Vision Based Tracker for Dart-Catching Robot

Linderoth, Magnus; Robertsson, Anders; Åström, Karl; Johansson, Rolf

Published in:
IFAC Proceedings Volumes (IFAC-PapersOnline)

2009

Vision Based Tracker for Dart Catching Robot

Magnus Linderoth*. Anders Robertsson. Karl Åström***. Rolf Johansson******

*Department of Automatic Control, LTH, Lund University, SWEDEN
(Tel: +46 46 222 0847; e-mail: magnus.linderoth@control.lth.se).
**Department of Automatic Control, LTH, Lund University, SWEDEN
(Tel: +46 46 222 8790; e-mail:anders.robertsson@control.lth.se).
***Centre for Mathematical Sciences, LTH, Lund University, SWEDEN
(Tel: +46 46 222 4548; e-mail: kalle@maths.lth.se).
****Department of Automatic Control, LTH, Lund University, SWEDEN
(Tel: +46 46 222 8791; e-mail: rolf.johansson@control.lth.se).

Abstract: This paper describes how high-speed computer vision can be used in a motion control application. The specific application investigated is a dart catching robot. Computer vision is used to detect a flying dart and a filtering algorithm predicts its future trajectory. This will give data to a robot controller allowing it to catch the dart. The performance of the implemented components indicates that the dart catching application can be made to work well. Conclusions are also made about what features of the system are critical for performance.

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

1. INTRODUCTION

In robot applications it is often desirable to have systems that respond quickly to input from the robot environment. 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 image analysis algorithms. Some image properties may be very difficult to extract and the execution time is often significant.

This paper describes how high speed computer vision can be used 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 Fig. 1. A dart board was mounted on a robot and when a dart was thrown toward the board, the robot should move the board so the dart always hit the bull's eye. The detection of the dart was performed with cameras that provided data to the algorithms that estimated the position and future trajectory of the dart. In turn this information should be used to move the dart board to the correct position.

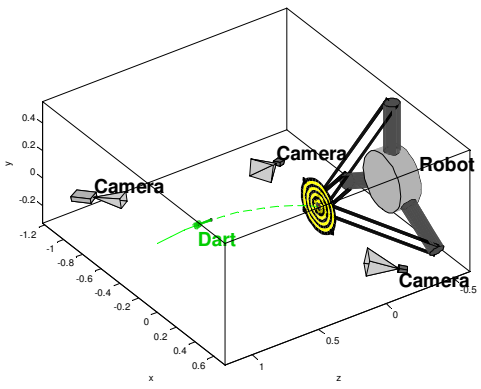


Fig. 1. Overview of the experiment setup.

An overview of object tracking techniques can be found in e.g. (Yilmaz et al. 2006). The focus of my work has been to use the simplest possible algorithm that does the job robustly in order to optimize for speed.

The dart catching application is closely related to the table tennis playing robot, which has been treated in numerous works, e.g. (Matsushima et al. 2003), which uses an adaptive black-box model for prediction.

2. METHODS

2.1 Hardware and System Architecture

Two cameras were located one on each side of the dart board. The application required good real-time camera performance and for this two Basler A602fc with an IEEE1394 serial interface (also known as Firewire) were used. They could supply color images at resolutions up to 656×490 pixels with a maximum frame rate of 100 fps at full resolution. A third slower camera was pointed toward the dart board to evaluate where the dart hit.

Most off-line calculations were performed in Matlab®. Image acquisition, image analysis and state estimation were performed in C for speed and because the API used for image acquisition was made in C. The trajectory generation and robot control was done in Java, because the interface to the robot was made in Java. A schematic overview of the system

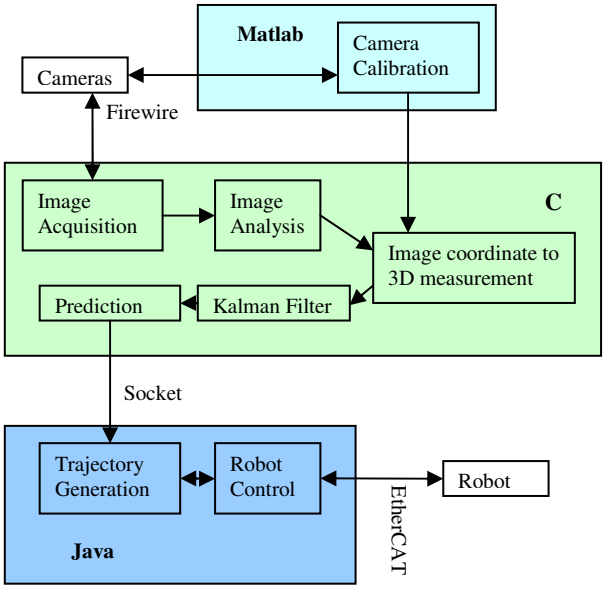


Fig. 2. A schematic overview of the components in the system and how they communicate.

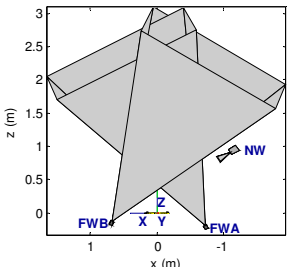


Fig. 3. Top view of the setup illustrating the stereo coverage. The cones show the field of view for the respective cameras.

architecture can be seen in Fig. 2. For more information about the robot control system, see (Robertz et al. 2007).

2.2 Camera Positioning

Many different ways of positioning the cameras were possible. The chosen positions of the high speed cameras were on each side of the dart board, pointing toward the thrower, illustrated in Fig. 3. This positioning allowed the cameras to see the dart at a big distance if the thrower was straight in front of the board. The dart catcher application had a higher requirement on the accuracy of the estimated dart trajectory when the dart was close to the dart board, which was fulfilled by the chosen camera positioning, since the dart was close to the cameras and observed from better angles when it was close to the board.

2.3 Camera Calibration

For camera calibration the Camera Calibration Toolbox for Matlab® (Bouguet 2008) by Jean-Yves Bouguet was used. It uses series of images of a checkerboard pattern in different positions to extract both intrinsic camera parameters and the relative position of the cameras in a stereo pair.

The stereo camera calibration feature in the calibration toolbox by Bouguet was not well adapted to the needs for this project, so a new script for calibration of the extrinsic parameters was developed. It used a function in the toolbox to extract the 3D position of a checkerboard grid with respect to a camera with known intrinsic parameters. To measure the relative position of two cameras, a set of synchronized images was captured, showing the checkerboard in the same positions from the different viewpoints of the cameras. Two sets of 3D points were then obtained, each belonging to one camera. The relative position of the cameras could be obtained by matching these points. For this purpose, a weight function was defined as the root-mean-square value of the distances between the corresponding points. Its minimum was then found numerically using the Nelder-Mead simplex method (Nelder and Mead 1965).

2.4 Image Analysis

In the development of the image analysis algorithms an important consideration was 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 was examined in order to determine whether it was likely to be part of a dart in the image. To do this the RGB (red, green, blue) image was converted to the HSV (hue, saturation, value) color space. To determine if a pixel was part of a dart, the likelihood criterion

$$(0.55 < h/360° < 0.65) \wedge (0.2 < v < 0.7) \quad (1)$$

was found to work well empirically for the dart depicted in Fig. 4. The basic idea was to use mostly the hue, since it does not vary much with different lighting conditions. Bounds on the intensity were used to discard ranges where the hue calculation was not reliable. Using (1), a new image could be generated, where the value of each pixel was 1 if it was likely to be part of a dart, 0 otherwise. An example of this can be seen in Fig. 9 (c).

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 was calculated a large number of times. To do this efficiently the integral image was first calculated. This was quite a heavy computation, but once it was done the number of dart pixels in any rectangle could be computed by only summing 4 values.

Finding Darts

The criterion (1) for detecting dart pixels generated quite a large number of false positives, as can be seen in Fig. 9 (c). A large number of these were caused by the Bayer pattern color decoding, which can be seen in Fig. 9 (b). Because of the displacement of the color filters a large number of colors were generated at sharp edges in the images. These artifacts

occurred in lines that were one, or possibly two, pixels wide. Thus they could be eliminated by only keeping the pixels that were the centers of 3-by-3 pixel boxes where at least 7 out of the 9 pixels satisfied (1). This resulted in Fig. 9 (d).

Still, some outliers remained. To filter these out, the pixel that was the center of the 5-by-5 pixel box with the largest number of pixels satisfying (1) was used as the estimate of the position of the dart. If several pixels shared the same number of neighbors satisfying (1), their center of gravity was used. By means of the integral image the darts could be found efficiently with a binary search over the image. The basic idea was to discard a rectangular part of the image if it did not contain enough dart pixels to contain a dart. Otherwise the rectangle was split in half and each half was recursively examined for the possibility of containing a dart.

2.5 State Estimation and Prediction

A standard Kalman filter (Kalman 1960) was used to estimate the state of the dart and to predict its future trajectory. The general form of the filter is derived e.g. in (Åström, Wittenmark 1997).

Model of Dart Flight Dynamics

The dart was assumed to fly with negligible air friction. Since the image analysis did not have the functionality to extract the dart's orientation, the dart was modeled as a particle. This was represented by the state-space model

$$\begin{aligned} x(kh+h) &= \Phi x(kh) + \Gamma u(kh) + v(kh) \\ y(kh) &= C(kh)x(kh) + e(kh) \end{aligned} \tag{2}$$

with the state vector $x = \begin{pmatrix} x_d & y_d & z_d & \dot{x}_d & \dot{y}_d & \dot{z}_d \end{pmatrix}^T$ and

$$\Phi = \begin{pmatrix} 1 & 0 & 0 & h & 0 & 0 \\ 0 & 1 & 0 & 0 & h & 0 \\ 0 & 0 & 1 & 0 & 0 & h \\ 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \end{pmatrix}, \quad \Gamma = \begin{pmatrix} 0 \\ -gh^2/2 \\ 0 \\ 0 \\ -gh \\ 0 \end{pmatrix}, \quad u(kh) \equiv 1$$

where h is the time step, g is the earth gravitation and v and e are discrete time Gaussian white noise processes.

A problem with this simple model was that the point that conformed well to the model was the center of mass, but only the position of the tail was measured, and the point of interest was at the tip, since this determined where the dart hit the board. For the model to be good it was thus required that the throws were "friendly", so that the three points traversed approximately the same trajectory in space. A wobbling tail would cause bad velocity estimates that got amplified when the trajectory was extrapolated to the dart board. The different parts of the dart are shown in Fig. 4.

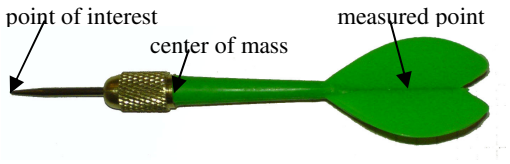


Fig. 4. Image of dart.

Prediction

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

Outlier Detection

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

Each measurement, y, and each expected measurement based on the state estimate, $\hat{y} = C\hat{x}$, were associated with a covariance matrix. Hence the covariance of the difference $d = y - \hat{y}$ and a corresponding confidence ellipsoid could be calculated. For the measurement to be accepted, d had to be within this ellipsoid.

Managing of Multiple Trajectories Simultaneously

On top of the Kalman filter was a layer that could keep track of several state vectors, each representing the trajectory of one dart. The obvious advantage of this was that the trajectories of several darts could be tracked simultaneously. A more important advantage had to do with outlier detection. In each iteration of the Kalman filter, the measurements were discarded if they were not close to where they were expected to be (cf. Outlier Detection). If the measurement initiating the trajectory were a false positive, successive correct measurements would be discarded, since they were not close to what was expected after the incorrect measurement. This meant 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
        throw it away
if the measurement did not match any existing
trajectory
    create a new trajectory
```

2.6 Conversion from Image Measurements to 3D Space Measurements

Two approaches were considered for converting the image coordinates of the darts to coordinates in 3D space.

Approach 1

In the first approach a pair of stereo images acquired simultaneously could be used only if the dart was detected properly in both images. The *x*, *y*, and *z* coordinates of the dart at that instant were then calculated and used as input to the Kalman filter that estimated the state of the dart. When the position of the dart was detected at two time samples, an estimate of both position and velocity could be estimated and be used as the initial state to the Kalman filter.

From the position of the dart in a single image it was possible to determine a line in 3D-space along which the dart must be. In this paper this line will be referred to as the *viewline* (cf. Fig. 5). If viewlines for two cameras were known for the same dart, the position of the dart could be calculated as the point where the viewlines intersected. In practice, measurement noise caused the lines not to intersect. Instead, the points closest to each other ($\mathbf{p_a}$ and $\mathbf{p_b}$ in Fig. 6) were computed and the midpoint of the line connecting them, $\mathbf{p} = (\mathbf{p_a} + \mathbf{p_b})/2$, was used as an estimate of the dart position. A simple way to detect outliers during triangulation was to examine the shortest distance

$$d = \left\| \mathbf{p_a} - \mathbf{p_b} \right\|_2 \tag{3}$$

between the viewlines. If *d* was above some threshold it was considered that in at least one of the images something else than the dart was found.

Approach 2

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

A viewline could be expressed as the intersection of two planes, e.g. the one containing the focal point and $\mathbf{l_x}$, and the one containing the focal point and $\mathbf{l_y}$ in Fig. 5. Each of these planes puts a constraint of the form $\boldsymbol{\pi}^{\mathbf{T}}\mathbf{X} = 0$ on the dart position $\mathbf{X}$. This could be rewritten to a constraint on the state vector of the Kalman filter:

$$0 = \boldsymbol{\pi}^{\mathbf{T}}\mathbf{X} = \begin{pmatrix}\pi_1 & \pi_2 & \pi_3 & \pi_4\end{pmatrix}\begin{pmatrix}x_d & y_d & z_d & 1\end{pmatrix}^T \Leftrightarrow$$
$$-\pi_4 = cx$$
$$\text{where } c = \begin{pmatrix}\pi_1 & \pi_2 & \pi_3 & 0 & 0 & 0\end{pmatrix} \tag{4}$$
$$\text{and } x = \begin{pmatrix}x_d & y_d & z_d & \dot{x}_d & \dot{y}_d & \dot{z}_d\end{pmatrix}^T$$

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 gave two constraints, each specified by a row vector *c* in (4), one in the *x* direction and one in the *y* direction of the image. If several images were available, this was handled simply by adding rows in the measurement matrix *C* of the model (2):

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

No explicit triangulation was done. However, if the rows in *C* corresponded to measurements in many different directions, an estimate with low variance in all directions could be obtained.

The Kalman filter was initialized to some reasonable state with a big variance in all directions. Thus, the actual value of the initial state was negligible when measurements had been done with *c* in (4) pointing in many different directions.

One advantage of this second approach was that it easily extended to any number of cameras. Two more rows were simply added to *C* for every camera. Another advantage was that a measurement could be used even if there were no valid measurements from any other cameras.

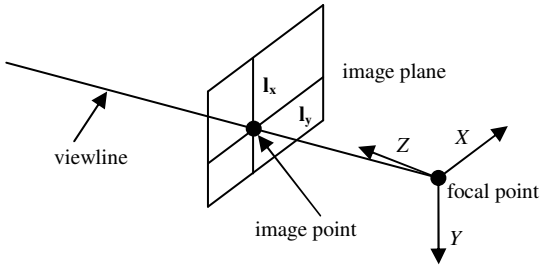


Fig. 5. Illustration of viewline. An object projected onto some image point, can be located anywhere along the corresponding viewline.

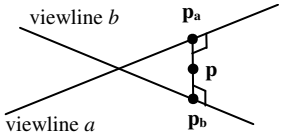


Fig. 6. Illustration of triangulation process.

3. RESULTS

3.1 Camera Calibration

Performing camera calibration took quite long, up to 2 hours. 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. Fig. 7 shows a histogram of the distances between two cameras' 3D-coordinate estimates of the same point during calibration. Fig. 8 shows a histogram of *d* in (3) for flying darts. Small values of *d* indicate good accuracy in the image analysis and camera calibration.

3.2 Image Analysis

The image analysis worked well with few false positives and undetected darts. It was hard to give a useful quantitative measure of the error rate, since it varied much with the

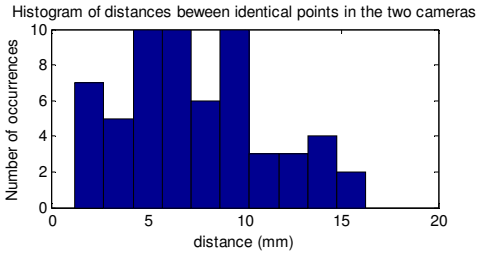


Fig. 7. Distribution of distances between corresponding points in the calibration of the extrinsic parameters.

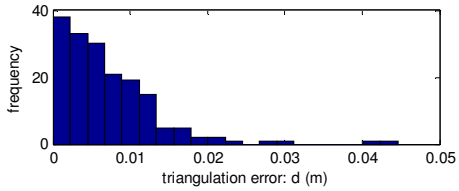


Fig. 8. Distribution of the triangulation error, *d*.

background and lighting conditions. An example of how the image analysis performed can be seen in Fig. 9.

The computation time for the image analysis of a stereo image pair was approximately 10 ms. Considering that the system ran with a sampling period of 20 ms it becomes apparent that the image analysis was the step that consumed 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 was that the background must not contain large objects with the same color as the darts. Another limitation was that it could not handle the situation with more than one dart in the image correctly. Further, detection of the dart's orientation would improve the ability of the system to predict the future movements of the dart.

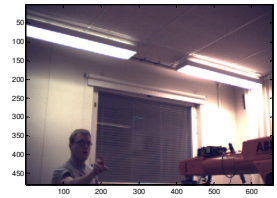
3.3 State Estimation

The entire process of extracting 3D-coordinates from the image coordinates, running an iteration of the Kalman filter and predicting where the dart would hit the dart board plane took a fraction of a millisecond and its computation time was hence negligible in comparison to that of the image analysis.

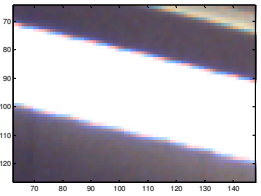
Approximately 1 % of the position measurements were erroneously classified as outliers by the state filter. If one, instead of throwing the dart, kept it in the hand and tried to move it as if it was thrown, it was not trivial to fool the state estimator into believing that it was an actual dart throw. Hence it can be concluded that the modeling worked well and most measurements were classified correctly with respect to being the position of a real dart or not.

3.4 Composite System

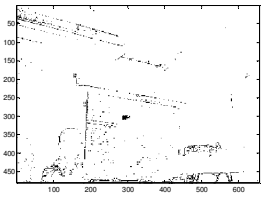
Fig. 10 and Fig. 11 show how the hit point estimate evolved as a dart approached the dart board. Each plot corresponds to one throw. The perimeter and center of the dart board are



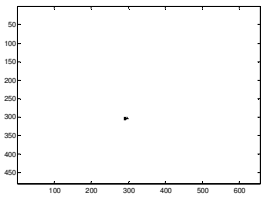
(a) Original image. There is a dart at (r, c) = (303, 289).



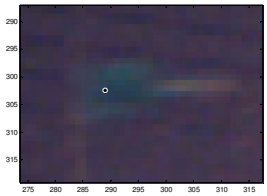
(b) Close-up showing artifacts of Bayer pattern.



(c) Pixels with colors similar to a dart. Many false positives.



(d) Pixels left after dart detection.



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

Fig. 9. Illustration of image analysis process.

marked in red. The estimates of where the dart would hit the board are marked with blue stars and connected with blue lines to show how the estimate evolved. 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 Fig. 10 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 Fig. 11 were performed, the dart wobbled more since the thrower did not pay any attention to reducing the wobbling. It was estimated that the dart orientation deviated up to 30° from the direction of movement. The error of the last estimate was then up to 5 cm.

In all it took 40-50 ms from the time when an image was captured until the estimation of the hit point was received and could start being processed in the Java part of the system.

The first successful measurements were usually done when $z_d \approx 2$ m (when the dart was approximately 2 m away from the dart board) and the last when $z_d \approx 0.5$ m.

When the first hit point estimate was received in the Java program there was usually 150-200 ms left to impact and when the last estimate was received there was usually 40-60

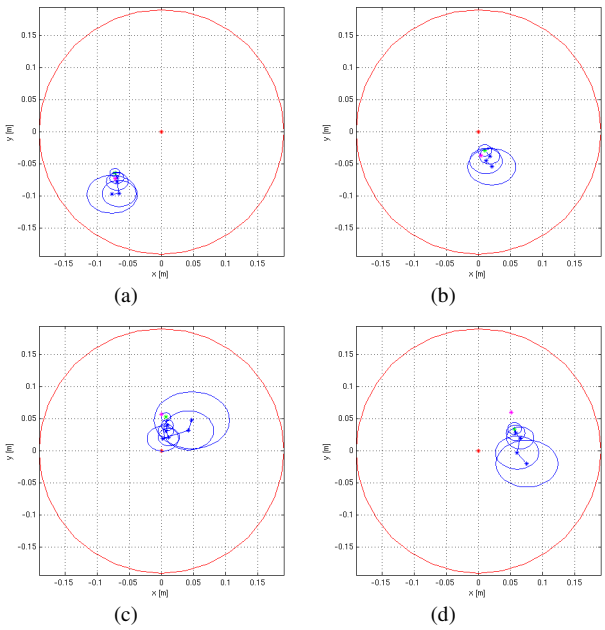


Fig. 10. Development of estimated hit points for darts thrown in a “friendly” non-wobbling way.

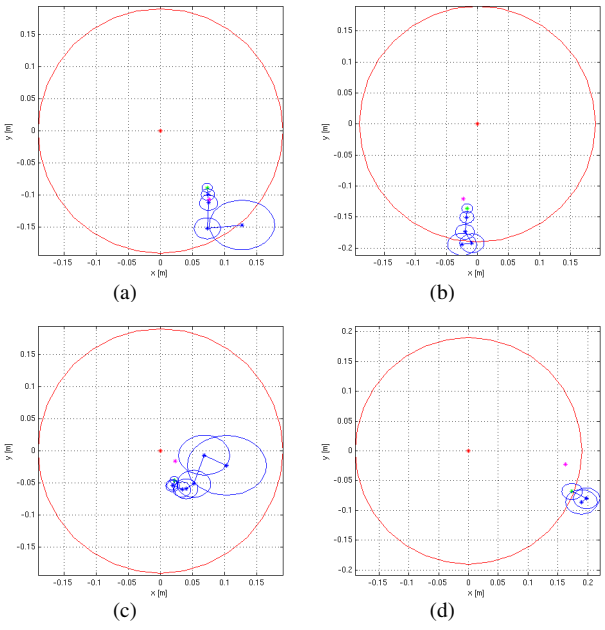


Fig. 11. Development of estimated hit points for darts thrown in a “non-friendly” wobbling way.

ms left. During one throw 5-10 position measurements were usually acquired.

4. DISCUSSION

The two hours it took to calibrate the cameras was an issue, since it made one hesitate to perform a calibration. Much time was 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 implementation of the Kalman filter was straight forward, but the choice of covariance matrices has not been thoroughly investigated. The variances were set to crude estimates of what seemed reasonable. Possibly better tuning could e.g. filter out the wobbling of the dart tail better.

With a frame rate of 50 fps 5-10 position measurements were made during one throw. This seemed quite sufficient, but if the wobbling of the dart should be modeled, its dynamics are probably so fast that a higher frame rate is needed. To allow this, prediction could be used 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.

CONCLUSIONS

Seeing how wobbling of the dart degraded the performance, it became obvious that detection of the dart orientation was 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 could be used to calculate the position of the center of mass. If this was used as input to the Kalman filter describing a particle, the accuracy could probably be improved a lot.

The system typically had to move the board to its destination in less than 200 ms. For comparison a robot with a maximum acceleration of 6*g* can move the dart board 0.2 m in 120 ms. This distance is probably longer than is usually needed, but on the other hand the final destination is not known from the beginning, causing the movement time to be longer. The numbers indicate that the system had the performance needed to catch a dart, but the margins were not large.

REFERENCES

Bouguet, J.Y. *Camera Calibration Toolbox for Matlab*, California Institute of Technology. Site visited August 2008. http://www.vision.caltech.edu/bouguetj/calib_doc/

Kalman, R.E. *A new approach to linear filtering and prediction problems*, Trans. ASME—J. Basic Engineering, 82 (1960), pp. 35–45.

Matsushima, M., Hashimoto, T. and Miyazaki, F. *Learning to the Robot Table Tennis Task – Ball Control & Rally with a Human*, Proc. 2003 IEEE Int. Conf. SMC, pp. 2962-2969, 2003.

Nelder, J.A. and Mead, R. (1965). *A simplex method for function minimization*, Comput. J., 7, pp. 308–313.

Robertz, S.G., Henriksson, R., Nilsson, K., Blomdell, A., Tarasov, I. *Using Real-time Java for Robot Control*, Journal 2007 IEEE Conference on Emerging Technologies & Factory Automation (EFTA 2007), pp. 1453-1456, 2007.

Yilmaz, A., Javed, O. and Shah, M. *Object Tracking: A Survey*, ACM Computing surveys, Vol. 38, No. 4, pp. 1-45, 2006.

Åström, K.J. and Wittenmark, B. (1997). *Computer Controlled Systems*, Chapter 11. Prentice Hall, Upper Saddle River, New Jersey