Proceedings of the 2001 IEEE/RSJ
International Conference on Intelligent Robots and Systems
Maui, Hawaii, USA, Oct. 29 - Nov. 03, 2001

Off-the-Shelf Vision for a Robotic Ball Catcher

U. Frese B. Bäuml S. Haidacher G. Schreiber I. Schäfer M. Hähnle
G. Hirzinger

Institute of Robotics and Mechatronics
German Aerospace Center (DLR Oberpfaffenhofen)
82230 Wessling, GERMANY
www.robotic.dlr.de Email: *Udo.Frese@dlr.de*

Abstract

In this paper we present a system for catching a flying ball with a robot arm using off-the-shelf components (PC based system) for visual tracking. The ball is observed by a large baseline stereo camera, comparing each image to a slowly adapting reference image. We track and predict the target position using an Extended Kalman Filter (EKF), also taking into account the air drag. The calibration is achieved by simply performing a few throws and observing their trajectories, as well as moving the robot to some predefined positions. The robustness of the system was demonstrated at the Hannover Fair 2000.

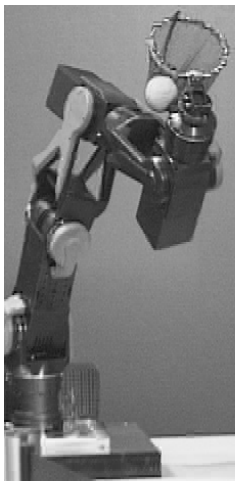
Figure 1: New DLR lightweight robot arm with the net (16cm diameter) used for catching the ball (7cm diameter). Technical data: 7-DOF, 18kg weight, 7kg payload, 1m radial workspace, 80^0/s max. joint speed due to power supply restrictions with the prototype used.

1 Introduction

The goal of this paper is to show that 3D visual tracking of a flying object is possible without making strong assumptions on the environment and using only off-the-shelf components. As a reference application we use the new DLR light weight robot arm (figure 1) [3] to catch a ball that is thrown by someone towards the robot. The problem has been addressed by several researchers and successful approaches have been documented which mostly differ from ours in that they utilize specialized hardware [1].

The most important problem that had to be addressed is that of selecting the feature to use for segmenting the ball. Using color would have been unstable or at least would have required a large contrast to the background. This is, because different sides of the ball are exposed to different lighting conditions, that may also change during the flight, so even a single-colored object would appear in different colors or shades. Another approach is to detect the ball by its shape, but this means performing some general object recognition, that is difficult even without real time constraints. Instead, we made use of the motion of the object, computing the difference between the actual image and some reference image and detecting its shape in the thresholded result. This approach proved to be very robust, as it only required the background to be static and the object to appear different from the background without assuming any specific appearance.

In our general setting, the object is in the air for about 0.8s to 1s, covering a distance of about 5m. This is just long enough for the robot to reach an arbitrary point of its workspace. Therefore it is essential to provide an early prediction of the trajectory to quickly command the catch position to the robot. As the ball is tracked for a longer time, the prediction becomes increasingly precise and the robots destination is modified.

2 General Setup

Predicting the ball with only a single camera is an extremely ill-conditioned problem. We therefore use a stereo camera system with a large baseline of 1m (figure 2) with the cameras mounted on a vertical pole.

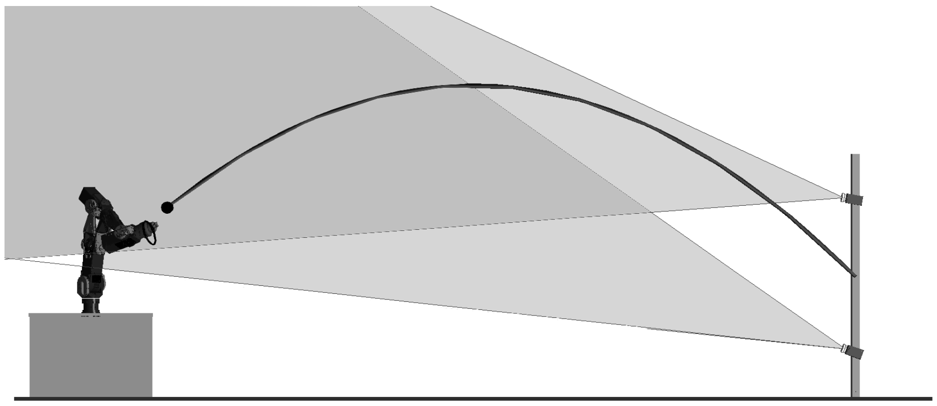

Figure 2: General Setup: The cameras are mounted on a vertical pole beside the thrower to increase the precision in the early phase of the trajectory, utilizing a long baseline (1m) for more precise triangulation. The cameras are rotated by 90^0 yielding a larger vertical field of view.

This configuration allows triangulation of the 3D position of the ball with a precision of $\approx 3cm$. Naturally the position error, especially in the radial direction, is much smaller when the ball is near the cameras.

The ball is observed at time t_0 yielding position $\vec{p_0}$ and at time t_1 yielding position $\vec{p_1}$. We now want to predict the position of the ball $\vec{p_2}$ at time t_2. This can be done by the following equation, that also yields the standard deviation of the prediction σ_2, if both positions have a standard deviation σ_{01}:

$$\vec{\gamma}_{01} = \frac{1}{2}\vec{g}\,(t_1 - t_0)^2\,, \vec{\gamma}_{12} = \frac{1}{2}\vec{g}\,(t_2 - t_1)^2\,,$$

$$\vec{p_2} = \vec{p_1} + (\vec{p_1} - \vec{p_0} - \vec{\gamma}_{01})\,\frac{t_2 - t_1}{t_1 - t_0} + \vec{\gamma}_{12},$$

$$\sigma_2 \approx \sqrt{2\frac{t_2 - t_0}{t_1 - t_0}}\sigma_{01}.$$

From the above equation it is clear that an early prediction is difficult due to the small denominator $t_1 - t_0$. To compensate therefore we place the stereo camera system slightly behind the thrower (instead of, for instance, at the robot), getting much more precise measurements of the position of the ball in the first phase.

We use standard PAL video cameras and therefore have to handle interlaced images (the same holds for NTSC cameras): Instead of 25 complete images per second one gets 50 so called fields, that consist alternating of all odd and even lines. In the whole system each field is treated as an independent image, effectively cutting the y-resolution by 2 and ignoring the effect that odd and even fields are shifted by 0.5 pixel with respect to each other. To avoid, that the reduced y-resolution further increases the position error in the radial direction, the cameras are rotated by 90°, so that the x direction, now pointing downwards, is used for triangulation. Moreover this configuration leads to a larger vertical field of view due to the cameras' aspect ratio of 4:3.

The cameras are synchronized to each other and are connected to an off-the-shelf PC (PII/300Mhz) via two frame-grabber cards (BT848 based TV card). The PC runs the computer vision algorithm to extract the ball in both images, an Extended Kalman Filter (EKF) to track its state, an predictor for the trajectory and an algorithm to select an appropriate position for catching. The catch point is transmitted to the robot control software (inverse kinematics, interpolation, joint control) via Ethernet (see figure 3). This procedure is executed for each video field (50Hz) supplying the robot with a stream of increasingly precise catch positions. The latency of the whole system is about 75 ms (details in table 1).

exposure time	5ms
image transfer	20ms
frame-grabber driver	20ms
vision algorithm	<14ms
prediction	2ms
robot control	10ms

Table 1: Latencies in the system. The delay in the frame-grabber driver is due to its inability to capture fields but only complete images.

3 Object Segmentation based on Difference Images

The main idea of our approach is to compare the actual image with some reference image. The resulting difference image is binarized using a threshold and decomposed into connected regions (blobs). Then the object shape is fit to each region taking the best fit as the object. To process the high data rate of 50 fields/s from two cameras one must carefully choose the algorithm to be fast and to allow implementation using the MMX instruction set [2].

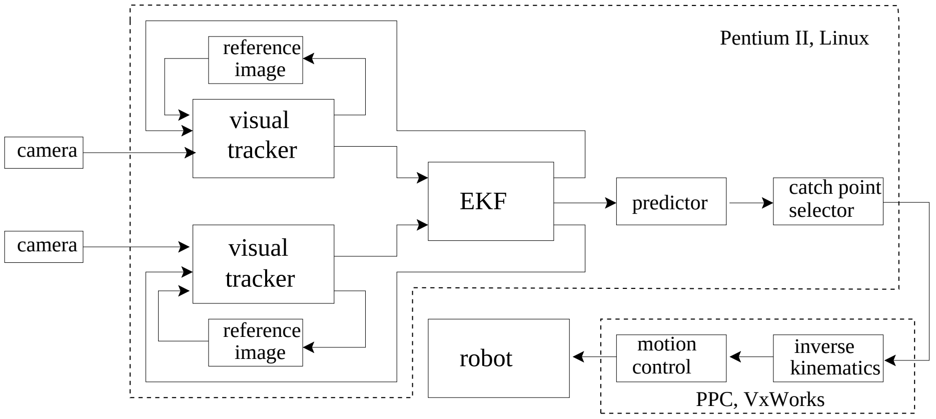

Figure 3: Data-flow diagram. The visual tracker passes image positions to the EKF, that estimates the state of the ball $(\vec{x}, \vec{v})$ and provides a ROI for the tracker. The predictor estimates the trajectory used by the catch point selector. The inverse kinematics generates joint angle profiles that are executed by the motion controller.

3.1 Difference Image

Comparing consecutive images will yield two maybe overlapping copies of the object in the difference. To avoid this problem we compare each image to a reference image instead. Simply fixing one image as reference however will not work, since there proves to be slow changes even in an otherwise static image. These often occur due to changes of lighting condition, automatic gain control in the camera or frame-grabber or due to flickering neon lamps.

Other problems arise through fast changes that occur at image edges due to pixel jitter, small camera vibrations or interlacing. Therefore we compare the pixel not to a fixed reference intensity but to a reference interval `[low..high]` of intensities "normally" occurring at this pixel. Pixels having a value that is within some tolerance between `low` and `high` are classified as "background" ('0'), whereas lower respectively higher values are classified as "object" ('1') in the binary difference image output. To cope with slow image changes the reference intervals adapt to the presented image. The following pseudocode shows the adaption as well as the difference image computation applied to each pixel:

```
difference(pix, low, high, diff, delta) {
  low++; high−−; diff = 0;
  if (pixel < low-delta) {
     diff = 1; low -= delta;
  } else if (pixel < low) low = pixel;
  if (pixel > high+delta) {
     diff = 1; high += delta;
  } else if (pixel > high) high = pixel;
}
```

`pix` is the value of the actual pixel, `[low..high]` is the reference interval, the binary result is returned in `diff` and `delta` is a parameter specifying the threshold for binarization and the maximal adaption rate (`delta=8` with values scaled to [0..255] for our experiments). Figure 4 shows the algorithms behaviour on

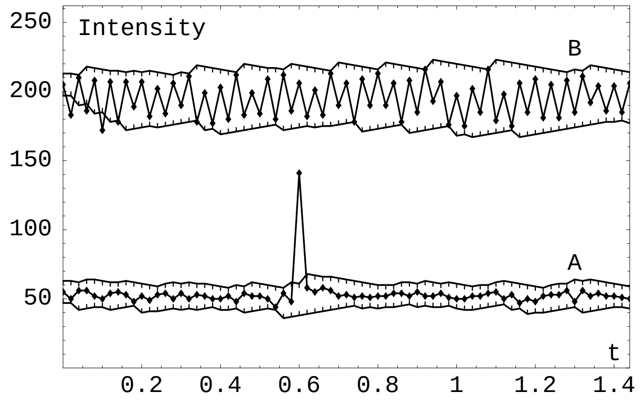

Figure 4: Adaptive difference image computation on two exemplary pixels (see pixels A and B in figure 8). The plot shows the pixels intensity and reference intervals (`[low-delta..high+delta]`, including tolerance) as they adapt over time. A is a typical pixel that is affected by some noise. It is passed by the ball at $t \approx 0.6s$. (the peak). B lies on a high contrast edge of a reflection and undergoes systematic changes due to interlacing and illumination effects. It can be observed, that the reference interval adapts to the high variation of the pixel in the first 0.2s.

two typical pixels of the image sequence shown in figure 8.

The reason for choosing an adaption scheme based on `if(...)` tests rather than for instance a lowpass filter is, that it can be much more efficiently implemented on the MMX instruction set. Another advantage for MMX implementation is, that the algorithm acts independently on all pixels.

The execution time is dominated by the CPUs memory bandwidth, so we store the result in packed binary format. To save further computation time, we apply the difference image computation only to a region of interest (ROI) (2% to 40% of the image) and process the reference images in an interleaved pattern updating only every fifth line of each image.

3.2 Segmentation

The next step is converting the binarized difference image into a set of horizontal stripes being tolerant to some incorrect pixels. We use a simple heuristic algorithm applying a counting scheme on each line.

The intention is to get stripes of a minimal length containing gaps (pixels classified '0') of at most a certain maximal length (in our algorithm both are `length`=5). This is done by sweeping through the line with a counter (`ctr`) that is incremented for each '1' pixel and decremented for each '0' pixel. The counter is clipped to [0..`length`]. If the counter exceeds `length` a new stripe is found that starts at the last pixel where the counter changed from 0 to 1, ranging beyond the actual pixel. Every time the counter exceeds `length` again the stripe gets longer. If the counter falls below 0, the stripe (ranging up to the pixel, where the counter has fallen below `length`) is finished and stored. The following pseudocode implements this algorithm. The actual stripe is represented as [`bs`..`es`] and after being finished it is stored with `storeStripe(bs, es)`.

```
for (es=bs=ctr=x=0; x<width; x++) {
  if (diff[x]==0) {
    if (ctr > 0) ctr--;
    else {
      if (es>bs) storeStripe(bs, es);
      bs = es = x+1;
  } } else {
    if (ctr < length) ctr++;
    else es = x;
} }
```

Now we take a closer look at the loop. Our intention is to build a look up table, that can process 8 pixels (one byte) at once. This is possible since the behavior of the loop body depends only on the value of `ctr` and the value of the difference image pixel `diff[x]`. The variables `bs` and `es` determine, whether a stripe is stored and contain the coordinates of the stripe but do not influence the algorithm otherwise. Exploiting this independence allows to generate a look-up-table `LUT` taking `ctr` and 8 consecutive binary pixels as input and producing the values of `ctr`, `bs` and `es` after execution of the inner loop on those 8 pixels as output. Here follows an optimized version of the algorithm utilizing `LUT`:

```
for (es=bs=ctr=x=0; x<width; x+=8) {
  (nes,nbs,ctr)=LUT[ctr,diff[x..x+7]];
  if (nbs!=unchanged) {
    if (nes+x>bs) storeStripe(bs,nes+x);
    else if (es>bs) storeStripe(bs,es);
    bs=nbs+x;
  } if (nes!=unchanged) es=nes+x;
}
```

Changes to `es` and `bs` are separated by `length` pixels. This implies, that `length` must be at least 5 to assert, that there is only one call to `storeStripe` in the 8 pixel block processed by a single LUT access. Whether there had been a call, and with which parameter can be seen from `bs`, `es`, `nbs` and `nes`.

3.3 Identification

The stripes computed by the segmentation algorithm are grouped into connected regions. We reject some regions that have implausible area or aspect ratio. Then we fit the remaining regions with the shape of the object to be tracked. In the special case of tracking a ball this reduces to fitting a 2:1 ellipse (due to interlacing). The best fitting region is decided to be the object and it's center of gravity is passed to the EKF.

4 Prediction and Interception

4.1 Dynamic Model

Modeling the trajectory of a ballistic throw is simple Newtonian mechanics. The only difficulty is the need for a rather precise model that takes into account the air drag, because the catch point has to be predicted from the first camera images for a point of time 1s in the future. The Reynolds number for an object with size of a tennis or soft ball moving in air is about $2 \cdot 10^4$ (at normal conditions), so the air drag generates a force proportional to the square of the velocity [4]. The equations of motion read then

$$\begin{aligned} \dot{\vec{v}} &= \vec{g} - \alpha\,|\vec{v}|\,\vec{v} \qquad (1)\\ \dot{\vec{x}} &= \vec{v}, \end{aligned}$$

with $\alpha = \frac{c_w A \rho}{2m}$, $c_w = 0.45$ for spheres, density of air $\rho = 1.293\frac{\mathrm{kg}}{\mathrm{m}^3}$ and $\vec{g} = -9.81\frac{\mathrm{m}}{\mathrm{s}^2}\hat{e}_z$. For a typical throw with initial velocity $v_0 = 7\frac{\mathrm{m}}{\mathrm{s}}$ and a release angle of 45° table 2 shows that especially for the soft ball the effect of the air drag can not be neglected.

	m [g]	A[cm^2]	α[cm]	x_{end}[m]
no drag	–	–	0	5.0
tennis	50	20	1.14	4.8
soft	12	36	8.75	4.0

Table 2: Effect of air drag.

The dynamical state of the ball is completely determined by its actual position and velocity $(\vec{x}_{\mathrm{act}}, \vec{v}_{\mathrm{act}})$. Given an estimate of $(\vec{x}_{\mathrm{act}}, \vec{v}_{\mathrm{act}})$ the trajectory can be predicted by integrating (1). The integration has to be done numerically (we used simple Euler integration), because (1) has no closed form solution.

4.2 State Tracking

An EKF is used for tracking the state of the ball. The system equations of the filter are given by (1). Measurements consist of the 2D image positions $\vec{b}^c$, which the vision subsystem delivers at each time step and for each camera c. We use a pin hole model without distortion for the cameras yielding the following meassurement equations, that connect the image position $\vec{b}^c$ of a ball with its real position $\vec{x}$:

$$\begin{aligned} \vec{b}^c &= \frac{f^c}{\vec{x}' \cdot \hat{e}_z} \left(\vec{x}' \cdot \hat{e}_x, \vec{x}' \cdot \hat{e}_y\right), \text{with} \qquad (2) \\ \vec{x}' &= T_{(\varphi^c, \vartheta^c, \psi^c)} \left(\vec{x} - \vec{r}^c\right), \end{aligned}$$

and with the camera parameters focus f^c, position $\vec{r}^c$ and orientation described by the matrix $T_{(\varphi^c, \vartheta^c, \psi^c)}$, where $(\varphi, \vartheta, \psi)$ are Euler angles.

For the measurement noise an rms value of 4 pixel is used and no system noise is assumed.

During implementation of the EKF it was very convenient to first prototype it in Mathematica 4.0 [7] (especially the symbolical calculation of the Jacobian of (3)) and then produce a fast C version automatically by using the MathCode [8] package.

4.3 Catch Point Generation

Once the EKF reports the prediction to have a sufficiently small covariance, a suitable point for catching the ball must be chosen along the trajectory. Because the time the robot needs to reach a position grows rapidly, when it gets more and more stretched, the catch point should not lie too close to the boundary of the workspace. On the other hand a catch point too close to the base of the robot may cause it to fold up and to reach its joint limits.

To determine the catch point we use a heuristic (figure 5). The allowed workspace is defined as the intersection W of a sphere with a convex, possibly infinite polyhedron. Moreover some "forbidden region" containing the base of the robot is defined by an additional convex polyhedron P. The trajectory is linearized at the point it intersects the sphere. On this linearized trajectory the point, which is closest to the initial position and lies in the allowed region, is chosen. This point in space corresponds to a point in time t_c. Calculating position and velocity of the ball at t_c, now using the non linearized trajectory, yields the catch point c and the orientation for the catching net.

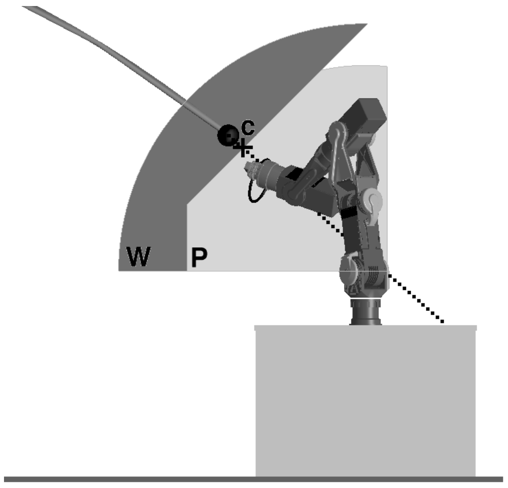

Figure 5: Catch point determination.

4.4 Motion Control

The motion algorithm (inverse kinematics and interpolation) optimizes the movement of the robot within its joint speed and acceleration limits. To reach the desired catch point fast, translational speed is given priority over orientation in trajectory generation [6], because the latter is less critical for catching the ball.

5 Calibration

For the calibration of the cameras we use a "Calibrating by Doing" procedure. Two requirements are important: First, high precision ($\approx$ 2cm) relative to the robot coordinate system is needed and second, throughout the whole space covered by the trajectories, the camera model has to be quite exact, so that already after $\approx$ 100ms a good prediction for the catch point can be made with an error of $\approx 10 - 20$cm.

For the calibration one needs samples consisting of a 3D position and the corresponding image positions. To get samples for fulfilling the first requirement the robot is moved to $\approx$ 10 positions covering the whole workspace and the image positions of the net are manually determined.

To fulfill requirement two, $\approx 2 - 3$ throws are made and the image positions of the ball while flying are recorded. To get the corresponding 3D positions the dynamical model (1) is used, which allows to calculate the trajectory in space, depending on the initial conditions $(\vec{x}_0, \vec{v}_0)$. The unknown parameters $(\vec{x}_0, \vec{v}_0)$ are treated as additional free parameters, that also have to

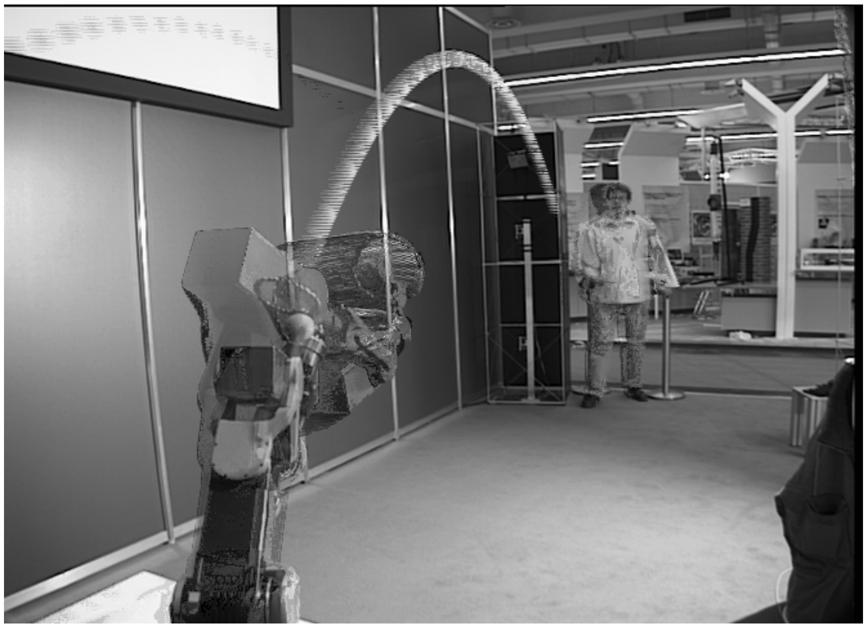

Figure 6: Example of a successful catch. The image shown is an overlay of a video sequence.

be "calibrated". This procedure is well defined since every throw gives about 100 image positions and introduces only 6 new parameters $(\vec{x}_0, \vec{v}_0)$.

The actual calibration procedure is performed using a nonlinear least square model fit. The target function Q, which has to be minimized, is defined as a sum of the quadratic deviations between the measured image positions and the image positions, which are calculated from the known corresponding 3D positions by using the measurement equations:

$$Q = \sum_c \left(\frac{1}{\sigma_{\text{rob}}^2} \sum_i \left(\vec{b}^c(\vec{x}_{\text{rob},i}) - \vec{b}^c_{\text{rob},i} \right)^2 + \frac{1}{\sigma_{\text{tra}}^2} \sum_n \sum_i \left(\vec{b}^c(\vec{x}_{\text{tra},n}(t_i)) - \vec{b}^c_{\text{tra},n}(t_i) \right)^2 \right).$$

The first term sums over the samples, that were generated by moving the robot. In the second term, n counts the recorded throws and t_i specifies the time, when an image is taken. The prediction $\vec{x}_{\text{tra},n}(t)$ of the trajectory is calculated by numerical integration of (1) and depends on the parameters $(\vec{x}_{0,n}, \vec{v}_{0,n})$. The relative weighting of both terms is controlled by σ_{rob} and σ_{tra}, which are measures for the error of the samples.

Typically about 30 parameters have to be fit: 7 for each camera (position, orientation and focus) and 6 for each ball throw. Because Q is a sum of quadratic terms the Levenberg-Marquardt method [5] for minimization can be used.

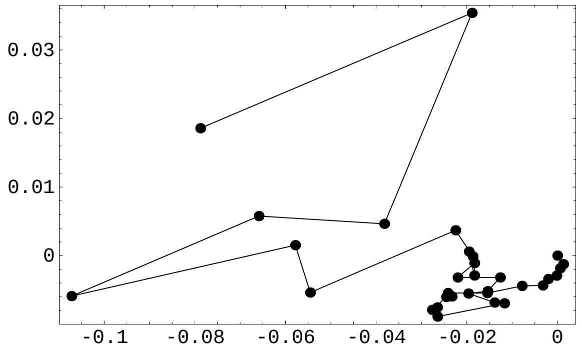

Figure 7: Predicted trajectory as a function of time. The intersection of the trajectory with a plane going through the final catch point and being orthogonal to the last trajectory is shown. This way each trajectory is represented by one point. (axis in meters; 20ms time step between the points; first point 60ms after detecting the ball)

6 Experiments

The catching system was presented at the Hannover Fair 2000 (figure 6). About 100 throws with soft balls were made by different people, including visitors. In about 2/3 of the tries the robot was successful in catching the ball. The majority of faults was due to the camera's limited horizontal field of view resulting in the system seeing the ball too late.

Figure 7 visualizes the quality of the trajectory pre-

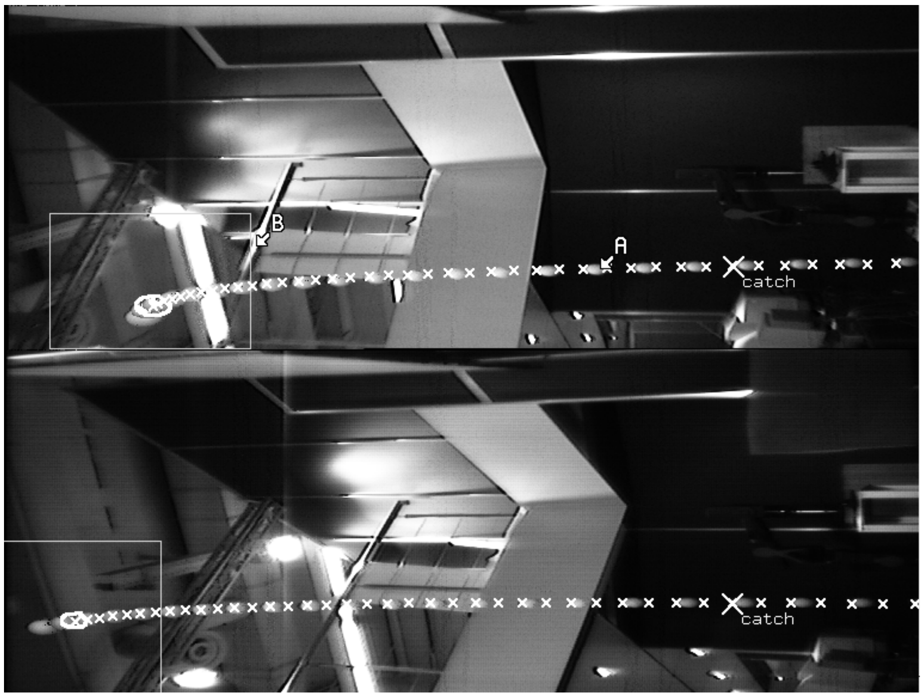

Figure 8: Overlay of all images of the top and bottom camera (rotated by 90^0). The tracker state and the predicted trajectory after observing 3 images (60ms) is shown: ROI (rectangle), extracted ball (ellipse), predicted trajectory with timesteps of one image (crosses), catch point ("catch"). Observe the difficult nonuniform background and changing lighting conditions along the trajectory. The pixels A and B are referenced in figure 4.

dictions as a function of time. One can observe the high accuracy ($<$ 10cm) achieved even for the early predictions, with the major error occurring in the direction of the throw (x-axis in the figure). Figure 8 shows the throw as seen by the cameras. Overlayed is a early prediction of the trajectory.

7 Summary

We have shown that it is possible to catch a thrown ball with a robot arm using computer vision, Kalman filtering and prediction on a PC, when one carefully designs algorithms suitable for high speed implementation. We generated reasonable predictions very early by choosing a good camera position and achieved the necessary calibration mainly by fitting a model to some observed trajectories of the ball ("Calibrating by Doing").

What comes next ? The natural idea could be to use the DLR artificial hand instead. One may even be tempted to take a racket and try to play tennis. This means being confronted with the challenge of how to generate constrained optimized motion as, for example, a forehand stroke, requiring synergy between perception, control and mechanics. In this way the old proverb: "Mens sana in corpore sano" could be an interpretation of mechatronics.

References

[1] R.L. Andersson. *A Robot Ping-Pong Player.* MIT Press, 1998.

[2] Intel Corporation. *Complete Guide to MMX Technology.* McGraw-Hill, 1999.

[3] G. Hirzinger, A. Albu-Schäffer, M. Hähnle, I. Schaefer, and N. Sporer. On a new generation of torque controlled light-weight robots. *Proc. of the Internation Conference of Robotics and Automation, Seoul, Korea*, 2001.

[4] L.D. Landau and E.M. Lifschitz. *Hydrodynamik.* Akademie-Verlag Berlin, 1976.

[5] W. H. Press, S. A. Teukolsky, W. T. Vetterling, and B. P. Flannery. *Numerical Recipes in C.* Cambridge University Press, 1992.

[6] G. Schreiber, M. Otter, and G. Hirzinger. Solving the singularity problem of non-redundant manipulator by constraint optimization. In *IEEE/RSJ Intl. Conference on Intelligent Robots and Systems 1999, Kyongju, Korea*, pages 1482–1488, 1999.

[7] S. Wolfram. *The Mathematica Book.* Cambridge University Press, 1999.

[8] http://www.mathcore.com/products/mathcode/.