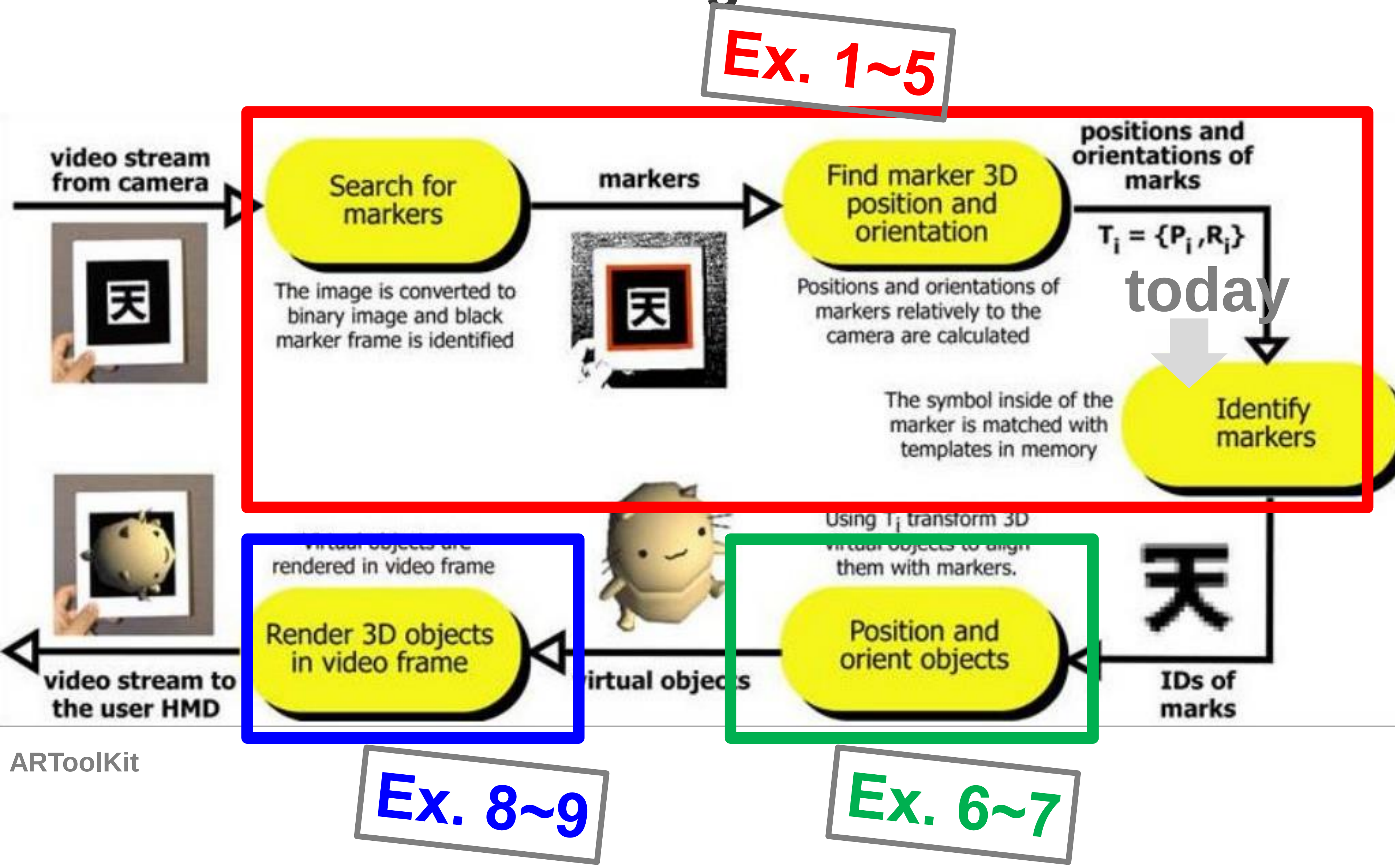
Introduction to Augmented Reality

Tutorial 4: Marker Tracking Part 4
May 9th 2018

Andreas Langbein, Adnane Jadid, David Plecher

Marker-based Tracking

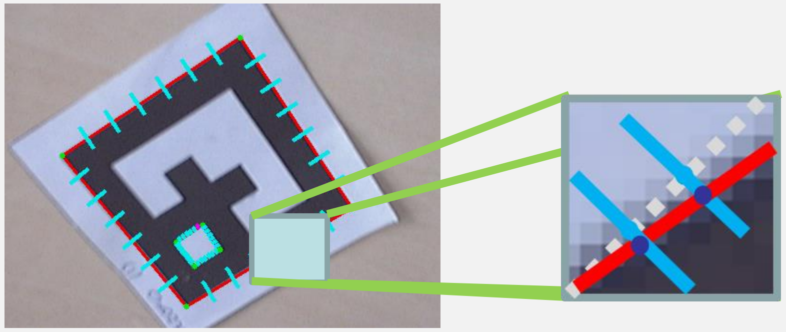

Ex. 1~5
video stream from camera
Search for markers
The image is converted to binary image and black marker frame is identified
markers
Find marker 3D position and orientation
Positions and orientations of markers relatively to the camera are calculated
positions and orientations of marks
$T_i = \{P_i, R_i\}$
today
Identify markers
The symbol inside of the marker is matched with templates in memory
IDs of marks
Using T_i transform 3D virtual objects to align them with markers.
Position and orient objects
virtual objects
Virtual objects are rendered in video frame
Render 3D objects in video frame
video stream to the user HMD
Ex. 8~9
Ex. 6~7

ARToolKit

Solution for the Previous Tutorial

Ex. 3

Find marker in 2D **precisely**

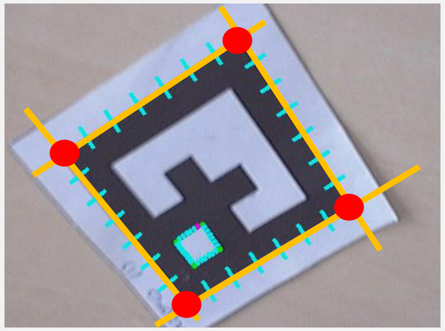


Code walkthrough

Today’s Tutorial

Ex. 4

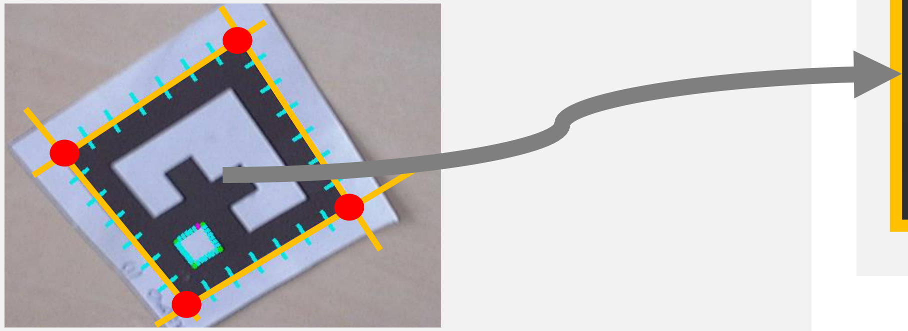
Find marker corners **precisely**



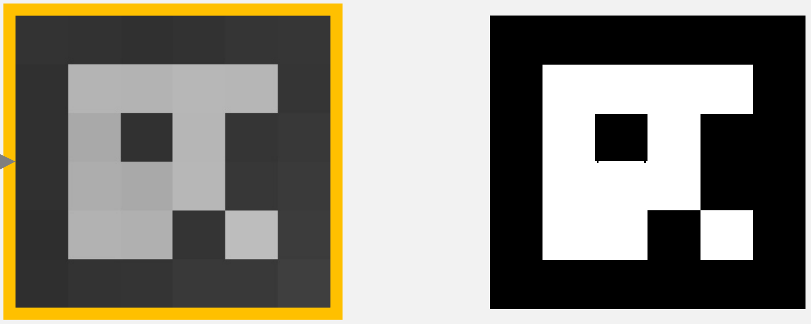
Today's Tutorial

Ex. 4

Find marker corners **precisely**



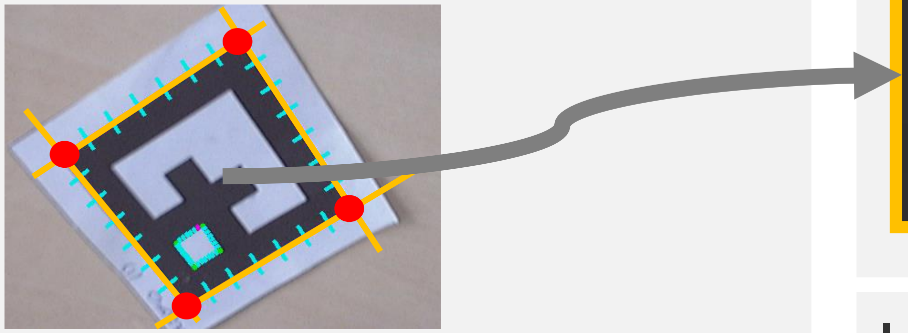
RECTIFY marker



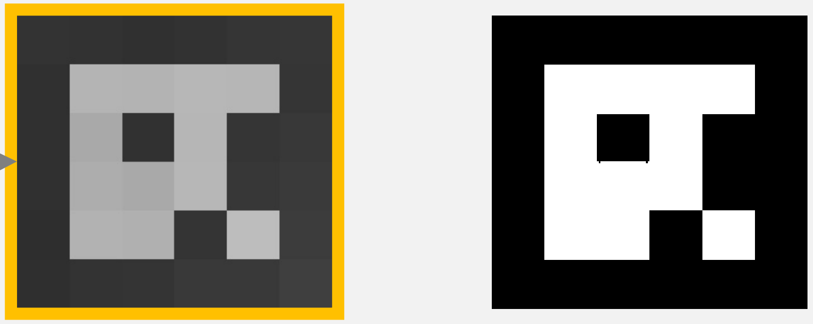
Today's Tutorial

Ex. 4

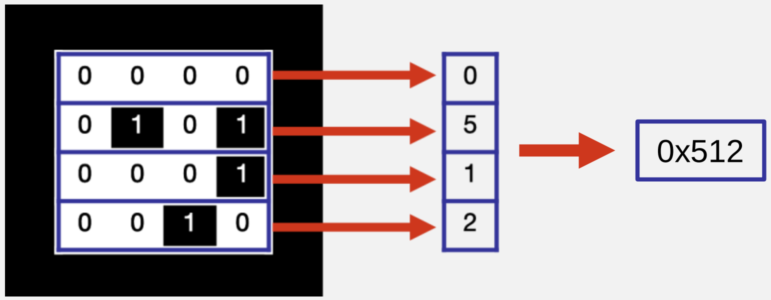
Find marker corners **precisely**



RECTIFY marker



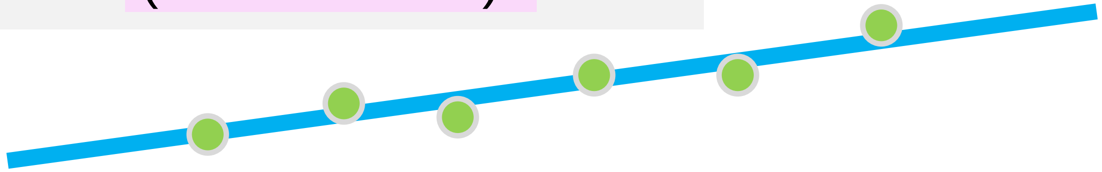
Identify marker ID



Precise Corner Detection

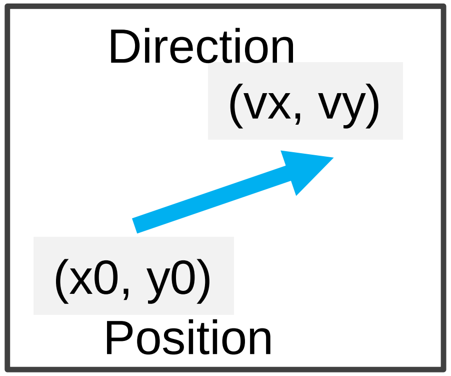
Fit line through six points of each side

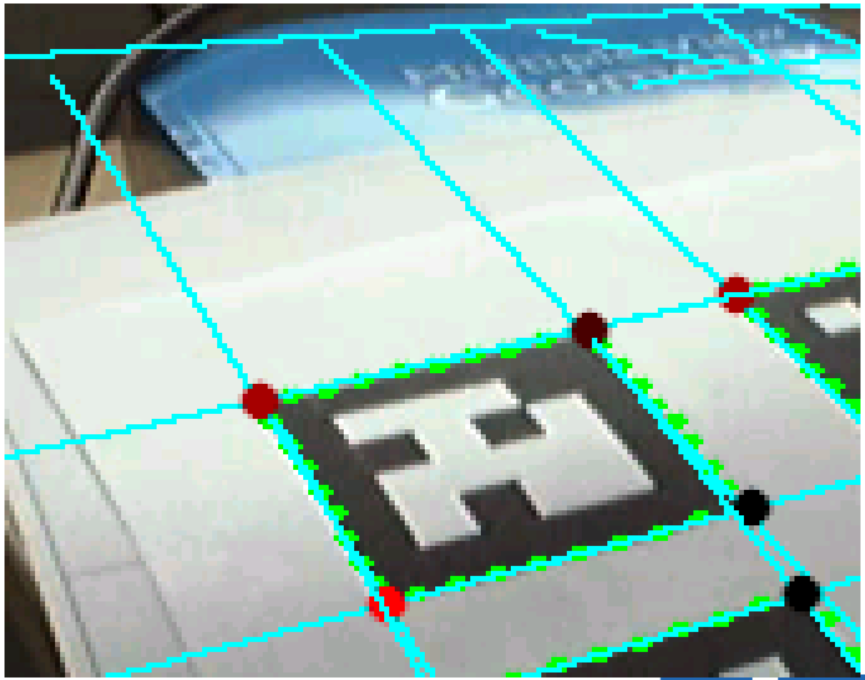
simple least-squares approach `(cv::fitLine)`



```
cv::fitLine( points, line, CV_DIST_L2, 0, 0.01, 0.01)
```

line = (vx, vy, x0, y0)

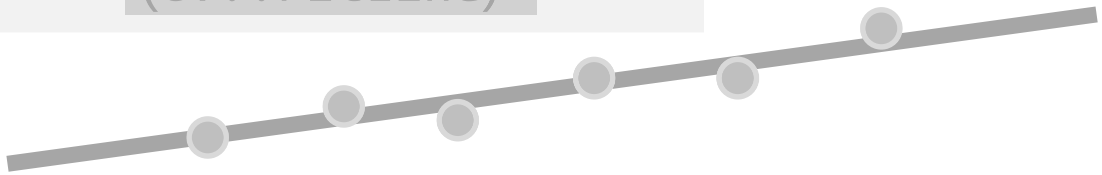




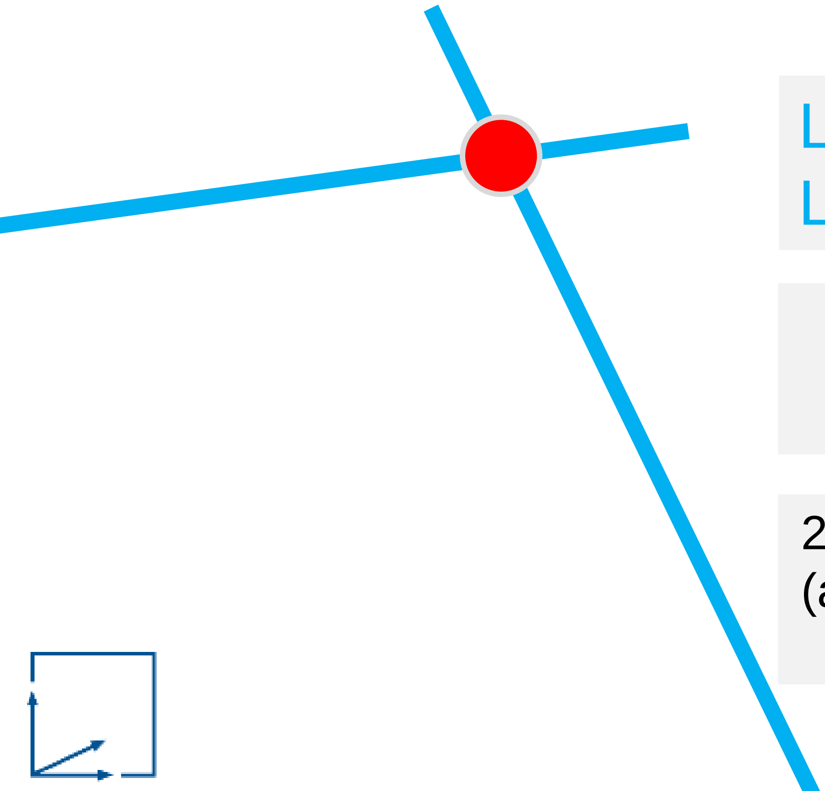
Precise Corner Detection

Fit line through six points of each side

simple least-squares approach `(cv::FitLine)`



Compute corners as intersections of sides



Line1: $2x + y = 8$

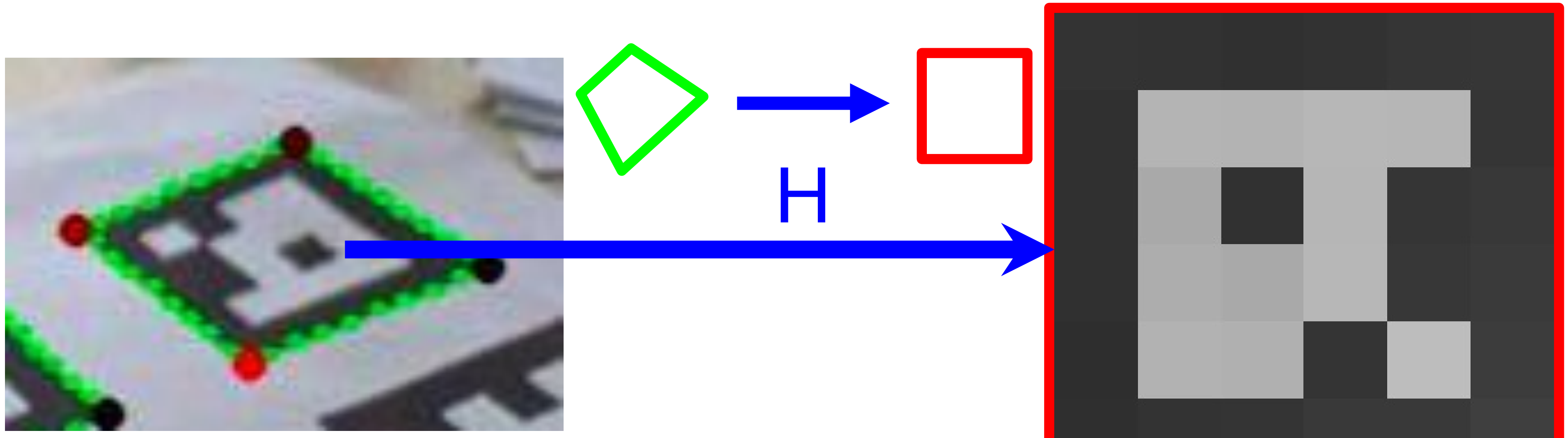
Line2: $4x + 2y = 6$

$$\begin{bmatrix} 2 & 1 \\ 4 & 2 \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix} = \begin{bmatrix} 8 \\ 6 \end{bmatrix}$$

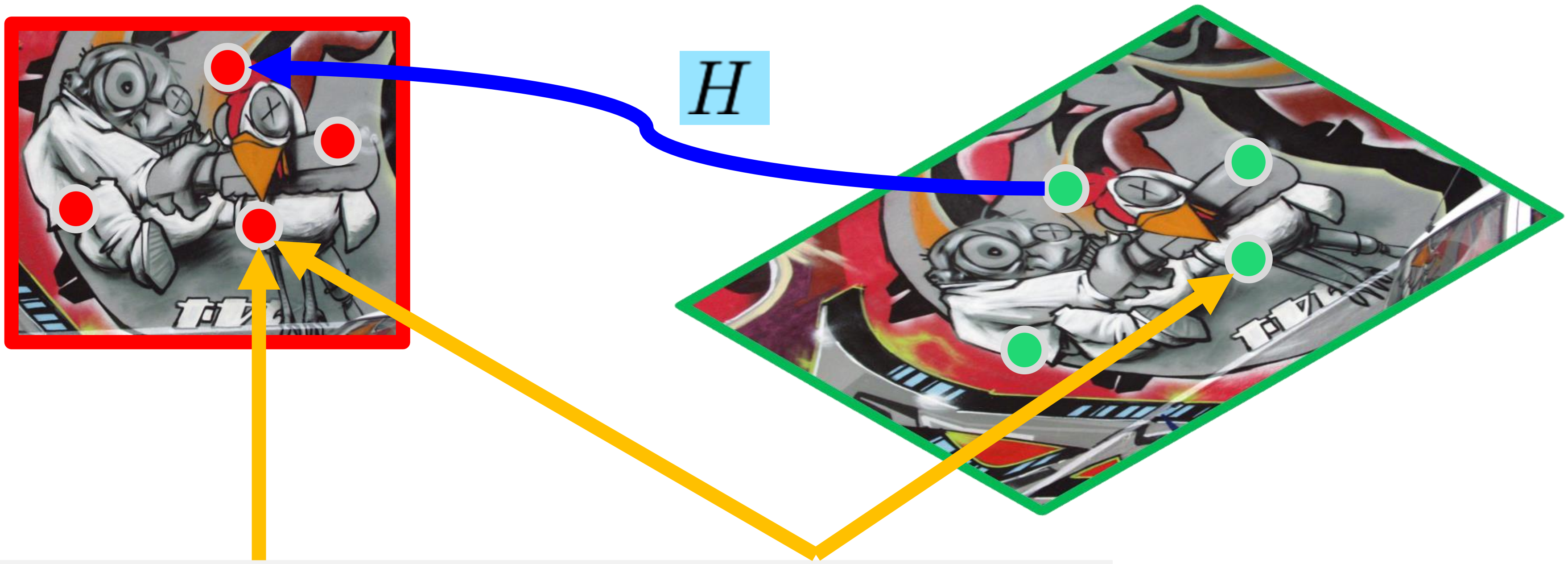
2D line intersection
(advanced:
usage of Cramer's rule and some transformations)

Marker Rectification

- Create 6*6 Pixel ID image
  - coordinates (-0.5,-0,5) to (5.5,5.5)
- Calculate Homography from corner points in original image to ID image
  - `cv::getPerspectiveTransform` or `cv::warpPerspective`



[FYI] Markerless Tracking (Natural feature)



Feature descripter & matching

Square marker detection

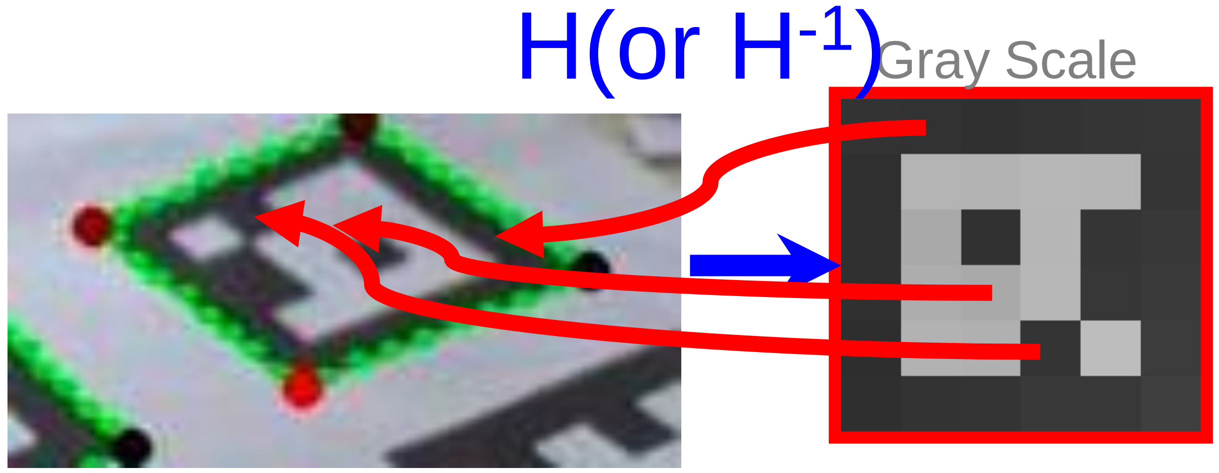
Marker Rectification

`cv::getPerspectiveTransform`
or, `cv::warpPerspective`

Calculate ID image

– multiply each pixel of ID image with Homography

$$\begin{bmatrix} x_{\text{orig}} \\ y_{\text{orig}} \\ 1 \end{bmatrix} \approx H \begin{bmatrix} x_{\text{ID}} \\ y_{\text{ID}} \\ 1 \end{bmatrix}$$

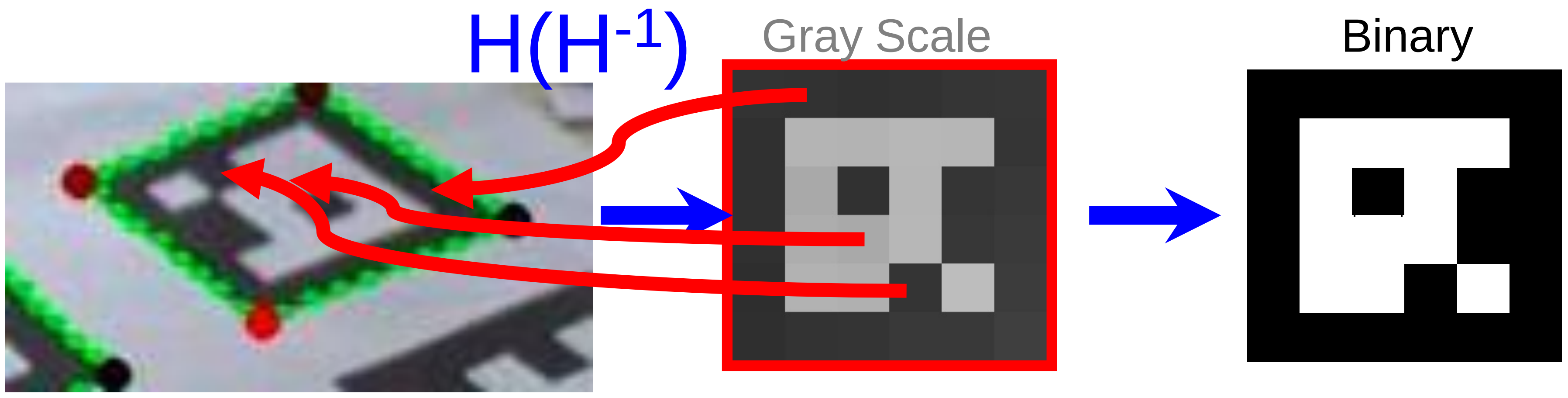


Marker Rectification

`cv::getPerspectiveTransform`
or, `cv::warpPerspective`

Calculate ID image

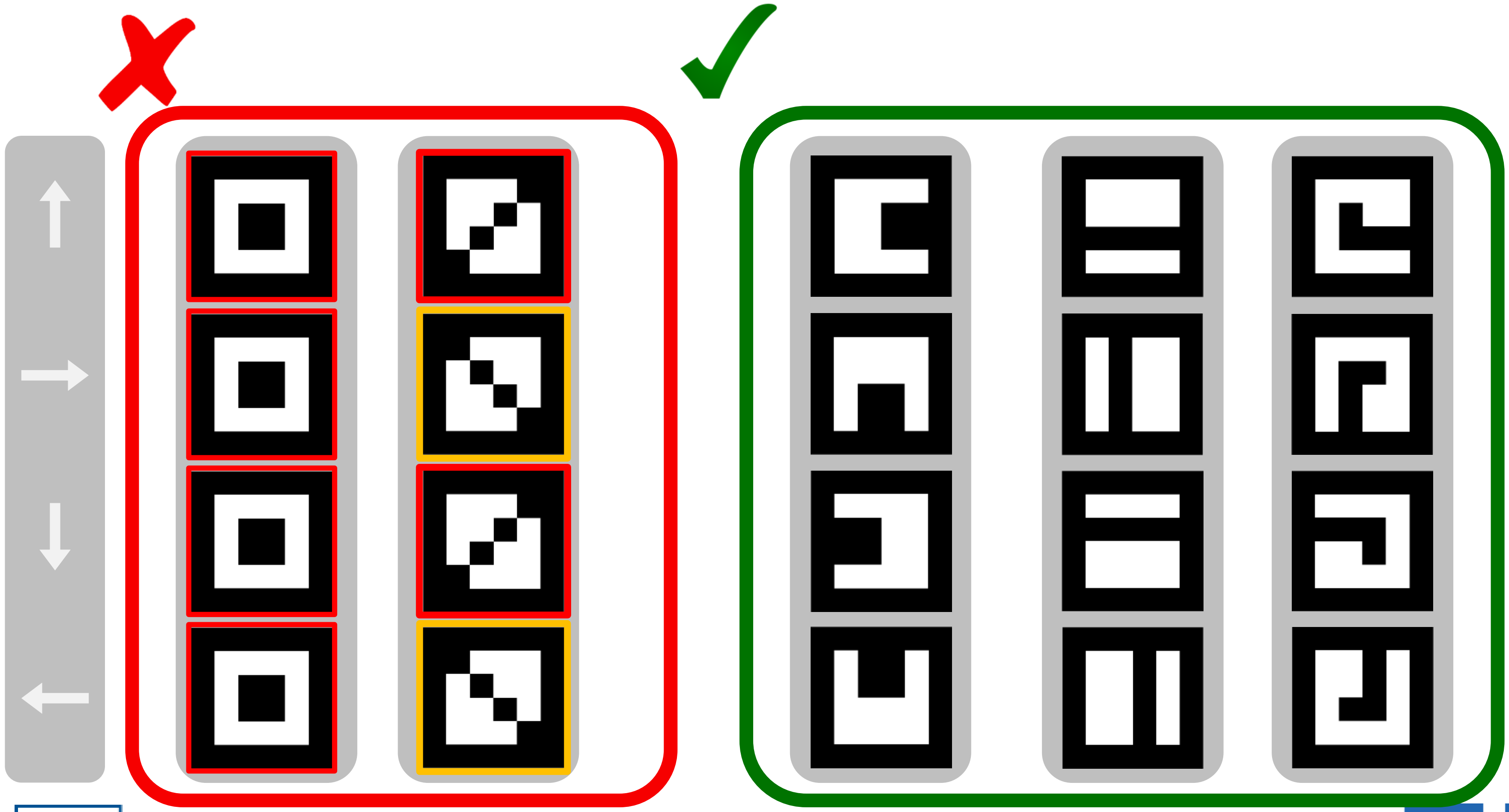
- multiply each pixel of ID image with Homography
- threshold the gray values to get a binary (B/W) image
  - (implement another trackbar to control this threshold)



Marker Identification

Account for symmetry!!

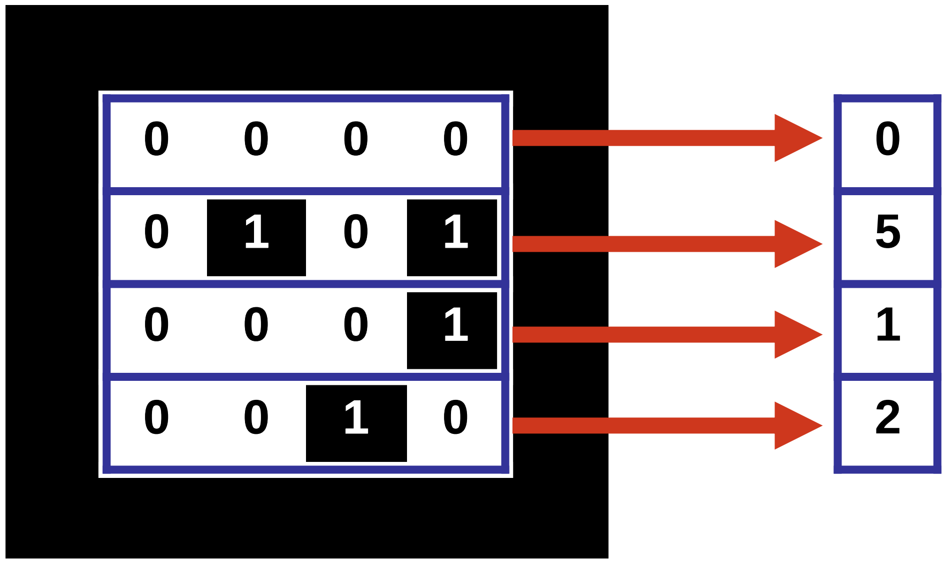
Rotation, how to make it unique.



Marker Identification

Discard markers without black border

- Calculate ID out of 4x4 inner pixels
- Codes: black = 1, white = 0

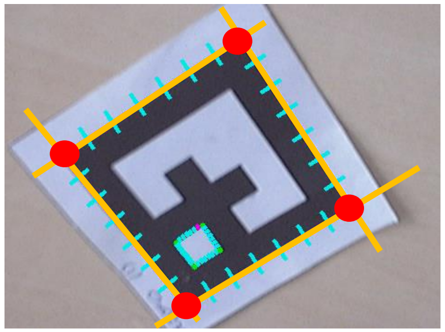


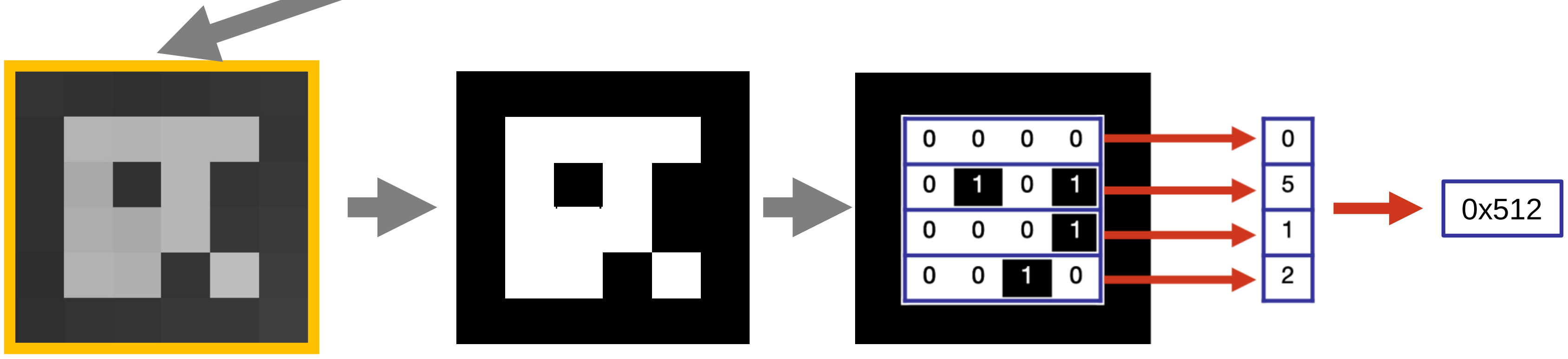
Smallest ID

Homework

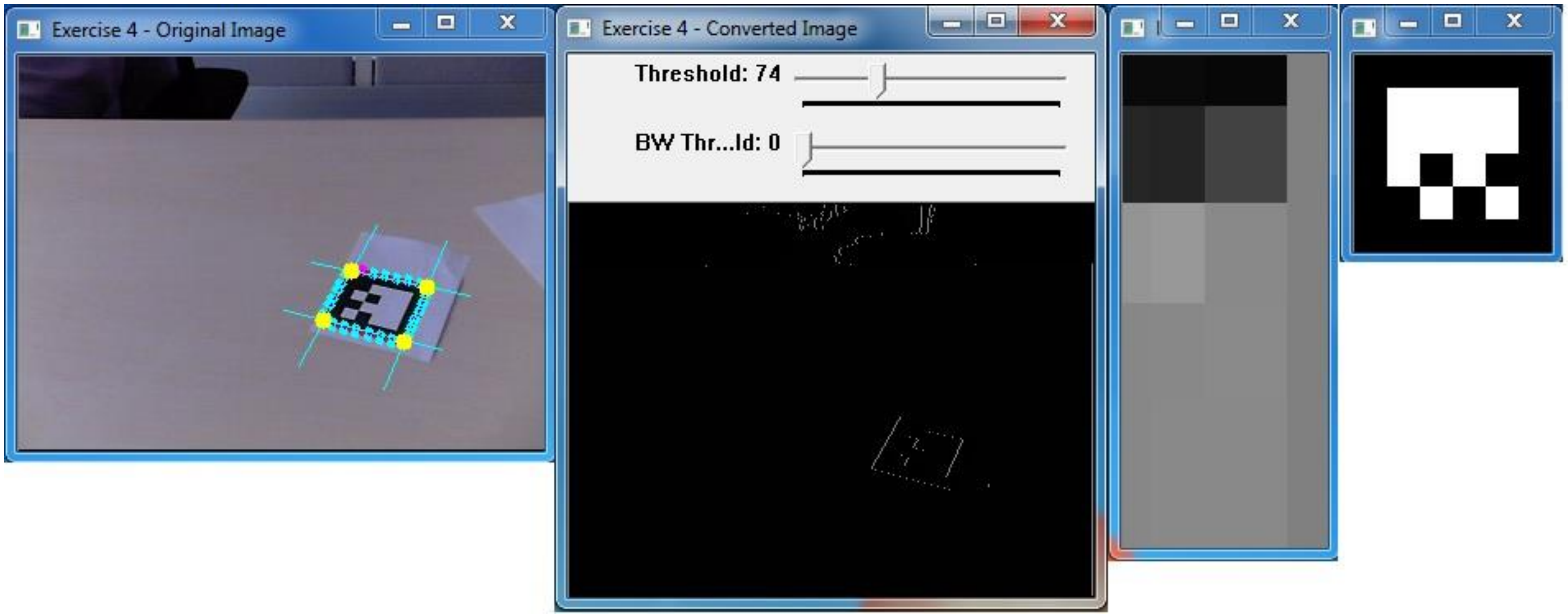
1. Corner estimation

2. Marker rectification/identification





Homework

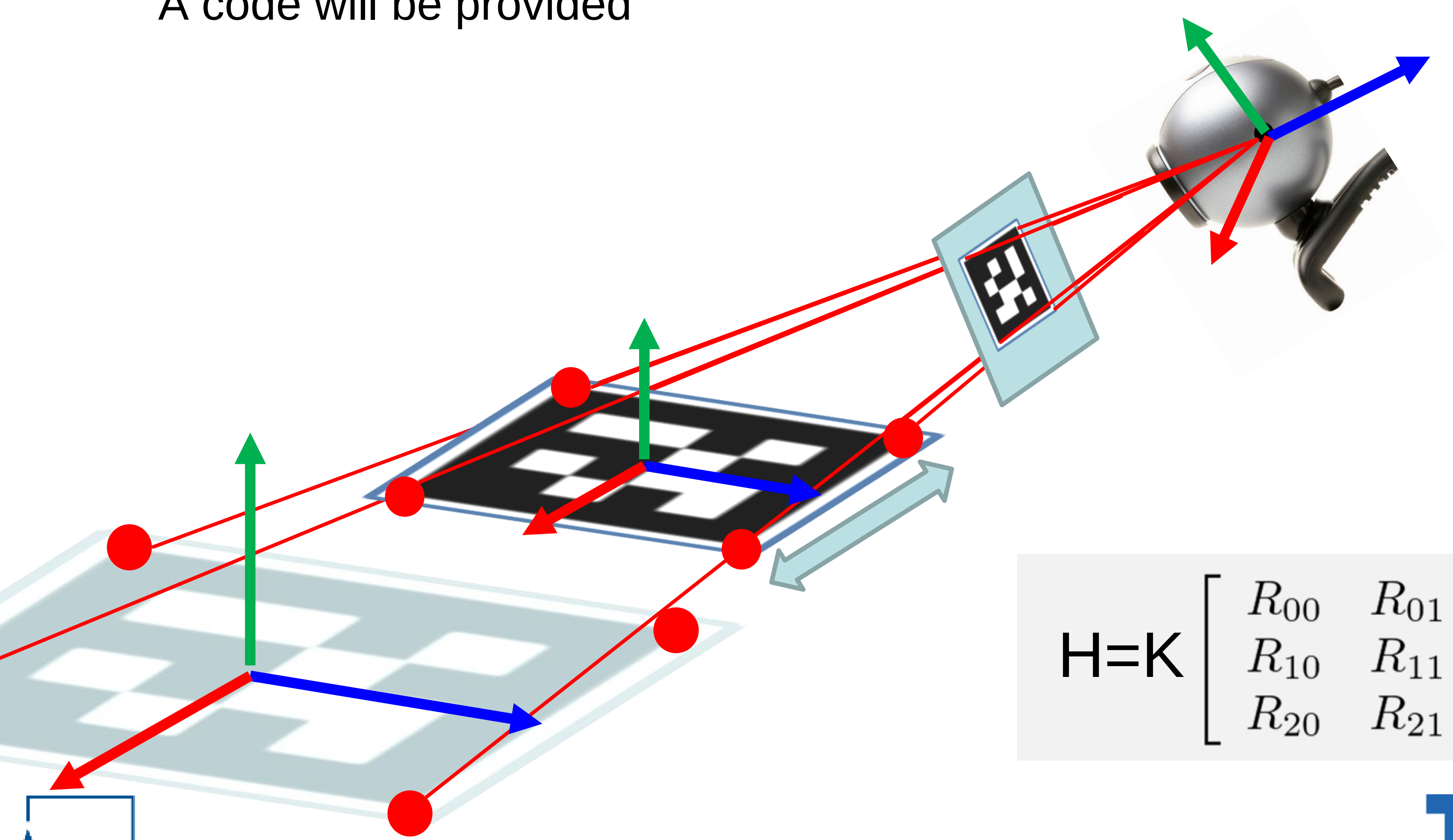

Exercise 4 - Original Image
Exercise 4 - Converted Image
Threshold: 74
BW Thr...ld: 0

Spoiler of the next tutorial

Marker-Pose Estimation

Pose = Position + Orientation

A code will be provided



$$H=K\begin{bmatrix} R_{00} & R_{01} & t_0 \\ R_{10} & R_{11} & t_1 \\ R_{20} & R_{21} & t_2 \end{bmatrix}$$

Sketch Solution for Exercise 4

```
...
    //loop over edgepoints of one edge
        //End of sobel
        //save edge points
    //end of loop over edgepoints of one edge
    ...
    cv::fitLine(...)//to get the line parameters of each edge
                 //end of loop over the 4 edges
                 //calculate exact corners using
                 //the calculated line parameters
...
//to get the matrix of perspective transform
projMat = cv::getPerspectiveTransform(...);

//create Marker Image
cv::warpPerspective(...)

//threshold the marker image to get a B/W image
cv::threshold(...)

//Identify the Marker
...
```

That‘s it…

- Questions