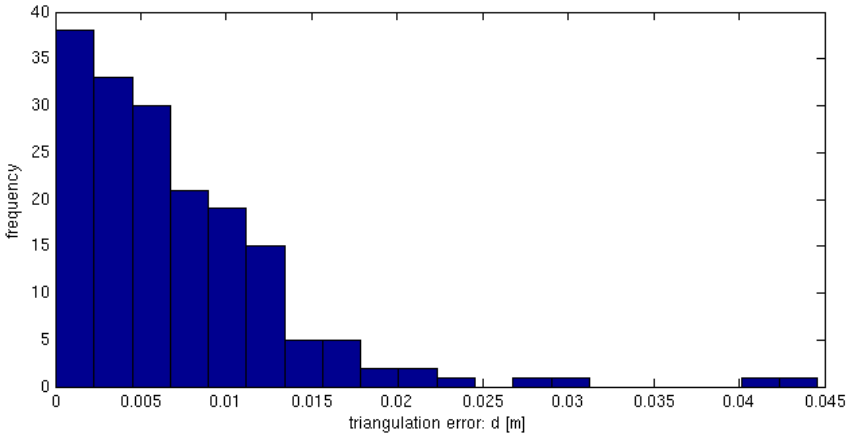


Figure 3.13 Distribution of the triangulation error, d. The histogram is the outcome of an experiment with 175 measurements.

Initialization

When two position measurements with known time difference are available it is possible to make a unique initial estimate of all states in the Kalman filter (3 position states and 3 velocity states), including covariances. These are used as $x(0)$ and $P(0)$ in (3.15) – (3.17).

Approach 2

The second approach for converting measurements in the images into input to the Kalman filter is a bit more flexible and does not perform any explicit triangulation. This approach has not yet been implemented on the real system.

The planes in (3.33) are calculated just like in the first approach. Using the more general form in (3.32) this puts the constraint

$$\boldsymbol{\pi}^{\mathrm{T}}\mathbf{X} = 0 \tag{3.45}$$

on the dart position **X**. This can be rewritten to a constraint on the state vector of the Kalman filter:

$$0 = \boldsymbol{\pi}^{\mathrm{T}}\mathbf{X} = \begin{pmatrix}\pi_1 & \pi_2 & \pi_3 & \pi_4\end{pmatrix}\begin{pmatrix}x_d\\ y_d\\ z_d\\ 1\end{pmatrix} \Leftrightarrow \begin{pmatrix}\pi_1 & \pi_2 & \pi_3\end{pmatrix}\begin{pmatrix}x_d\\ y_d\\ z_d\end{pmatrix} = -\pi_4 \Leftrightarrow$$

$$cx = -\pi_4 \text{ where } c = \begin{pmatrix}\pi_1 & \pi_2 & \pi_3 & 0 & 0 & 0\end{pmatrix} \text{ and } x = \begin{pmatrix}x_d\\ y_d\\ z_d\\ \dot{x}_d\\ \dot{y}_d\\ \dot{z}_d\end{pmatrix} \tag{3.46}$$

If $\boldsymbol{\pi}$ is normalized so $\sqrt{\pi_1^2 + \pi_2^2 + \pi_3^2} = 1$, $-\pi_4$ can be interpreted as the signed length of the orthogonal projection of the dart position onto the direction $\begin{pmatrix}\pi_1 & \pi_2 & \pi_3\end{pmatrix}$.

Each image with a measurement of the dart position gives two constraints specified by the row vector c in (3.46), one in the x direction and one in the y direction of the image. If more than one image is available this can be handled simply by adding rows in the C matrix of (3.12):

$$C = \begin{pmatrix}c_{1x}\\ c_{1y}\\ c_{2x}\\ c_{2y}\\ \vdots\end{pmatrix} \tag{3.47}$$

No explicit triangulation is done. However, if the rows in C correspond to measurements in many different directions, an estimate with low variance in all directions can be obtained. The Kalman filter simply makes sure to minimize the estimate variance based on the covariance matrix, P, of the previous estimate and the covariance matrix R_2 of the measurement.

One advantage of this second approach is that it easily extends to any number of cameras. Two more rows are simply added to C for every camera. Another advantage is that a measurement can be used even if there are no valid measurements from any other cameras. The extreme case is where camera FWA only gives measurements at samples with odd time indices and camera FWB only gives measurements at samples with even time indices. In this case the second approach would give a good estimate, while the first approach would give no estimate at all.

Initialization

The state vector is initialized to the approximate state that the dart is expected to have when it is first detected, e.g. $(x_d \quad y_d \quad z_d \quad \dot{x}_d \quad \dot{y}_d \quad \dot{z}_d) = (0 \quad 0 \quad 2 \quad 0 \quad 0 \quad -8)$, corresponding to the dart being 2 m away from the dart board straight in front of bull's eye with a velocity of 8 m/s straight towards the board. The initial estimate covariance, P_0, is set to a very large value so that the initial estimate is almost completely discarded when there are sufficient measurements. Care must be taken not to make P_0 too big, though. In (3.17), if $P(k)$ is much larger than $R_2(k)$, the first and third term are very large and approximately equal. Since these terms are subtracted from each other the calculations get sensitive to numerical errors. It is however not hard to find a good compromise for the value of P_0.

3.7 Socket Communication

A simple interface has been implemented to allow communication between the C and the Java parts of the system. The communication channel is a socket, making it easy to either run both parts on the same computer or on two different computers. The protocol includes hand shaking, clock synchronization and the sending of estimated hit points from the C part to the Java part. Together with the estimate of the hit point the expected impact time and variances are sent.

3.8 Trajectory Generation

The framework for generation of dart board trajectories has been implemented in Java. Its structure will be described briefly here.

One thread does a blocking read on the socket described in the previous section and waits for an estimate of the hit point. When a new estimate is received the thread generates a new trajectory and puts it in a buffer. Then it does another blocking read on the socket.

A second thread periodically sends position and velocity references to the robot controller. The references are taken from whatever trajectory was last put in the buffer by the other thread. This makes the system fairly robust to unreliable

transmission of hit points or long execution times for generation of trajectories, though the performance is degraded by long delays.

The socket input buffer only has space for one hit point. If a new hit point arrives before the one in the buffer is used, the old one is discarded. Consequently the trajectory generator always uses the newest available data even if the hit points arrive at a higher rate than the trajectories can be generated.

The code has not been specialized for any special kind of robot geometry. The only place where there is robot specific code is in an object describing the robot's kinematics. Hence the only action needed to use the program with another robot is to switch the kinematics object and possibly the interface to the robot.

4. Results

This chapter presents some data showing the performance of the dart tracking system as a whole and of its different components.

4.1 Camera Calibration

It takes quite long to perform camera calibration, up to 2 hours. The result of a calibration of the intrinsic parameters with error estimates is presented in Table 3.1. No explicit error estimate of the extrinsic parameters has been derived. A good indicator of the accuracy, though, is the consistency of the position estimates made by the different cameras. Figure 3.5 shows a histogram of the distances between two cameras' 3D-coordinate estimates of the same point during calibration. Based on a single image of a dart, a *viewline*, along which the dart must be, can be determined. Figure 3.13 shows a histogram of the minimum distance between the viewlines from two different cameras used to estimate the position of a dart at one time instance.

4.2 Image Analysis

The image analysis works well with few false positives and undetected darts. It is hard to give a useful quantitative measure of the error rate, since it varies much with the background and lighting conditions. An example of how the image analysis performs can be seen in Figure 3.9. False positives occur mostly when there are bluish gray objects in the background. It may appear strange that bluish gray objects are detected as green darts, but without white balancing the green color of the darts actually has a higher intensity on the blue channel than on the green channel with the cameras used, cf. (3.8).

The computation time for the image analysis of a stereo image pair is approximately 10 ms and may increase a few ms when there are very many pixels with colors similar to the color of a dart. Recalling that the system runs with a sampling period of 20 ms it becomes apparent that the image analysis is the step that consumes the most computing power, and a faster computer (or more efficient algorithms) would allow additional image analysis steps to increase robustness.

The main limitation imposed by the image analysis algorithm is that the background must not contain large objects with the same color as the darts that the system is tuned for. Another limitation is that it cannot handle the situation with more than one dart in the image. The current implementation would then report a dart at the center of gravity of the dart pixels. Further, only the position of the dart's tale is detected. Detection of the dart's orientation would improve the ability of the system to predict the future movements of the dart.

4.3 State Estimation

The entire process of extracting 3D-coordinates from the image coordinates, running an iteration of the Kalman filter and predicting where the dart will hit the

dart board plane takes a fraction of one millisecond and its computation time is hence negligible in comparison to that of the image analysis.

Approximately 1 % of the position measurements are erroneously classified as outliers by the condition (3.26). If one, instead of throwing the dart, keeps it in the hand and tries to move it as if it was thrown, it is not trivial to fool the state estimator into believing that it was an actual dart throw. It took the writer of this report several minutes of training to learn how to move the dart in such a way that the fourth position measurement was not discarded as an outlier. Hence it can be concluded that the modeling works well and most measurements are classified correctly with respect to being the position of a real dart or not.

4.4 Composite system

Figure 4.1 and Figure 4.2 show how the hit point estimate evolves as a dart approaches the dart board. Each plot corresponds to one throw. The perimeter and center of the dart board are marked in red. The estimates of where the dart will hit the board are marked with blue stars and connected with blue lines to show how the estimate evolves. The last estimate is marked with a green star. Each estimate is surrounded by an ellipse at the distance of one standard deviation of the estimate. The coordinate where the dart actually hit the board is marked with a magenta star.

When the experiments presented in Figure 4.1 were performed, the thrower tried to make the dart wobble as little as possible. The last estimate of the hit point was then generally within 1 or 2 cm of the actual hit point. When the experiments presented in Figure 4.2 were performed, the dart wobbled more since the thrower did not pay any attention to reducing the wobbling. A crude estimate, made by letting the dart thrower look at the dart, is that the dart orientation deviated up to 20-30° from the direction of movement. The error of the last estimate was then up to 5 cm.

When the exposure of an image is done it takes 20 ms for it to be transferred to the computer. In all it takes 40-50 ms from the time when the image is captured until the estimation of the hit point is received and can start being processed in the Java part of the system.

The first successful measurements are usually done when $z_d \approx 2\,\mathrm{m}$ (when the dart is approximately 2 m away from the dart board) and the last when $z_d \approx 0.5\,\mathrm{m}$. The resolution of the cameras should allow darts do be detected further away than 2 m. The reason that this rarely happens should be investigated further. One reason could be that the image analysis algorithms need to be tuned better to detect smaller objects.

When the first hit point estimate is received in the Java program there is usually 150-200 ms left to impact and when the last estimate is received there is usually 40-60 ms left. During one throw 5-10 position measurements are usually acquired.

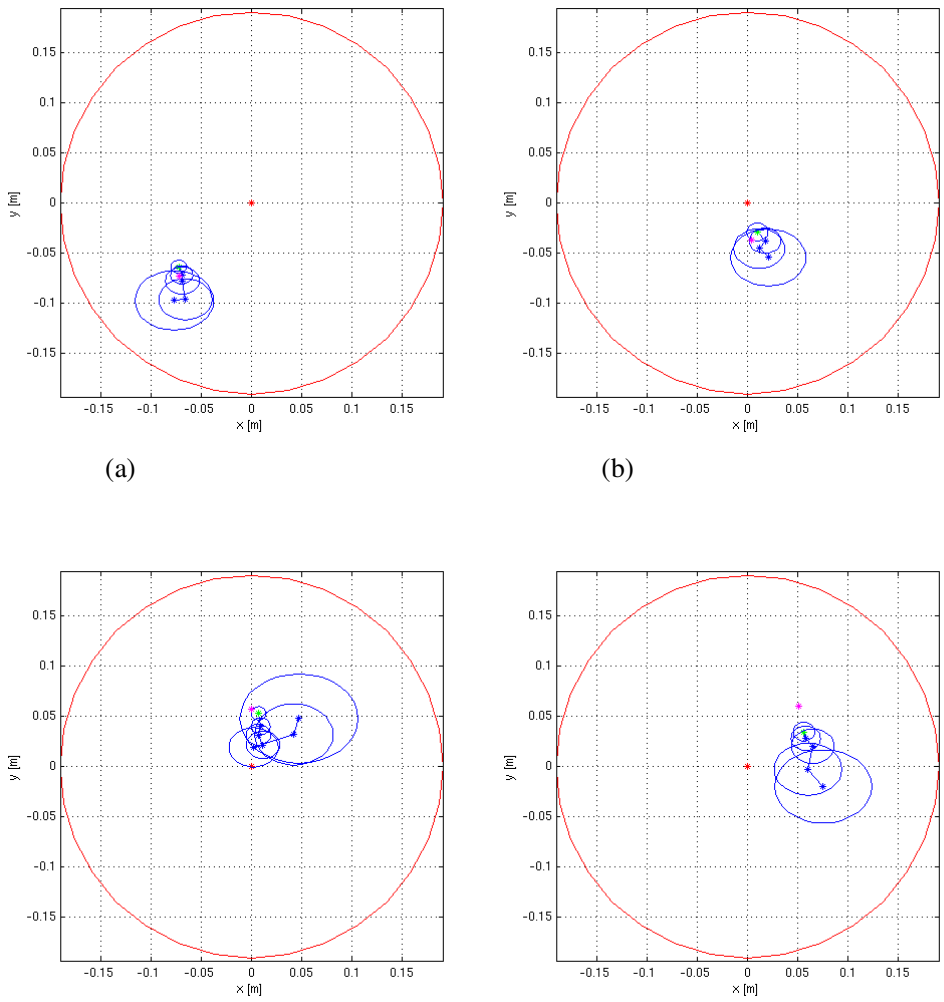


Figure 4.1 Development of estimated hit points for darts as more measurements are acquired along the trajectory of the dart. When these darts were thrown, the thrower tried to make the darts wobble as little as possible.

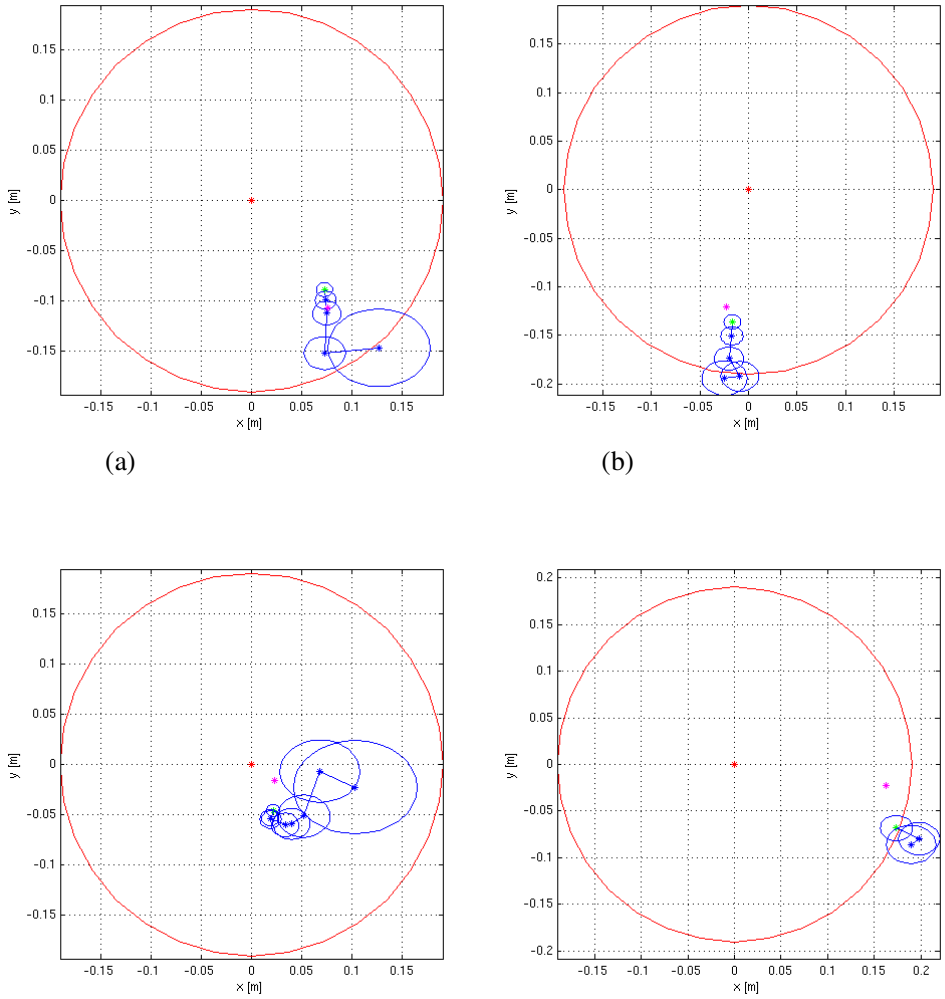


Figure 4.2 Development of estimated hit points for darts as more measurements are acquired along the trajectory of the dart. When these darts were thrown, the thrower threw the way that felt natural, without paying attention to how much this would make the dart wobble.

5. Discussion

The camera calibration procedure can be improved in several ways. One issue is how errors are accumulated. One way around this problem is to use 3D-points with known positions, which are then identified in images from the cameras. The problem that then arises is how to measure the position of the points accurately. Possibly a combination of the two methods could be used, with the drawback of a longer calibration time. Another alternative is to use more cameras, so the checkerboard can be seen by more than two cameras simultaneously. The additional cameras could then either be used to improve the position estimate of the dart in real-time (at the cost of computation time) or be removed after the calibration.

The time it takes to perform a calibration is also an issue. Currently it takes two hours to calibrate the cameras, which makes one hesitate to perform a calibration. Much time is spent on manually clicking on the corners in every image. With fully automatic corner identification the calibration time could probably come down to 30-45 minutes.

The image analysis is fairly robust to different lighting conditions and backgrounds. Still with some work it could be improved in several ways. For instance, currently different colors with the same hue in an image are perceived as different hues across different intensities and saturations on the real objects, which limits the range of lighting intensities that the system can handle with fixed parameter values. One solution to this could be performing white balancing, e.g. by capturing an image of a white reference object. The improvement of the image analysis that would have the most obvious impact on the overall system performance is detection of the dart orientation, which would allow more accurate modeling of the dart dynamics.

The implementation of the Kalman filter is straight forward, but the choice of covariance matrices (3.13) is not thoroughly investigated. Currently the variances are set to crude estimates of what seems reasonable. Possibly better tuning could e.g. filter out the wobbling of the dart tale better.

6. Conclusions

Seeing how wobbling of the dart degrades the performance, it becomes obvious that detection of the dart orientation is needed to get good performance. As a first step this could be used without making a more complex model of the dart dynamics. The orientation of the dart and the position of the tail can be used to calculate the position of the center of mass. If this is used as input to the Kalman filter describing a particle, the accuracy could probably be improved a lot.

With the current frame rate of 50 fps 5-10 position measurements are made during one throw. This seems to be quite sufficient, but still more measurements could probably improve the measurement accuracy. If the wobbling of the dart should be modeled, its dynamics are probably so fast that a higher frame rate is needed. This is not possible when the full resolution of the cameras is used. One solution is to use prediction to calculate where the dart is expected to be seen and only capture this part of the image, thus reducing time for data transfer and image analysis.

The system typically has to move the board to its destination in less than 150-200 ms. Recalling the feasibility analysis in Section 1.3 it was concluded that it takes the robot approximately 120 ms to move the dart board 0.2 m. This distance is probably longer than is usually needed (depending on the skill of the thrower), but on the other hand the final destination is not known from the beginning, causing the movement time to be longer. On top of that the trajectory generation will also cause some delay. The numbers indicate that the system has the performance needed to catch a dart, but the margins are not big. If possible, optimization of the algorithms to reduce the delays would be useful.

7. Future Work

The project described in this thesis serves many areas of future development. This chapter lists a few of them:

- The most obvious is to finish the trajectory generation and robot control, so the system can start catching darts.
- Measure the angle of the dart in the images and use this in a more advanced model of the dart dynamics.
- Add a third high speed camera. This would help making good estimates of the dart orientation. Using only two cameras it is hard to measure rotations with a rotation axis orthogonal to the plane containing the cameras and the dart.
- Use the state estimate of the Kalman filter to calculate the expected position and size of the dart in the images. This can be fed back to the image analysis algorithms to make them faster and more accurate. Similarly it can be used to read in smaller parts of the images from the cameras, reducing the time for data transfer.
- Let the image analysis algorithms vary the measurement uncertainty based on the image quality.
- Make the image analysis algorithms more robust to different lighting conditions and backgrounds.
- Finish the image analysis algorithms for detecting a dart on the dart board, making it possible to automatically measure the position where the dart actually hit the board.
- Make it possible to detect more than one dart in every image. This could be used in a future extension where the sum of the darts' distances from bull's eye should be minimized or the score on a dart board should be maximized.
- Make it possible to catch darts with high rotation rates in any direction. This puts high demands on the modeling and prediction. It also makes it useful to move the dart board in the z-direction to make sure the dart always hits the board with the tip first.
- Make it possible to use fully automatic grid detection in camera calibration. This would remove the time consuming step where grid corners have to be marked manually.

References

[1] Wikipedia page on “HSL and HSV”. Visited August 2008. http://en.wikipedia.org/wiki/HSL_color_space

[2] libdc1394 Homepage. Visited August 2008. http://damien.douxchamps.net/ieee1394/libdc1394/

[3] The OpenCV Wiki. Visited August 2008. http://opencvlibrary.sourceforge.net/

[4] Camera Calibration Toolbox for Matlab by Jean-Yves Bouguet, California Institute of Technology. Site visited August 2008. http://www.vision.caltech.edu/bouguetj/calib_doc/

[5] Multi-Camera Self-Calibration by Tomas Svoboda, Czech Technical University. Site visited August 2008. http://cmp.felk.cvut.cz/~svoboda/SelfCal/index.html

[6] EasyCal from University of Pennsylvania. Site visited August 2008. http://www.cis.upenn.edu/~kostas/tele-immersion/research/downloads/EasyCal/

[7] Wikipedia page on “Nelder-Mead method”. Visited August 2008. http://en.wikipedia.org/wiki/Nelder-Mead_method

[8] Wikipedia page on “Rodrigues’ rotation formula”. Visited August 2008. http://en.wikipedia.org/wiki/Rodrigues'_rotation_formula

[9] Forsyth D. A. and Ponce J., “Computer Vision A Modern Approach”, Prentice Hall, 2003

[10] DC 1394-based Digital Camera Specification, IIDC (Instrumentation & Industrial Digital Camera), available e.g. at http://damien.douxchamps.net/ieee1394/libdc1394/iidc/

[11] Basler A600f User’s Manual, Document number DA00056107, 2 March 2005

[12] Åström K. J. and Wittenmark B., “Computer Controlled Systems” Prentice Hall, 1997

[13] Fully Automatic Camera and Hand Eye Calibration by Christian Wengert, ETH Zürich. Site visited August 2008. http://www.vision.ee.ethz.ch/~cwengert/calibration_toolbox.php

[14] Enhanced Matlab Camera Calibration Toolbox by Graphics and Media Lab, Moscow State University. Site visited August 2008. http://graphics.cs.msu.ru/en/research/calibration/index.html

[15] Sven Gestegård Robertz, Roger Henriksson, Klas Nilsson, Anders Blomdell, and Ivan Tarasov *Using real-time Java for Industrial robot control*, in Proceedings of the 5th International Workshop on Java Technologies for Real-time and Embedded Systems, Sepember 26-28, 2007, Vienna, Austria. Paper available at http://www.cs.lth.se/home/Sven_Robertz/publ/JTRES07.html

[16] Homepage of EtherCAT Technology Group, visited August 2008. http://ethercat.org/default.htm