



How to build a simple marker tracker

Daniel Pustka

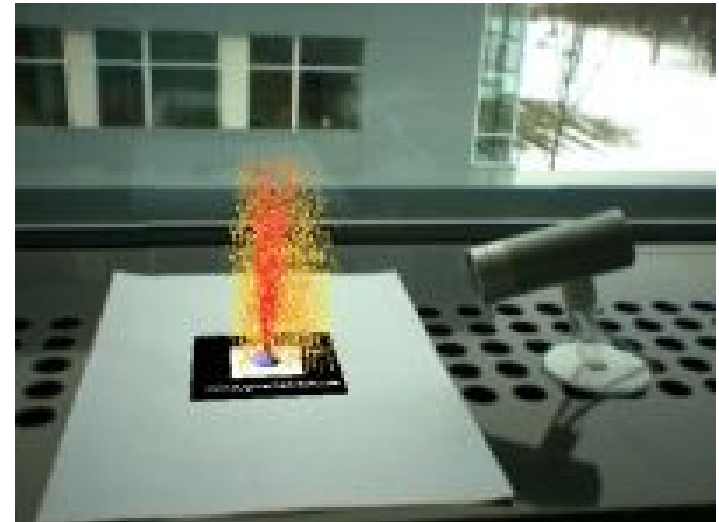
24 November 2005

Chair for Computer Aided Medical Procedures & Augmented Reality

Department of Computer Science | Technische Universität München

Marker-based Tracking for Augmented Reality

- What is Augmented Reality?
 - Display some virtual object at a fixed place in the world
 - Problem: Need to know position and orientation of objects in the image
- Why markers?
 - Easy to detect
 - No initialization problem
 - No 3D model of environment





The Tools

- Webcam
- C compiler
- OpenCV
 - Open-source C library
 - Image acquisition & display
 - Mathematics
 - Image processing
 - <http://www.intel.com/research/mrl/research/opencv/>
- Praktikum Augmented Reality SS2005
 - Tutorials, exercise sheets, code snippets, markers sheets
 - <http://campar.in.tum.de/Chair/TeachingSS05ARLabCourse>



Overview of the Tracker

- Grabbing an image
- Thresholding
- Finding connected components
- Rectangle identification
- Refining corner positions
- Marker identification
- Pose estimation
- OpenGL rendering

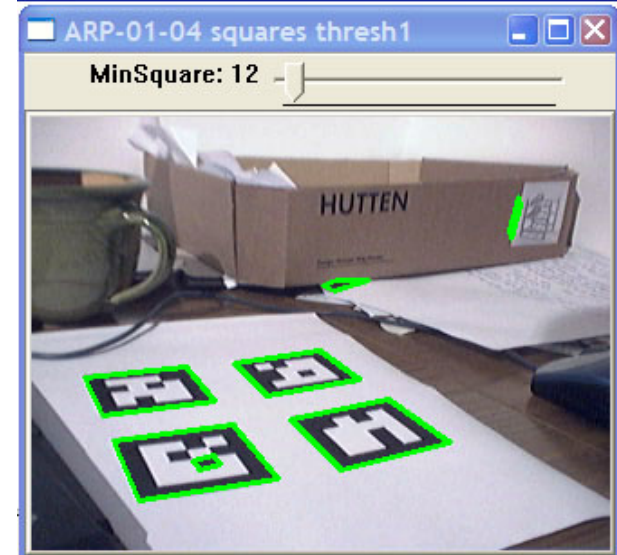
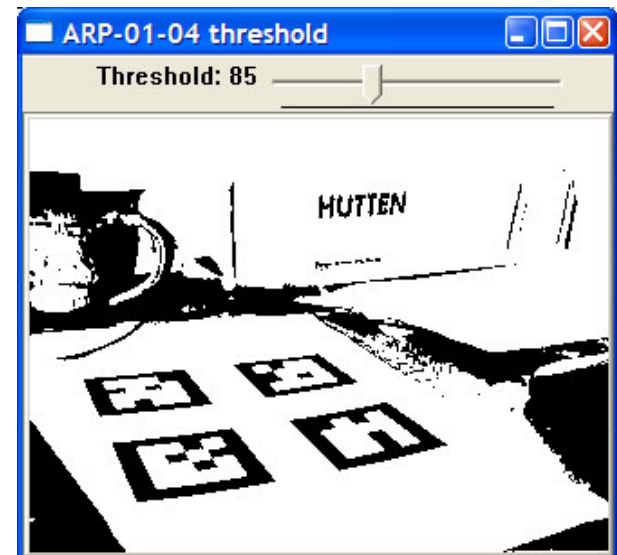
Thresholding

- Image can be seen as a matrix of pixel intensities
- Selecting one value between black and white
- `cvThreshold(...)`



Finding connected components

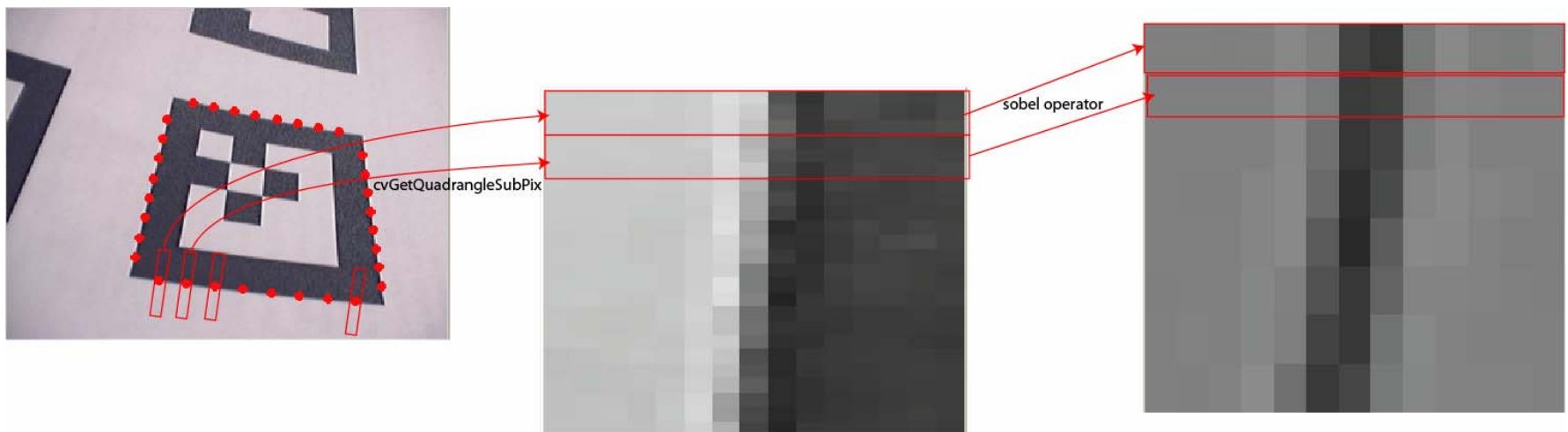
- Finding boundaries between black and white pixels
 - `cvFindContours(...)`
- Filtering too small ones (noise)
- Polygonal approximation
 - `cvApproxPoly(...)`
- Selecting only those with four corners



Extraction of stripes perpendicular to sides

- Compute eight points along each side
- Extract stripes three pixels wide
 - `cvGetQuadrangleSubPix`
- Apply Sobel operator:

$$S = \begin{bmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{bmatrix}$$



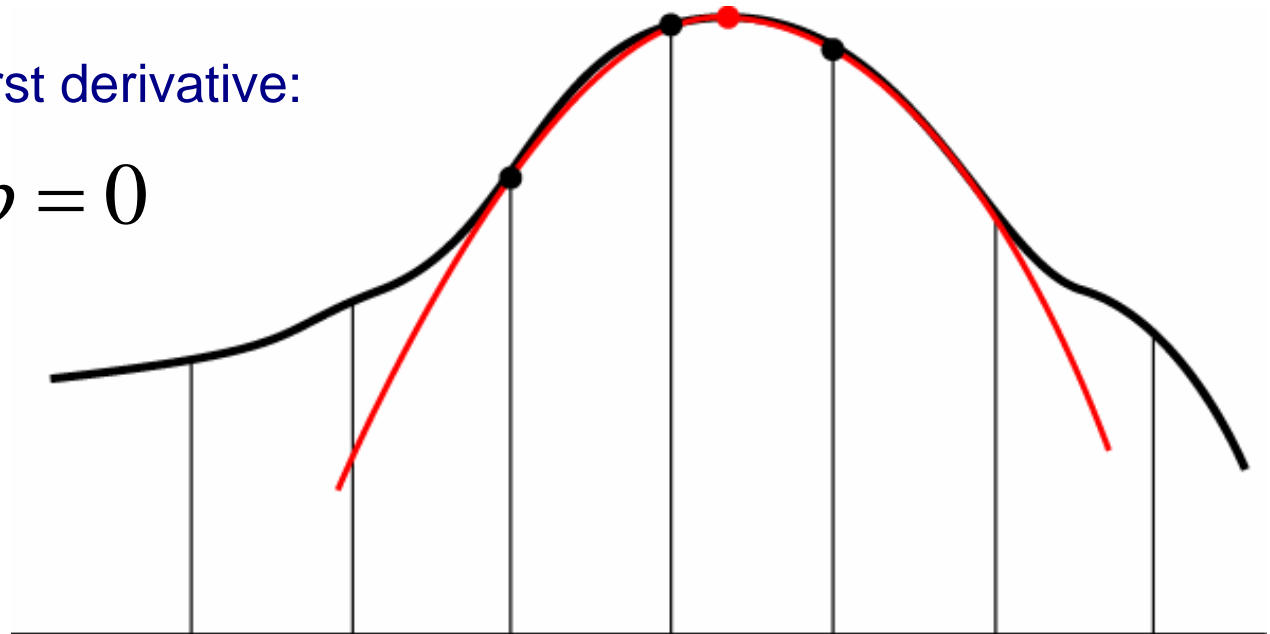
Subpixel-precise localization of edges

- Finding the point with highest intensity change in each stripe
- Fit parabola through point and two neighbors:

$$y = ax^2 + bx + c$$

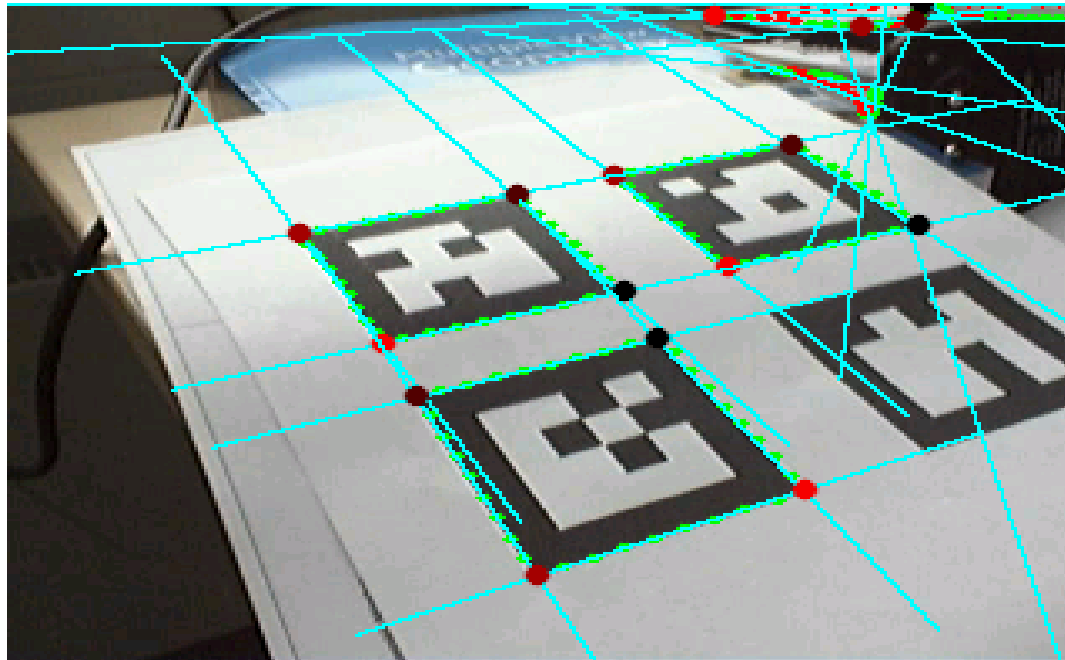
- Find zero of first derivative:

$$2ax + b = 0$$



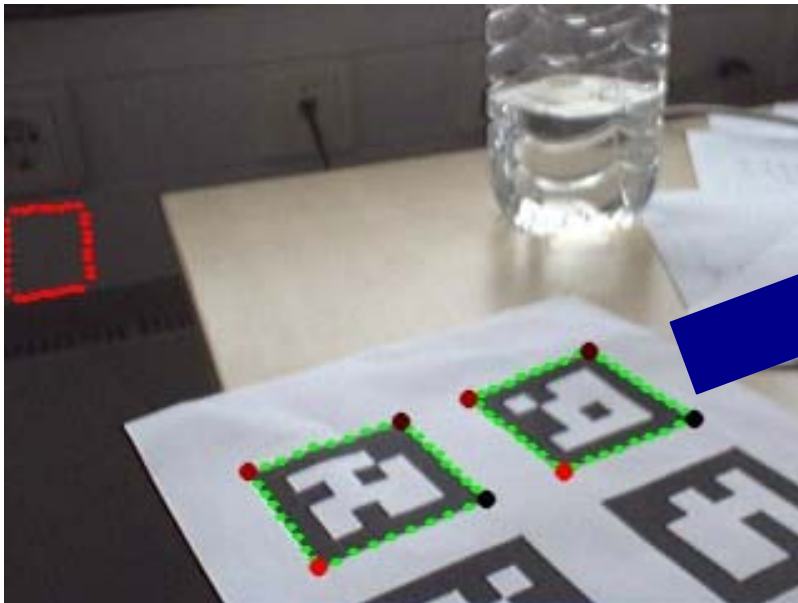
Precise corner detection

- Fit line through eight points of each side
 - → simple least-squares approach (`cvFitLine`)
- compute corners as intersections of sides



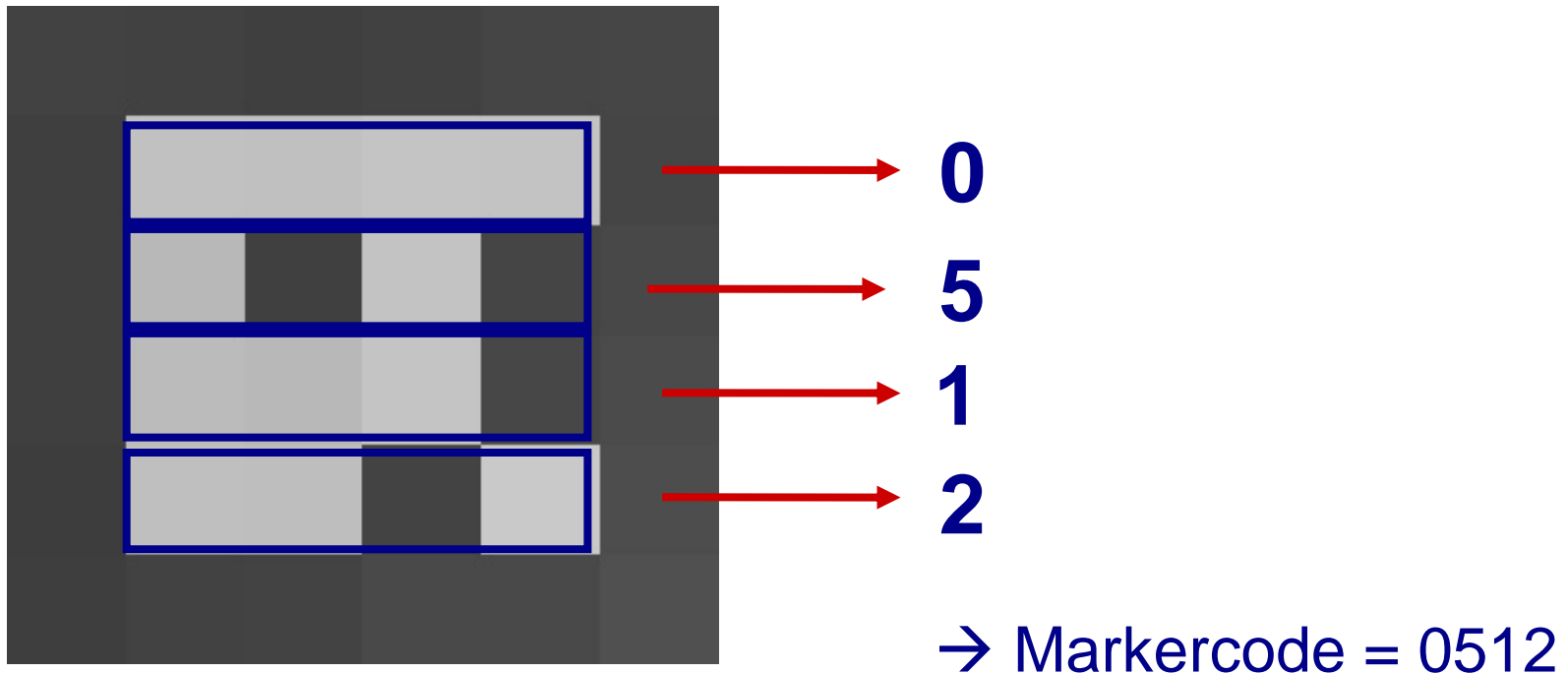
Rectification of Marker Image

- Set coordinate system on marker
- Use DLT algorithm to compute homography H
- Sample image: Apply H to 6×6 points (cvWarpPerspective)



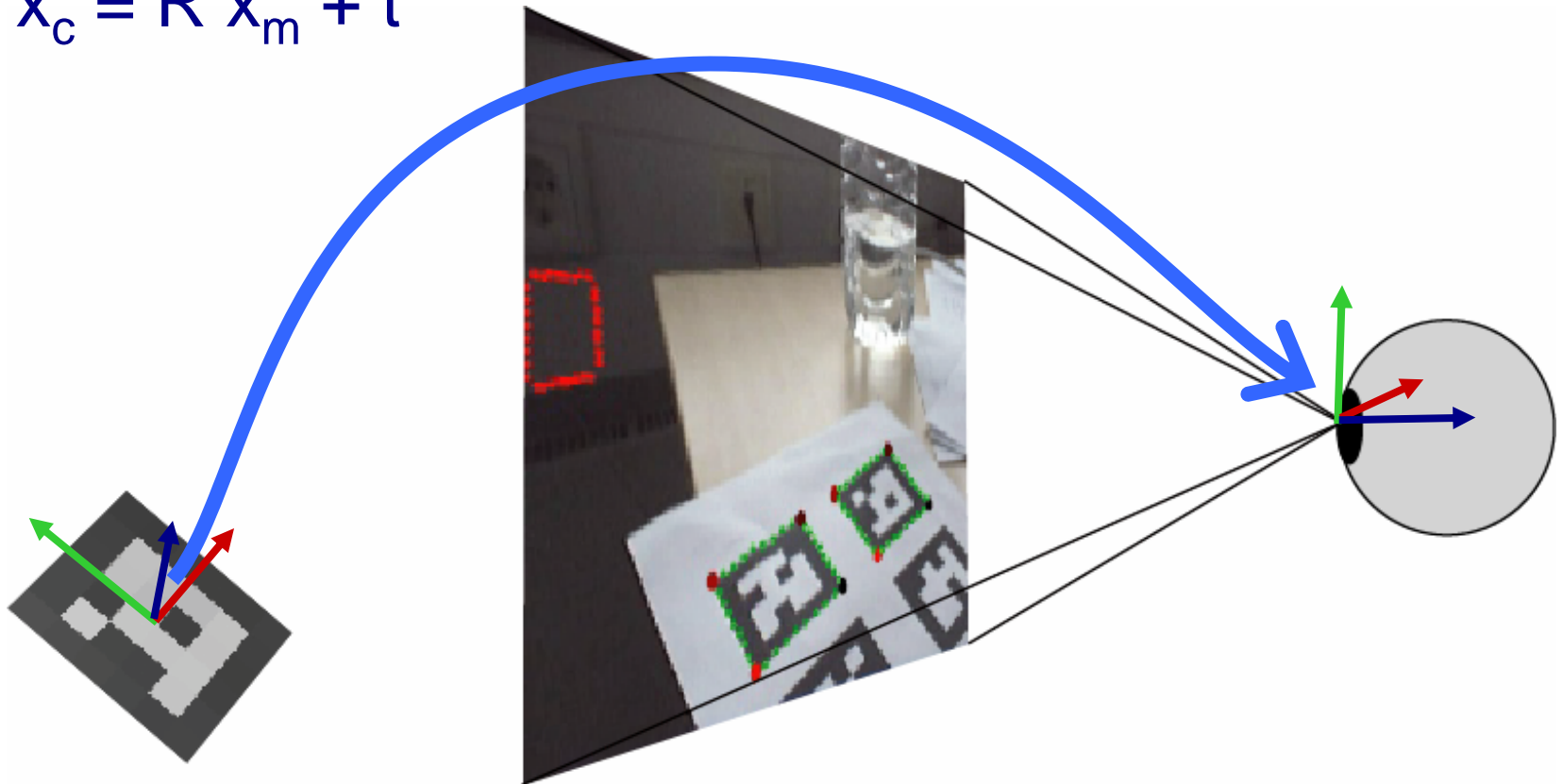
Marker Identification

- Compute 16-bit hexadecimal code from rectified marker image
- Account for symmetry!!



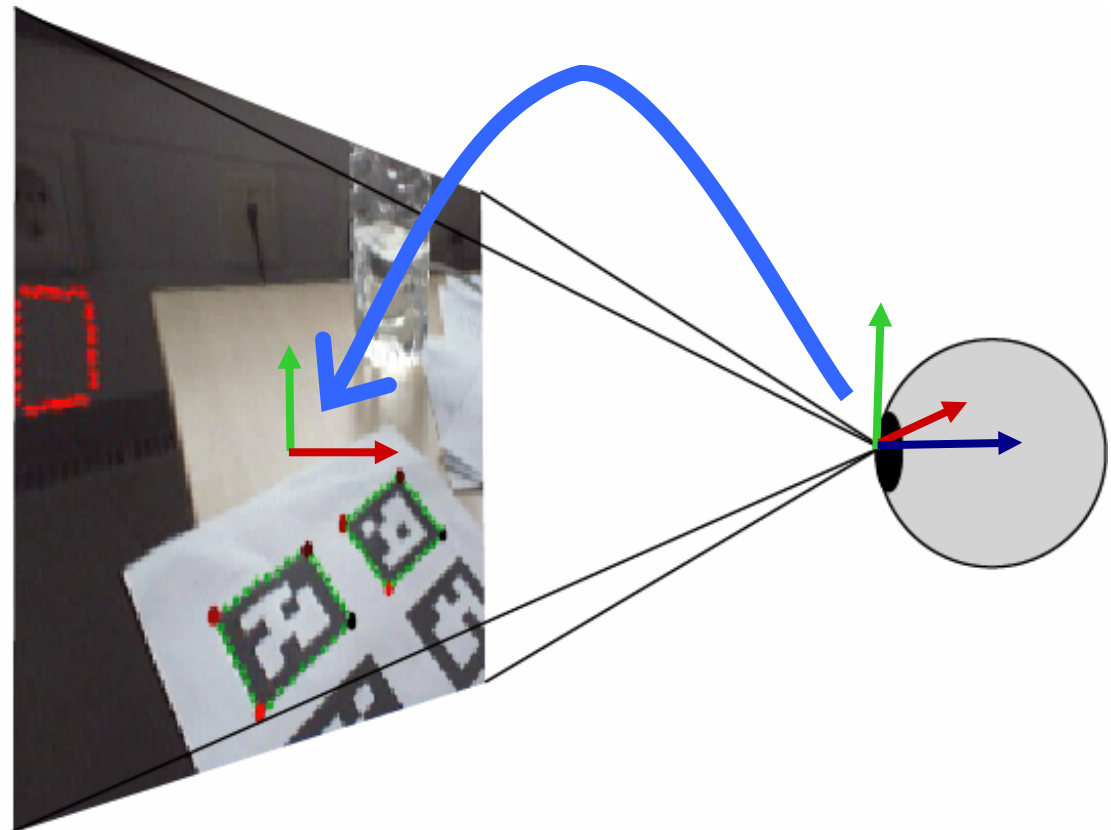
Coordinate Transformations: Marker \rightarrow Camera

$$x_c = R x_m + t$$



Coordinate Transformations: Camera \rightarrow Image

$$x_i = f * x_c / -z_c$$
$$y_i = f * y_c / -z_c$$



Homogeneous Projection Matrix

- Writing both transformations as homogeneous matrices

$$\mathbf{x}_i = \begin{bmatrix} f & 0 & 0 & 0 \\ 0 & f & 0 & 0 \\ 0 & 0 & -1 & 0 \end{bmatrix} \begin{bmatrix} \mathbf{R} & \mathbf{t} \\ \mathbf{0} & 1 \end{bmatrix} \mathbf{x}_m = \begin{bmatrix} f & 0 & 0 \\ 0 & f & 0 \\ 0 & 0 & -1 \end{bmatrix} \begin{bmatrix} \mathbf{R} & \mathbf{t} \end{bmatrix} \mathbf{x}_m$$

- Assuming marker coordinates are on marker plane

$$\mathbf{x}_m = [x \quad y \quad 0 \quad 1]^T$$

$$\mathbf{x}_i = \begin{bmatrix} f & 0 & 0 \\ 0 & f & 0 \\ 0 & 0 & -1 \end{bmatrix} \begin{bmatrix} \mathbf{r}_x & \mathbf{r}_y & \mathbf{t} \end{bmatrix}^T \mathbf{x}'_m = \mathbf{H} \mathbf{x}_m$$



Getting Rotation and Translation

- Get homography \mathbf{H} with DLT
- Compute \mathbf{r}_x , \mathbf{r}_y and \mathbf{t} from \mathbf{H}

$$\begin{bmatrix} \mathbf{r}_x & \mathbf{r}_y & \mathbf{t} \end{bmatrix} = \begin{bmatrix} 1/f & 0 & 0 \\ 0 & 1/f & 0 \\ 0 & 0 & -1 \end{bmatrix} \mathbf{H}$$

- correct scaling of translation requires normalization

$$|\mathbf{r}_x| = 1$$

- \mathbf{r}_z can be computed exploiting orthogonality of \mathbf{R}

$$\mathbf{r}_z = \mathbf{r}_x \times \mathbf{r}_y$$



Refinement Using Nonlinear Minimization

- Compute geometric distance between measured corner coordinates and their computed projection using the estimated pose

$$\mathbf{d} = \begin{bmatrix} x_i \\ y_i \end{bmatrix} - \text{inhom} \left(\begin{bmatrix} f & 0 & 0 \\ 0 & f & 0 \\ 0 & 0 & -1 \end{bmatrix} [\mathbf{R} \quad \mathbf{t}] \mathbf{x}_m \right)$$

- Note: non-linear!
- Use iterative non-linear optimization on $\mathbf{R} \quad \mathbf{t}$ which minimizes $|\mathbf{d}|$
- Problem: \mathbf{R} has too many DOF \rightarrow use quaternions for rotation

$$\mathbf{x}_c = \mathbf{q} \mathbf{x}_m \mathbf{q}^* + \mathbf{t}$$

Final Tracker

- Load projection and $[R \ t]$ into OpenGL
- Draw video image and some funny objects

