AR Application Working

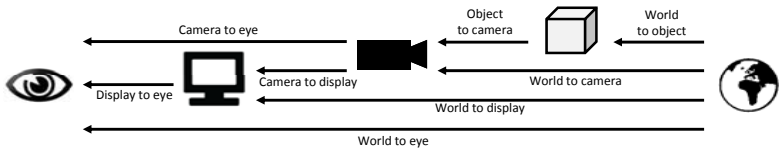
How does a basic AR application work?

- Main loop
  - Get a video frame from the camera
  - Estimate the position and the orientation of the camera
    - using the camera acquired images/video
      - detect & recognize marker(s)
        (*or natural objects, not basic in this case*)
    - using other sensor input (GPS, compass, accelerometer)
  - Render the augmented scene (join: video + virtual content)
  - Render GUI
  - Process user input
  - Update application status

Display Coordinate Systems

The spatial model of most AR displays can be defined as the spatial relationship of up to five components:

- the user's eye,
- the display,
- the camera,
- an object to be augmented, and
- the world.



Each coordinate transformation can be

- fixed and calibrated,
- tracked dynamically.

source: Schmalstieg & Hollerer 's book

Vision-based tracking

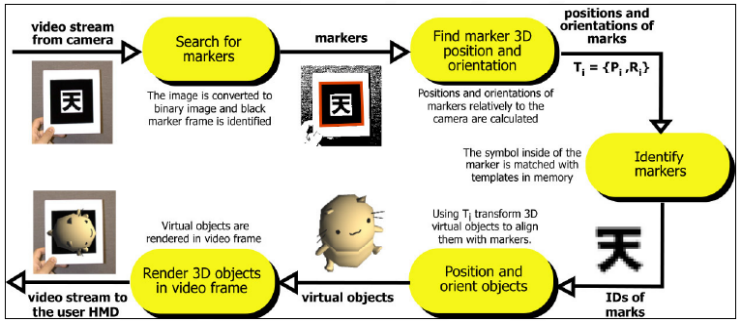
- Marker-based tracking
  - specially designed markers or fiducials
  - simpler and potentially more robust tracking algorithms
- Natural feature tracking
  - track the natural environment
- Both markers and natural features can be used for model based tracking
  - markers approach: the digital model exists first and a physical objet (ex: cardboard marker) is manufactured to match it
  - natural features approach: the physical object exists first, and a scanner is used to obtain a digital model to match it
    - ex: Kinect fusion
      https://www.microsoft.com/en-us/research/project/kinectfusion-project-page

Marker-based tracking

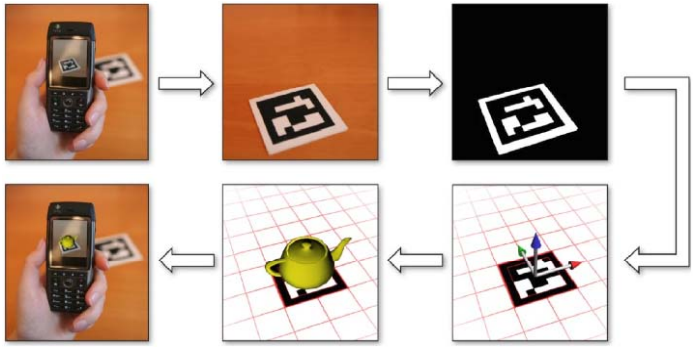
- Has been done for ~20 years
  - Some phones today are faster than computers at that time
- Several open source solutions exist
  - ARToolkit, ARTag, …
- Fairly simple to implement
  - Standard computer vision methods
- A rectangular marker provides 4 corner points
  - Enough for pose estimation!

Marker-based Augmented Reality

ARToolkit tracking



Marker tracking overview

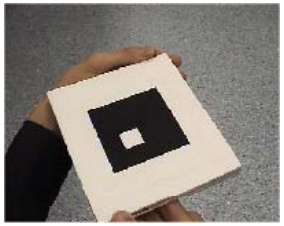


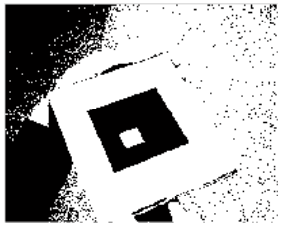
Marker-based tracking: marker characteristics

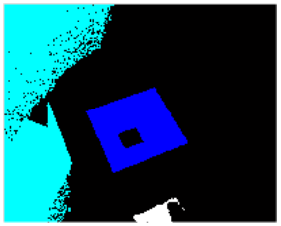
- Choosing the marker
  - It must be easily "extracted" from the image (**high contrast**)
    - adequate "color" and "contrast"
    - simple shape (ex: square or rectangle)
  - It must be possible to determine its orientation (**asymmetry**)
    - (the world coordinate system is, usually, associated to the marker)
    - => use an asymmetric pattern
  - It must have an adequate size & not too much detail (**size**)

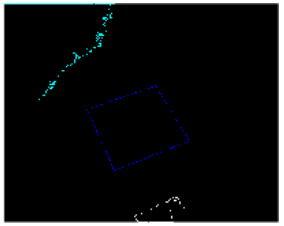
- Common markers
  - Black & white pattern
    - frequently used
      - white square/rectangle with black border (or vice-versa)
  - Colored pattern
    - may be useful in some situations

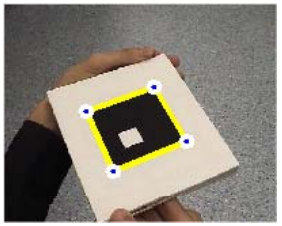
Marker-based AR steps

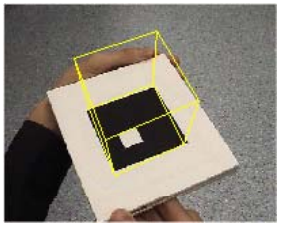

Original image


Binary image


Connected components


Contours


Corners


Augmented image

Marker-based AR steps (detailed)

1. Calibration of the real camera
   - intrinsic camera parameters
2. Acquisition of a real image, in which the marker is visible
3. Marker detection in the acquired image
4. Marker identification & orientation (/rotation) determination
5. Estimation of the pose (position and orientation) of the real camera relatively to the marker
   - extrinsic camera parameters
6. Creation of a virtual camera having the same parameters (intrinsic & extrinsic) of the real camera
7. Creation of a virtual image, obtained by rendering virtual objects using the virtual camera
8. Junction of the real and the virtual images
   (the virtual objects appear aligned with the real objects)

9. Tracking …

Camera modeling & calibration

(see additional slides)

Camera calibration

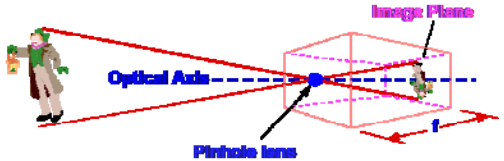
- Camera geometric model
  - how can we represent the image formation process ?
    - lens, image sensor, "pixelization", …
  - only interested in the geometry of image formation (no photometric concerns)
- Determination of the parameters of the model
  - requires imaging an object with some points whose 3D position is accurately known
  - how many points are needed ?
    - we'll see in the following

Camera model

- Image formation model
  - diaphragm model (pinhole)
  - lens model (thin lens)
  - perspective projection equation
  - aberrations (distortion)
- (Homogeneous coordinates)
- Camera model
  - relationship between world - (X,Y,Z) - and pixel - (I,J) - coordinates
  - perspective projection matrix
  - intrinsic & extrinsic parameters

Pinhole camera

- A simple camera model is the pinhole camera model
- Light passes through a small hole and falls on an image plane
- The image is inverted and scaled
- Early cameras used this approach



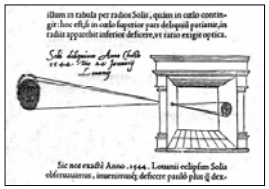
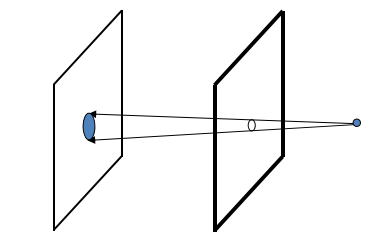
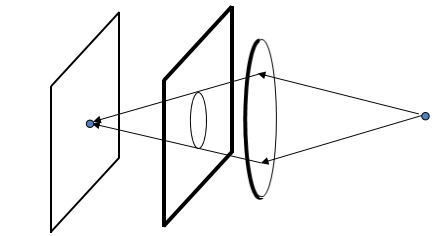


Image acquired with pinhole camera
Robert Kosara

Real camera

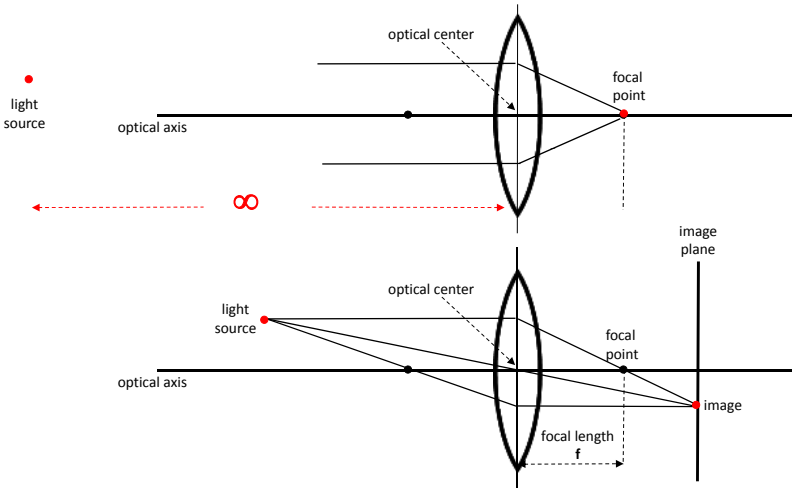
- Real cameras do not have pinholes
- A pinhole lets very little light in, leading to long exposures
- Larger holes lead to blurred images
- A lens can make a large hole act like a small one



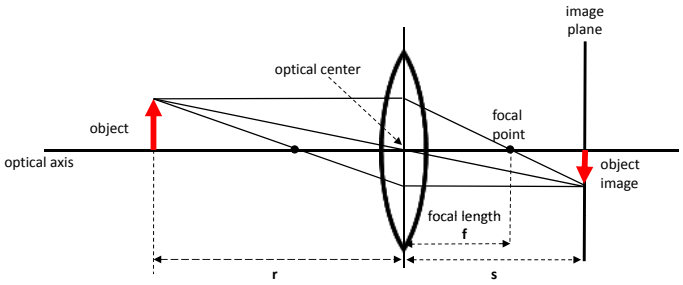


Imaging through a lens

- A light source at an infinite distance is focused on the focal point
- If the light source moves closer, the light rays collect behind the focal point



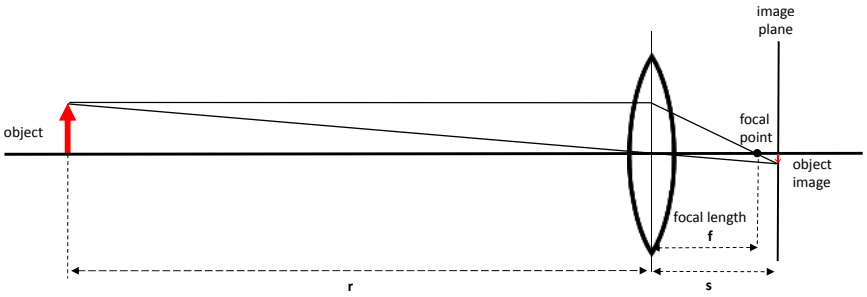
Thin lens equation



$$\frac{1}{r}+\frac{1}{s}=\frac{1}{f}$$

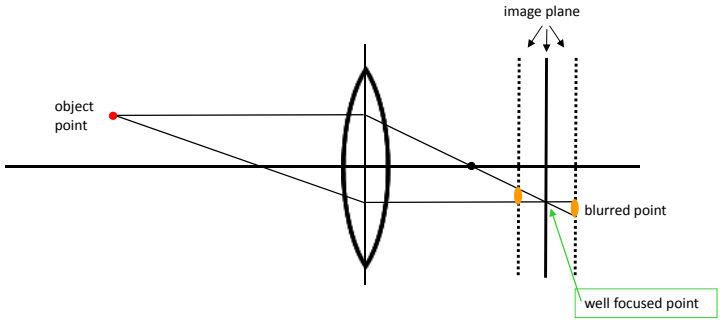
Gauss lens equation

Image formation model



- When r is large enough (r >> f) ➔ s ≈ f

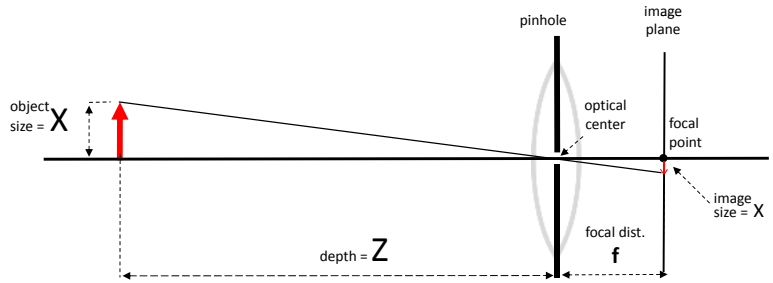
Image formation



- When the image plane is not on the right place the image is defocused
  - points are projected as blurred circles

Image formation model

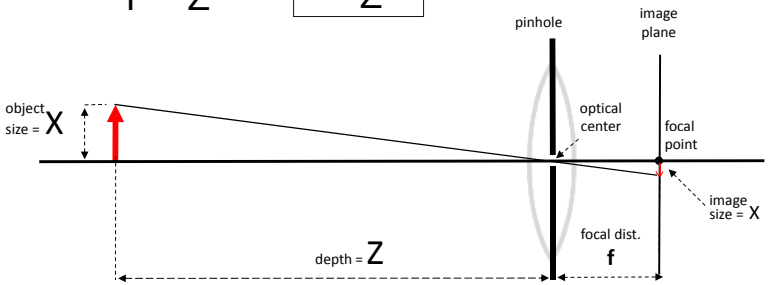
- Considering
  - objects distant enough
  - pinhole model
- The image is formed at the focal distance and is always focused (…!)



Perspective projection

- The image is a scaled version of the object
- The scaling depends on the distance to the object, Z (perspective projection)

$$\frac{x}{f} = \frac{X}{Z} \Rightarrow \boxed{x = \frac{f}{Z} X}$$



Weak perspective projection

- If the "object" extends through a small depth, ΔZ, when compared with the mean distance Z_0, the scaling can be considered constant (= f / Z_0) (weak perspective projection)

$$\Delta Z << Z_0 \Rightarrow \boxed{x \approx \frac{f}{Z_0} X}$$

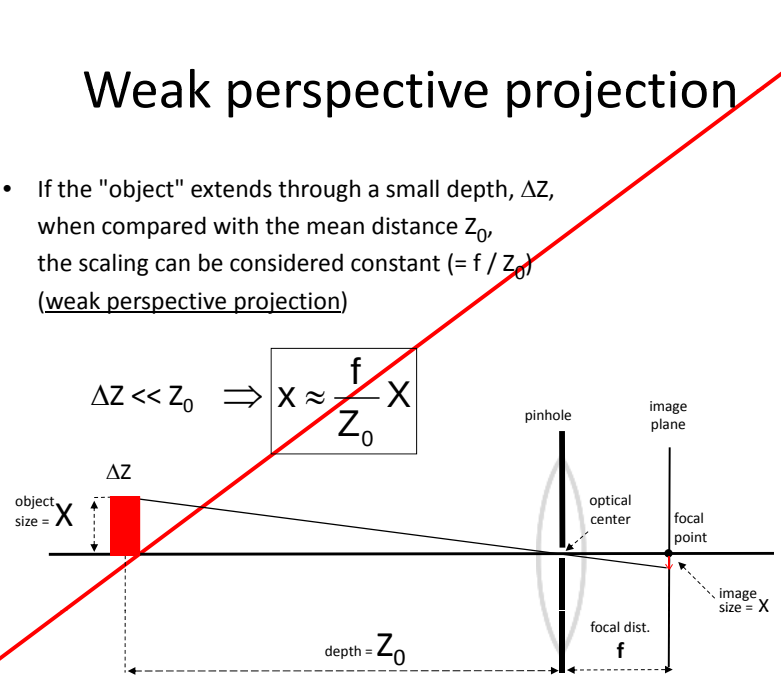


Image formation model

- The image plane is traditionally drawn in front of the optical center
- Avoids signal inversion, between object and image coordinates

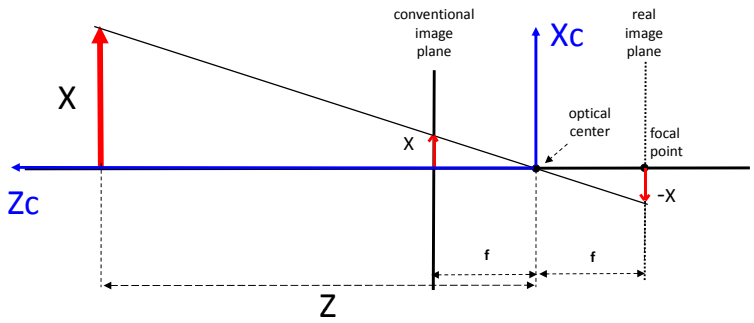
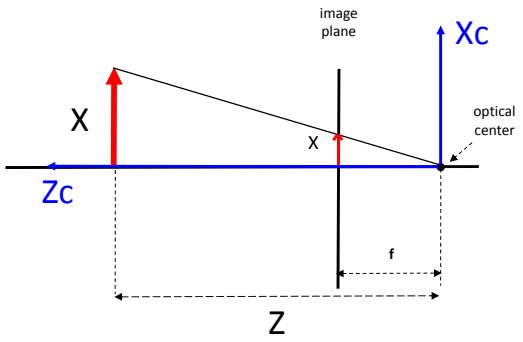
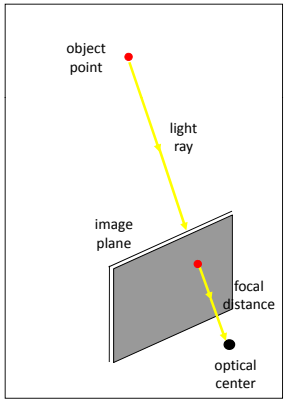


Image formation model



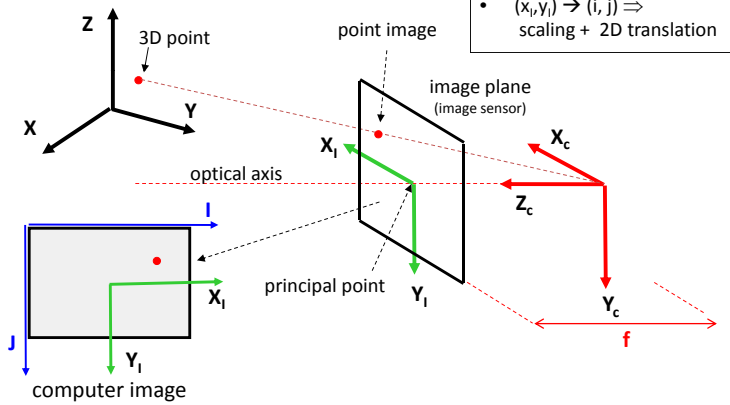
$$x = \frac{f}{Z} X \qquad y = \frac{f}{Z} Y$$



Camera model

Reference frames
- World - (X,Y,Z)
- Camera - (X_C,Y_C,Z_C)
- Sensor - (X_I,Y_I)
- Image - (I, J)

Coordinate transformations
- (x,y,z) → (x_C,y_C,z_C) ⇒ 3D rotation + 3D translation
- (x_C,y_C,z_C) → (x_I,y_I) ⇒ perspective projection
- (x_I,y_I) → (i, j) ⇒ scaling + 2D translation



Lens distortion

- The previous image formation model does not take into account distortions due to lens construction (ex. fish eye lenses) or bad quality lenses
- Distortion parameters can be estimated and image distortion can be corrected
- Distortion will be ignored in the following model

Distorted image
(barrel distortion)

Corrected image

Homogeneous coordinates

Homogeneous coordinates

- Homogeneous coordinates are a key tool for
  - 3D computer vision
  - 3D computer graphics
  - Modeling robotic manipulators
  - …
- They allow us to transform between reference frames with a single matrix multiplication
  - in normal Euclidean coordinates,
    rotation is expressed by a matrix multiplication
    but translation is expressed by an addition …

Homogeneous coordinates

- (N+1)-dimensional notation for points in N-dimensional Euclidean space
- Allows us to express translation and projection as linear operations

Normal coordinates

$$v = \begin{bmatrix} x \\ y \\ z \end{bmatrix}$$

Homogeneous coordinates

$$v = \begin{bmatrix} wx \\ wy \\ wz \\ w \end{bmatrix}$$

w is an arbitrary constant
wx = w·x

Example:

$$v = \begin{bmatrix} 4 \\ -2 \\ 5 \end{bmatrix} \quad \begin{matrix} w = 1 \\ \\ w = -2 \end{matrix} \quad \begin{matrix} v = \begin{bmatrix} 4 \\ -2 \\ 5 \\ 1 \end{bmatrix} \\ \\ v = \begin{bmatrix} -8 \\ 4 \\ -10 \\ -2 \end{bmatrix} \end{matrix}$$

- To recover normal coordinates, divide de first N components by (N+1)th, w.

Homogeneous coordinates

- 3D rotation

Normal coordinates

$$\begin{bmatrix} x' \\ y' \\ z' \end{bmatrix} = [R] \begin{bmatrix} x \\ y \\ z \end{bmatrix} = \begin{bmatrix} R_{11} & R_{12} & R_{13} \\ R_{21} & R_{22} & R_{23} \\ R_{31} & R_{32} & R_{33} \end{bmatrix} \begin{bmatrix} x \\ y \\ z \end{bmatrix}$$

Homogeneous coordinates

$$\begin{bmatrix} x' \\ y' \\ z' \\ 1 \end{bmatrix} = \begin{bmatrix} R & 0 \\ 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix} = \begin{bmatrix} R_{11} & R_{12} & R_{13} & 0 \\ R_{21} & R_{22} & R_{23} & 0 \\ R_{31} & R_{32} & R_{33} & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix}$$

- Some properties of matrix R
  - orthogonal matrix $\Rightarrow RR^T=I \quad \Rightarrow R^{-1}=R^T$
    - the dot product of any pair of rows or any pair of columns is zero
  - normalized matrix: the squares of the elements in any row or columns sum to 1
  - the rows of R represent the coordinates in the original space of unit vectors along the coordinate axis of the rotated space
  - the columns of R represent the coordinates in the rotated space of unit vectors along the coordinate axis of the original space

Homogeneous coordinates

- 3D translation

Normal coordinates

$$\begin{bmatrix} x' \\ y' \\ z' \end{bmatrix} = [T] + \begin{bmatrix} x \\ y \\ z \end{bmatrix} = \begin{bmatrix} T_X \\ T_Y \\ T_Z \end{bmatrix} + \begin{bmatrix} x \\ y \\ z \end{bmatrix}$$

Homogeneous coordinates

$$\begin{bmatrix} x' \\ y' \\ z' \\ 1 \end{bmatrix} = \begin{bmatrix} I & T \\ 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & T_X \\ 0 & 1 & 0 & T_Y \\ 0 & 0 & 1 & T_Z \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix}$$

- Translation, expressed as a sum in normal coordinates is transformed into a product, in homogeneous coordinates

Homogeneous coordinates

- 3D (rotation + translation)

Homogeneous coordinates

$$\begin{bmatrix} x' \\ y' \\ z' \\ 1 \end{bmatrix} = \begin{bmatrix} R & T \\ 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix} = \begin{bmatrix} R_{11} & R_{12} & R_{13} & T_X \\ R_{21} & R_{22} & R_{23} & T_Y \\ R_{31} & R_{32} & R_{33} & T_Z \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix}$$

- Using homogeneous coordinates,
  Rotation and Translation can be expressed by a single matrix

Homogeneous coordinates

- Perspective projection (along Z)

$$x' = \frac{f}{z}x \qquad y' = \frac{f}{z}y$$

Homogeneous coordinates

$$\begin{bmatrix} wx' \\ wy' \\ 0 \\ w \end{bmatrix} = \begin{bmatrix} f & 0 & 0 & 0 \\ 0 & f & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix}$$

- 3D scaling

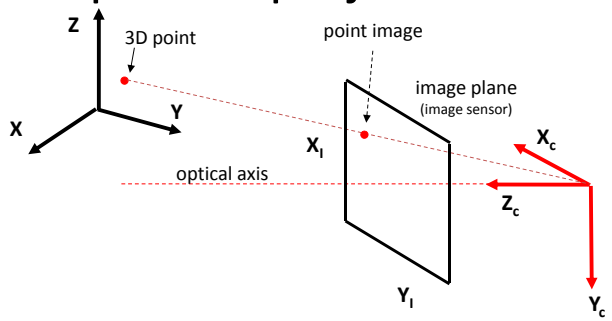
Homogeneous coordinates

$$\begin{bmatrix} x' \\ y' \\ z' \\ 1 \end{bmatrix} = \begin{bmatrix} S_X & 0 & 0 & 0 \\ 0 & S_Y & 0 & 0 \\ 0 & 0 & S_Z & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix}$$

Homogeneous coordinates

)

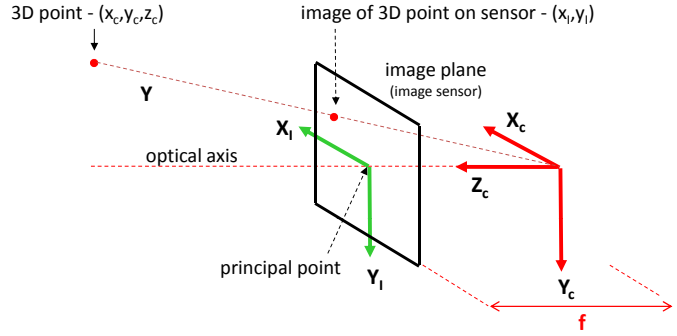
Perspective projection matrix



1. Transform world coordinates into camera coordinates

$$\begin{bmatrix} x_C \\ y_C \\ z_C \\ 1 \end{bmatrix} = \begin{bmatrix} R_{11} & R_{12} & R_{13} & T_X \\ R_{21} & R_{22} & R_{23} & T_Y \\ R_{31} & R_{32} & R_{33} & T_Z \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix} = \begin{bmatrix} R & T \\ 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix}$$

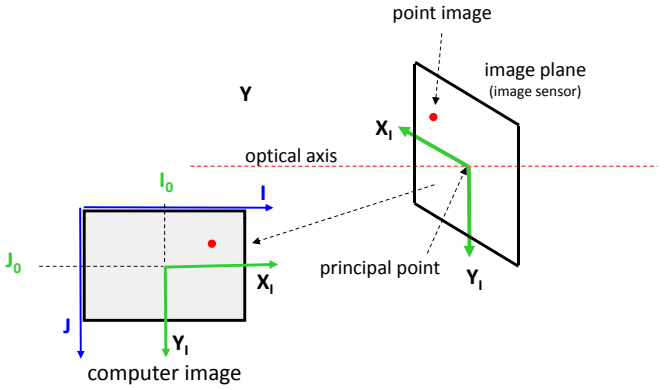
Perspective projection matrix



2. Transform camera coordinates into sensor coordinates

$$\begin{bmatrix} wx_I \\ wy_I \\ 0 \\ w \end{bmatrix} = \begin{bmatrix} f & 0 & 0 & 0 \\ 0 & f & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix} \begin{bmatrix} x_C \\ y_C \\ z_C \\ 1 \end{bmatrix}$$

Perspective projection matrix

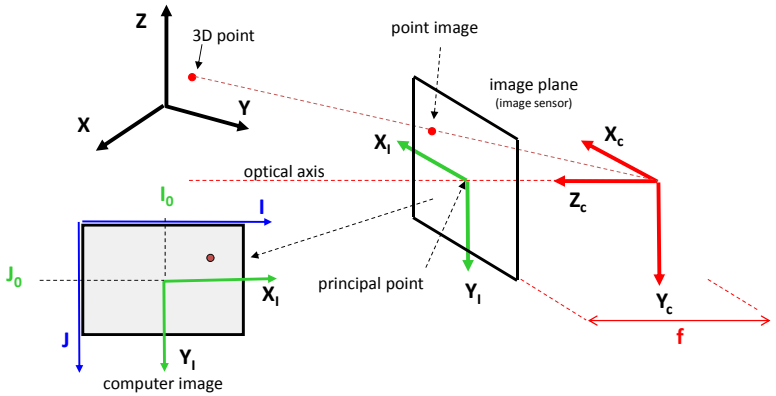


3. Transform sensor coordinates into pixel coordinates

$$\begin{bmatrix} i \\ j \\ 0 \\ 1 \end{bmatrix} = \begin{bmatrix} S_i & 0 & 0 & I_0 \\ 0 & S_j & 0 & J_0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x_I \\ y_I \\ 0 \\ 1 \end{bmatrix}$$

1/Si and 1/Sj represent the pixel size (in metric units), in the X and Y directions

Perspective projection matrix



Combining all the transformations:

$$\begin{bmatrix} wi \\ wj \\ w \end{bmatrix} = \begin{bmatrix} S_i f & 0 & I_0 & 0 \\ 0 & S_j f & J_0 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix} \begin{bmatrix} R_{11} & R_{12} & R_{13} & T_X \\ R_{21} & R_{22} & R_{23} & T_Y \\ R_{31} & R_{32} & R_{33} & T_Z \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix} = \begin{bmatrix} K_X & 0 & I_0 \\ 0 & K_Y & J_0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} R_{11} & R_{12} & R_{13} & T_X \\ R_{21} & R_{22} & R_{23} & T_Y \\ R_{31} & R_{32} & R_{33} & T_Z \end{bmatrix} \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix} = [K][R \mid T] \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix}$$

$K_X = S_i f$
$K_Y = S_j f$

Perspective projection matrix

$$[C] = [K][R \mid T] = \begin{bmatrix} K_X & 0 & I_0 \\ 0 & K_Y & J_0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} R_{11} & R_{12} & R_{13} & T_X \\ R_{21} & R_{22} & R_{23} & T_Y \\ R_{31} & R_{32} & R_{33} & T_Z \end{bmatrix} = \begin{bmatrix} K_X R_{11} + I_0 R_{31} & K_X R_{12} + I_0 R_{32} & K_X R_{13} + I_0 R_{33} & K_X T_X + I_0 T_Z \\ K_Y R_{21} + J_0 R_{31} & K_Y R_{22} + J_0 R_{32} & K_Y R_{23} + J_0 R_{33} & K_Y T_Y + J_0 T_Z \\ R_{31} & R_{32} & R_{33} & T_Z \end{bmatrix}$$

$$[C] = \begin{bmatrix} C_{11} & C_{12} & C_{13} & C_{14} \\ C_{21} & C_{22} & C_{23} & C_{24} \\ C_{31} & C_{32} & C_{33} & C_{34} \end{bmatrix} \qquad \begin{bmatrix} wi \\ wj \\ w \end{bmatrix} = \begin{bmatrix} C_{11} & C_{12} & C_{13} & C_{14} \\ C_{21} & C_{22} & C_{23} & C_{24} \\ C_{31} & C_{32} & C_{33} & C_{34} \end{bmatrix} \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix}$$

- $[C]$ - Perspective Projection Matrix of the camera
  - using it, the 2D image coordinates of a known 3D point can be obtained
  - also known as Direct Linear Transform (DLT) matrix
- $[K]$ - Intrinsic Parameter Matrix
  - represents the internal caracteristics of the camera
- $[R \mid T]$ - Extrinsic Parameter Matrix
  - represents the position and orientation of the camera relatively to the world coordinate system

Camera parameters

- Intrinsic parameters
  - principal point – I_0, J_0
    - intersection of the optical axis of the lens with the image sensor (pixel coord.)
    - usually not coincident with the image center
  - scale factors – $K_X = S_i f$, $K_Y = S_j f$
    - impossible to separate the S's from f, unless one of them is known *a priori*
  - *distortion coefficients* *(not considered in the previous model)*
  - *skew induced by angle between sensor axes*
    (not considered in the model; usually negligible)

$$[K] = \begin{bmatrix} K_X & s & I_0 \\ 0 & K_Y & J_0 \\ 0 & 0 & 1 \end{bmatrix}$$

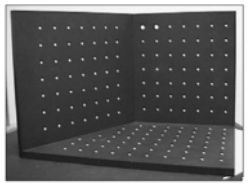
- Extrinsic parameters (camera pose)
  - rotation matrix
  - translation vector

Camera calibration

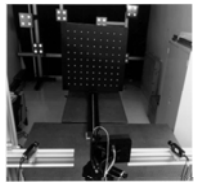
- Calibration is the process of estimating the intrinsic and extrinsic parameters of the camera.
- It can be thought of as a two stage process:
  - estimating matrix *C*, and
  - estimating the intrinsic and extrinsic parameters from *C*
- In many cases, particularly for stereo, the second stage is not necessary.
- Calibration consists of the following steps:
  - acquire an image of a set of known 3D points
  - determine the 2D image coordinates of each point
  - from each set of a 3D point and the corresponding 2D pixel a set of 2 equations results
  - establish enough correspondences to build a set build a set of equations from which the and the elements of matrix *C* can be calculated
- How many point correspondences are needed ?

Camera calibration

- Matrix *C* is defined up to an arbitrary scale factor and has therefore only 11 independent entries.
- So, one of the 12 C_{ij}'s can be arbitrarily chosen
- Since each pair of 3D-2D corresponding points provides 2 equations in the unknown C_{ij}'s, at least 5.5 points (!) ($\Rightarrow$ 6) are necessary
- In practice, much more than 6 points should be used
- To avoid degeneracies, calibration points must be non-coplanar



calibration target



stereo camera pair, calibration targets and translation stage

Camera calibration procedure DLT method

- Acquire an image of an object with at least 6 non-coplanar points whose 3D coordinates are known with high accuracy
  - in practice much more than 6 points should be used
  - points should cover all the field of view (FOV) of the camera
- Determine the 2D image coordinates of each point
- For each pair of corresponding 3D-2D points a set of 2 equations results from the perspective projection matrix

$$\begin{bmatrix} wi \\ wj \\ w \end{bmatrix} = \begin{bmatrix} C_{11} & C_{12} & C_{13} & C_{14} \\ C_{21} & C_{22} & C_{23} & C_{24} \\ C_{31} & C_{32} & C_{33} & C_{34} \end{bmatrix} \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix}$$

$$x_k C_{11} + y_k C_{12} + z_k C_{13} + C_{14} - i_k x_k C_{31} - i_k y_k C_{32} - i_k z_k C_{33} - i_k C_{34} = 0$$
$$x_k C_{21} + y_k C_{22} + z_k C_{23} + C_{24} - j_k x_k C_{31} - j_k y_k C_{32} - j_k z_k C_{33} - j_k C_{34} = 0$$

set of 2 equations for a generic pair, k:
$(x_k, y_k, z_k) \leftrightarrow (i_k, j_k)$

Camera calibration procedure
DLT method

- Build a system of equations for the whole set of corresponding points:

$$\begin{bmatrix} x_1 & y_1 & z_1 & 1 & 0 & 0 & 0 & 0 & -i_1x_1 & -i_1y_1 & -i_1z_1 & -i_1 \\ 0 & 0 & 0 & 0 & x_1 & y_1 & z_1 & 1 & -j_1x_1 & -j_1y_1 & -j_1z_1 & -j_1 \\ x_2 & y_2 & z_2 & 1 & 0 & 0 & 0 & 0 & -i_2x_2 & -i_2y_2 & -i_2z_2 & -i_2 \\ 0 & 0 & 0 & 0 & x_1 & y_1 & z_1 & 1 & -j_2x_2 & -j_2y_2 & -j_2z_2 & -j_2 \\ . & . & . & . & . & . & . & . & . & . & . & . \\ . & . & . & . & . & . & . & . & . & . & . & . \\ . & . & . & . & . & . & . & . & . & . & . & . \\ x_N & y_N & z_N & 1 & 0 & 0 & 0 & 0 & -i_Nx_N & -i_Ny_N & -i_Nz_N & -i_N \\ 0 & 0 & 0 & 0 & x_N & y_N & z_N & 1 & -j_Nx_N & -j_Ny_N & -j_Nz_N & -j_N \end{bmatrix} \begin{bmatrix} C_{11} \\ C_{12} \\ C_{13} \\ C_{14} \\ C_{21} \\ C_{22} \\ C_{23} \\ C_{24} \\ C_{31} \\ C_{32} \\ C_{33} \\ C_{34} \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \end{bmatrix}$$

- Imposing some restriction, solve this homogeneous system of equations:
  - $C_{34}=1$,
    and solve using the least-squares method or
  - $||\,[C_{31}\ C_{32}\ C_{33}]\,|| = ||\,[R_{31}\ R_{32}\ R_{33}]\,|| = 1$,
    and solve using Lagrange multipliers method

Camera calibration methods

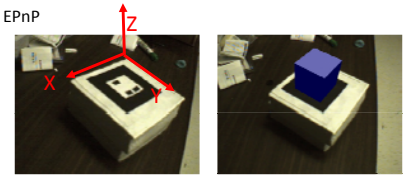
- Several methods have been proposed:
  - Karara & Abdel-Aziz
    - DLT (Direct Linear Transform) method *(the previously referred method)*
    - *Direct linear transformation from comparator coordinates into object space coordinates in close-range photogrammetry, 1971*
  - Tsai
    - *A versatile camera calibration technique for high accuracy 3D machine vision metrology using off-the-shelf TV cameras and lenses,IEEE-JRA,1987*
  - Heikkilä & Silvén
    - *A Four-step Camera Calibration Procedure with Implicit Image Correction, CVPR97*
  - Zhang
    - *Flexible Camera Calibration by Viewing a Plane from Unknown Orientations, ICCV99*
  - …

Obtaining the intrinsic & extrinsic parameters

- Once the perspective projection matrix C is obtained, using the DLT method, we can recover the intrinsic and extrinsic parameters from C.
  - This is a decomposition problem, not an estimation problem.
- Some of the calibration methods (Tsai, Zhang, ...) return directly
  - the intrinsic and extrinsic parameters
  - some distortion parameters
- See
  - "Camera Calibration Toolbox for Matlab" for extensive list and implementations
    - http://www.vision.caltech.edu/bouguetj/calib_doc/htmls/parameters.html
  - OpenCV documentation
    - http://docs.opencv.org/modules/calib3d/doc/camera_calibration_and_3d_reconstruction.html (2014-11-14)

Camera pose estimation

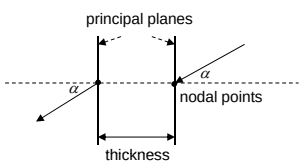
- Sometimes, the intrinsic parameters are kept constant while the extrinsic parameters are varying (ex: moving camera)
- In those cases, to update the Perspective Projection Matrix, only the extrinsic parameters need to be updated
- Camera Pose (rotation + translation) estimation algorithms
  - From 3D-2D points correspondences:
    - DLT algorithm + Extraction of K, R, T
      - not useful in the above situation
    - P3P algorithm (PnP – Perspective n Point problem) – OpenCV: solvePNP() and solvePnPRansac()
    - POSIT (=> 4 non-coplanar 3D points) – OpenCV: cvPOSIT()
    - Non-linear minimization
  - From a planar structure
    - homography between, at least, 4 points on a plane and the corresponding image points
    - Coplanar POSIT
    - PnP and EPnP



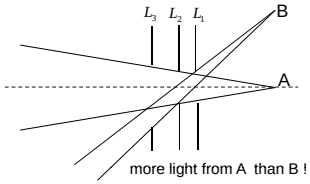
http://projekter.aau.dk/projekter/files/14427578/A_Comparison_of_2D-3D_Pose_Estimation_Methods.pdf

Lens distortions

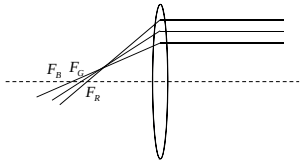
Compound (Thick) Lens



Vignetting



Chromatic Abberation



Lens has different refractive indices for different wavelengths.

Radial and Tangential Distortion

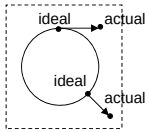
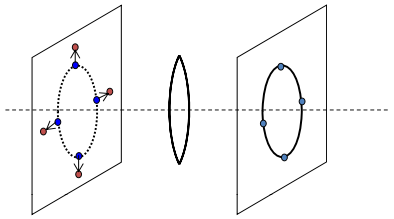
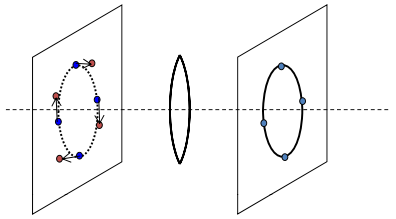


image plane

Lens geometric distortion



Radial distortion



Tangential distortion

- Both due to lens imperfection
- Radial: is a side effect of the round glass elements within a lens and the effect of bending light more/less near the edges of the lens than we encounter near the center of the lens
- Tangential: The tangential distortion is due to "decentering", or imperfect centering of the lens components and other manufacturing defects in a compound lens
- Rectify with geometric camera calibration

Lens geometric distortion

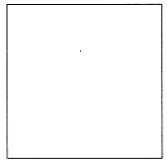
- Radial distortion

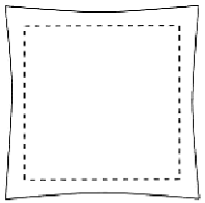
object

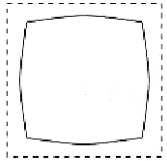
(a) (b) (c)

Figure 2.13 Radial lens distortions: (a) barrel, (b) pincushion, and (c) fisheye. The fisheye image spans almost 180° from side-to-side.

images







no distortion

pinchusion distortion

barrel distortion

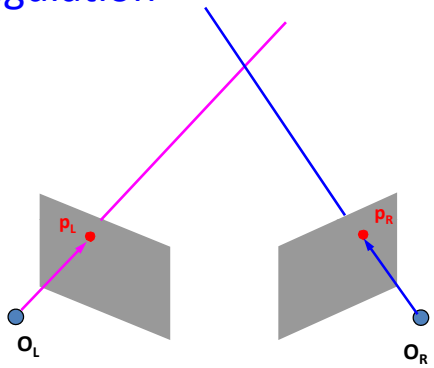
Geometric distortion model

- Several distortion models available.
- In this course, we will not analyse them.
- For more information:
  - Camera Calibration Toolbox: http://www.vision.caltech.edu/bouguetj/calib_doc/
  - OpenCV: http://docs.opencv.org/trunk/modules/calib3d/doc/camera_calibration_and_3d_reconstruction.html

3D data acquisition

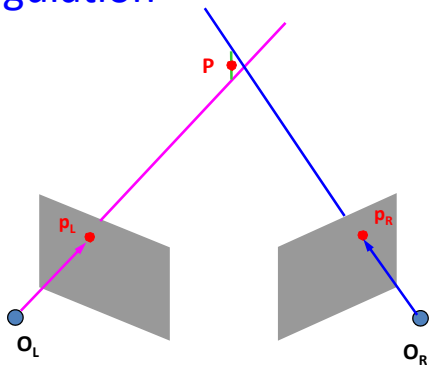
3D reconstruction by triangulation

- Assuming that
  - cameras are calibrated
  - correspondences between points of the stereo pair are established
- For each pair of corresponding pixels
  - determine the line-of-sight associated with each pixel
  - intersect the 2 lines-of-sight



3D reconstruction by triangulation

- Problem:
  - the 2 rays will not actually intersect in space due to errors in calibration and correspondences, and pixelization
- Solution:
  - find a point in space with minimum distance (in the least squares sense) from both rays



Left line-of-sight →

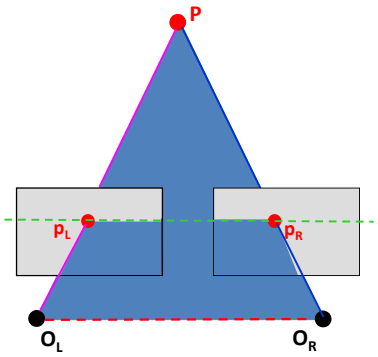
$$\begin{cases} (C_{L_{31}}i - C_{L_{11}})\cdot x + (C_{L_{32}}i - C_{L_{12}})\cdot y + (C_{L_{33}}i - C_{L_{13}})\cdot z + = C_{L_{14}} - C_{L_{34}}i \\ (C_{L_{31}}j - C_{L_{21}})\cdot x + (C_{L_{32}}j - C_{L_{22}})\cdot y + (C_{L_{33}}j - C_{L_{23}})\cdot z + = C_{L_{24}} - C_{L_{34}}j \end{cases}$$

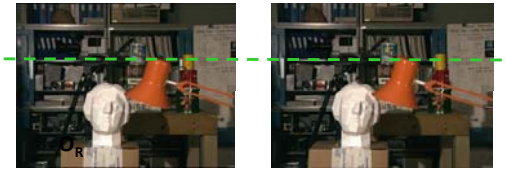
Right line-of-sight →

$$\begin{cases} (C_{R_{31}}i - C_{R_{11}})\cdot x + (C_{R_{32}}i - C_{R_{12}})\cdot y + (C_{R_{33}}i - C_{R_{13}})\cdot z + = C_{R_{14}} - C_{R_{34}}i \\ (C_{R_{31}}j - C_{R_{21}})\cdot x + (C_{R_{32}}j - C_{R_{22}})\cdot y + (C_{R_{33}}j - C_{R_{23}})\cdot z + = C_{R_{24}} - C_{R_{34}}j \end{cases}$$

Image matching: the correspondence problem

- Matching is easier if the cameras are perfectly equal and have parallel axis
  - corresponding points are on the same lines, in each image

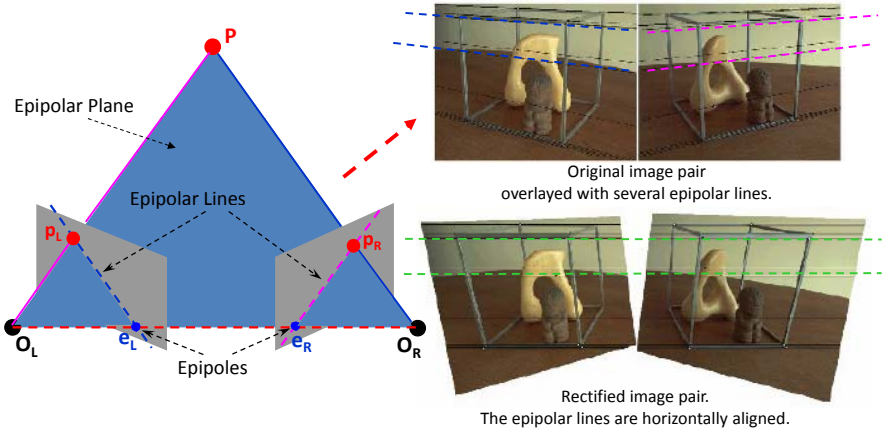




Tsukuba stereo pair

Image matching: the correspondence problem

- If the axis are not parallel the images can be rectified



Original image pair
overlayed with several epipolar lines.

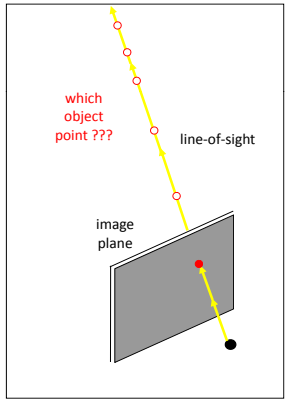
Rectified image pair.
The epipolar lines are horizontally aligned.

"Inverting" the perspective transformation

- Matrix C can't be inverted
  - it is, obviously, not possible to obtain the coordinates, (x,y,z), of a 3D point, given the coordinates, (i,j), of the corresponding 2D pixel
- Given matrix C and the coordinates of a pixel, (i,j), the following set of equations can be obtained:

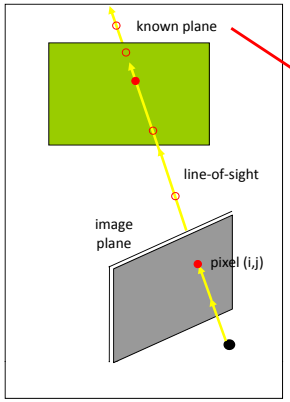
$$(C_{31}i - C_{11})\cdot x + (C_{32}i - C_{12})\cdot y + (C_{33}i - C_{13})\cdot z = C_{14} - C_{34}i$$
$$(C_{31}j - C_{21})\cdot x + (C_{32}j - C_{22})\cdot y + (C_{33}j - C_{23})\cdot z = C_{24} - C_{34}j$$

- These are the equations of 2 planes whose intersection determines the line-of-sight of pixel (i,j).



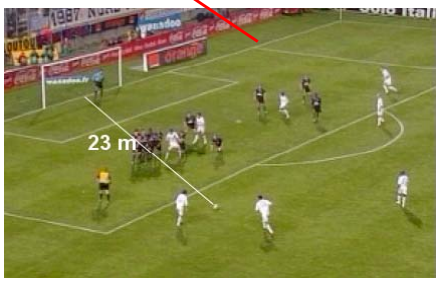
"Inverting" the perspective transformation

- If one knows that the scene point lies on a given 3D plane, its 3D coordinates can be obtained with a single image



$$\begin{cases}(C_{31}i - C_{11})\cdot x + (C_{32}i - C_{12})\cdot y + (C_{33}i - C_{13})\cdot z = C_{14} - C_{34}i \\ (C_{31}j - C_{21})\cdot x + (C_{32}j - C_{22})\cdot y + (C_{33}j - C_{23})\cdot z = C_{24} - C_{34}j \\ A\cdot x + B\cdot y + C\cdot z = D\end{cases}$$

ex: **z=0** (➔ A=0, B=0, C=1, D=0)



(Homography)

$$\begin{bmatrix} wi \\ wj \\ w \end{bmatrix} = \begin{bmatrix} C_{11} & C_{12} & C_{13} & C_{14} \\ C_{21} & C_{22} & C_{23} & C_{24} \\ C_{31} & C_{32} & C_{33} & C_{34} \end{bmatrix} \cdot \begin{bmatrix} x \\ y \\ 0 \\ 1 \end{bmatrix} \quad z = 0$$

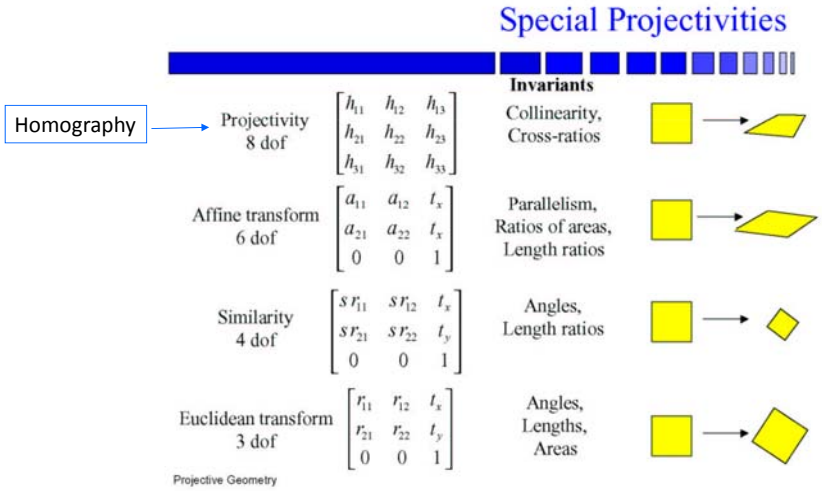
$$\begin{bmatrix} wi \\ wj \\ w \end{bmatrix} = \begin{bmatrix} H_{11} & H_{12} & H_{13} \\ H_{21} & H_{22} & H_{23} \\ H_{31} & H_{32} & H_{33} \end{bmatrix} \cdot \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$$

Homography
between 2 planes

3D data acquisition

Homographies

Geometric transformations



Homography

- Perspective projection of a plane
  - Other designations:
    - *texture-map, collineation, planar projective map*
  - Modeled as a 2D distortion (*warp*) using homogeneous coordinates

$$\begin{bmatrix} wx' \\ wy' \\ w \end{bmatrix} = \begin{bmatrix} H_{11} & H_{12} & H_{13} \\ H_{21} & H_{22} & H_{23} \\ H_{31} & H_{32} & H_{33} \end{bmatrix} \cdot \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$$

- Matrix H is defined up to an arbitrary scale factor and has therefore only 8 independent entries.

Homography

- Determination of the matrix coefficients
  - establish the correspondences between at least 4 points belonging to 2 different planes
  - solve the following set of equations (for the chosen **n** points)

$$\begin{bmatrix} x_1 & y_1 & 1 & 0 & 0 & 0 & -x'_1 x_1 & -x'_1 y_1 & -x'_1 \\ 0 & 0 & 0 & x_1 & y_1 & 1 & -y'_1 x_1 & -y'_1 y_1 & -y'_1 \\ x_2 & y_2 & 1 & 0 & 0 & 0 & -x'_2 x_2 & -x'_2 y_2 & -x'_2 \\ 0 & 0 & 0 & x_2 & y_2 & 1 & -y'_2 x_2 & -y'_2 y_2 & -y_2 \\ . & . & . & . & . & . & . & . & . \\ . & . & . & . & . & . & . & . & . \\ . & . & . & . & . & . & . & . & . \\ x_N & y_N & 1 & 0 & 0 & 0 & -x'_N x_N & -x'_N y_N & -x'_N \\ 0 & 0 & 0 & x_N & y_N & 1 & -y'_N x_N & -y'_N y_N & -y'_N \end{bmatrix} \begin{bmatrix} H_{11} \\ H_{12} \\ H_{13} \\ H_{21} \\ H_{22} \\ H_{23} \\ H_{31} \\ H_{32} \\ H_{33} \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ 0 \\ 0 \\ ... \\ ... \\ ... \\ 0 \\ 0 \end{bmatrix}$$

A (2n x 9) **H** (9) **0** (2n)

Solution: **H** = eigenvector of $\mathbf{A}^T\mathbf{A}$ corresponding to the smallest eigenvalue

Homographies in Augmented Reality

- Can be used:
  - to determine the position of the real camera used to acquire the images, and use this information to position the virtual camera, used to *render* the virtual objects
  - to generate a "frontal view" of the marker (reference pattern) before comparing it with stored markers (correction of the perspective distortion)

Camera pose estimation using an homography

- We have seen that the Perspective Projection Matrix [*C*] that establishes the relationship between the 3D coordinates of a point in space and its 2D coordinates

$$\begin{bmatrix} wi \\ wj \\ w \end{bmatrix} = [C] \cdot \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix}$$

can be written as:

$$[C] = \begin{bmatrix} K_x & 0 & I_0 & 0 \\ 0 & K_y & J_0 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix} \cdot \begin{bmatrix} R_{11} & R_{12} & R_{13} & T_x \\ R_{21} & R_{22} & R_{23} & T_y \\ R_{31} & R_{32} & R_{33} & T_z \\ 0 & 0 & 0 & 1 \end{bmatrix} = \begin{bmatrix} K_x & 0 & I_0 \\ 0 & K_y & J_0 \\ 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} R_{11} & R_{12} & R_{13} & T_x \\ R_{21} & R_{22} & R_{23} & T_y \\ R_{31} & R_{32} & R_{33} & T_z \end{bmatrix} = [K][R \quad T]$$

http://www.robots.ox.ac.uk/~dclaus/cameraloc/position/comppose.htm

Camera pose estimation using an homography

- When we acquire an image of a plane to which we associate the 3D world coordinate system, so that the points on the plane verify the equation z=0, we have

$$\begin{bmatrix} wi \\ wj \\ w \end{bmatrix} = [K] \cdot [R \quad T] \cdot \begin{bmatrix} x \\ y \\ 1 \end{bmatrix} = \begin{bmatrix} K_x & 0 & I_0 \\ 0 & K_y & J_0 \\ 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} R_{11} & R_{12} & T_x \\ R_{21} & R_{22} & T_y \\ R_{31} & R_{31} & T_z \end{bmatrix} \cdot \begin{bmatrix} x \\ y \\ 1 \end{bmatrix} = [K] \cdot [r_1 \quad r_2 \quad t] \cdot \begin{bmatrix} x \\ y \\ 1 \end{bmatrix} = [H] \cdot \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$$

where $[H] = [K] \cdot [r_1 \quad r_2 \quad t]$

represents an homography between the points on the plane and the points on the image.

- If we calculate the homography [*H*] and we know matrix [*K*] that contains the intrinsic camera parameters we can calculate the pose of the camera: $[P] = [r_1 \quad r_2 \quad t]$

$$[H] = [K] \cdot [P]$$
$$[K]^{-1} \cdot [H] = [K]^{-1} \cdot [K] \cdot [P]$$
$$[P] = [K]^{-1} \cdot [H] = [p_1 \quad p_2 \quad p_3] = [r_1 \quad r_2 \quad t]$$

Camera pose estimation using an homography

- At first, one could be tempted to say that $r_1 = p_1$, $r_2 = p_2$, $t = p_3$
- However, taking into account tthat matrix [*H*] is defined up to an arbitrary scale factor, it is probable that p_1 and p_2 are not in normalized form:

$$\|p_1\| = \|r_1\| = 1 \qquad \|p_2\| = \|r_2\| = 1 \qquad \|p_1 \cdot p_2\| = \|r_1 \cdot r_2\| = 0$$

- In order to normalize [*P*] we must divide its members by a normalization factor λ:

$$[\lambda] = \sqrt{\|p_1\| \|p_2\|}$$

NOTE: the sign of λ can be positive or negative and that will influence the sign of the elements of the translation vector; the choice of the adequate sign must take into account the position of the camera relatively to the reference plane

- Then, we calculate:

$$r_1 = \frac{p_1}{\lambda} \qquad r_2 = \frac{p_2}{\lambda} \qquad t = \frac{p_3}{\lambda}$$

NOTE: Do not forget that p_1, p_2 , p_3, r_1, r_2 ant t are, in fact, vectors.
In the following step we shall use vectorial notation to represent r_1 and r_2

Camera pose estimation using an homography

- However, this step does not yet guarantee that: $\vec{r}_1 \cdot \vec{r}_2 = 0$

  as it may happen that r_1 and r_2 are not exactly perpendicular.

- So, a second correction is necessary:

$$\vec{a} = \vec{r}_1 + \vec{r}_2 \qquad \vec{r}_{1f} = \frac{\vec{a}}{\|\vec{a}\|} + \frac{\vec{d}}{\|\vec{d}\|}$$
$$\vec{b} = \vec{r}_1 \times \vec{r}_2$$
$$\vec{d} = \vec{a} \times \vec{b} \qquad \vec{r}_{2f} = \frac{\vec{a}}{\|\vec{a}\|} - \frac{\vec{d}}{\|\vec{d}\|}$$

- Vectors r_{1f} and r_{2f} can still not be in normalized form, so it is necessary to normalize them

$$\vec{r}_{1f} = \frac{\vec{r}_{1f}}{\|\vec{r}_{1f}\|}$$
$$\vec{r}_{2f} = \frac{\vec{r}_{2f}}{\|\vec{r}_{2f}\|}$$

- Finally, we calculate $\vec{r}_{3f} = \vec{r}_{1f} \times \vec{r}_{2f}$

Camera pose in ARtoolkit

- Pose from homography function will give only a very coarse pose (usually called an initial guess).
- That's why one typically does an iterative refinement step.
- Other than pose via DLT, this is usually not done directly, but iteratively.
- The standard method for this is non-linear refinement using Gauss-Newton or Levenberg-Marquardt iteration.
- ARtoolkit does not use these methods, but does a refinement by "variation":
  - Starting with the initial guess it varies the pose into all directions (rotation only, position is adjusted respectively) and chooses the best variation.
  - This step is iterated several times.
- While it is a very simple method it is not suggested because it requires a lot of iterations.
- Gauss-Newton or Levenberg-Marquardt refinement, converges a lot faster due to putting more knowledge into the "system" (Jacobian matrix):
  ~ *3 vs. 300 iterations, according to Daniel Wagner*
- ARToolKit v4.3 uses the standard ICP algorithm for pose estimation.
  (*a short description of ICP is available in Wikipedia*)

source:
http://www.hitl.washington.edu/artoolkit/mail-archive/message-thread-01895--ARToolKit--What-is-the-.html

Marker detection

Marker detection in the acquired image

The main steps are:

- Image binarization (thresholding)
- Connected components (blobs) detection
- Checking for blobs size
  - reject blobs that are too small or too large
- Detect borders of blobs
- Detect corners
- Reject blobs that are not quadrilaterals
- Refine corner coordinates

Marker detection: image binarization / thresholding

- Image binarization (or thresholding) is one of the simplest forms of image segmentation
  - Image segmentation: the process of assigning a label to every pixel in an image such that pixels with the same label share certain characteristics
- The simplest way to use image binarization is to choose a threshold value, and classify all pixels with values above this threshold as white, and all other pixels as black.

Thresholding

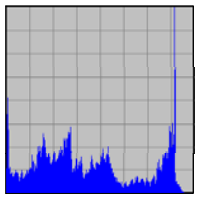
- The pixels of the image are classified taking into account the value of a certain image property
  - Assumption: that property is reasonably uniform inside each one of the regions
  - Frequently, the used property is the graylevel (or intensity)
  - Binarization
    - a single threshold is used (ThresholdValue)
    - if I(i,j)>ThresholdValue then $I_T(i,j)=1$ (*or* 0) else $I_T(i,j)=0$ (*or* 1)
  - Multilevel thresholding
    - several thresholds separate the classes
    - ex: 3 classes; 1- dark pixels; 2- ligth pixels; 3- the remaining ones
- It is the simplest segmentation technique => fast execution.

Thresholding

- Regions with uniform intensity give rise to strong peaks in the histogram.
- Select one or more intensity values (thresholds)
- Partition the image domain according to the intensity being higher or lower than those thresholds

$$R_i = \{x \in D_I : t_i \le I(x) < t_{i+1}\}$$

Original image



Histogram of the original image

Thresholding: threshold(s) selection

- Threshold
  - fixed
    - the threshold is previously chosen, it does not depend on image contents
    - only works well in some particular situations
  - dynamic
    - the contents of the image is taken into account when the threshold is chosen
    - the segmentation is more robust …
    - …but it takes more computation time
- The selection of the threshold(s) is usually done by analyzing the intensity histogram of the image
- Whenever possible one can use some available knowledge about the environment to choose the threshold automatically:
  - intensity characteristics of the objects
  - size of the objects
  - fraction of the image occupied by the objects
  - number of different types of objects that exist in the image

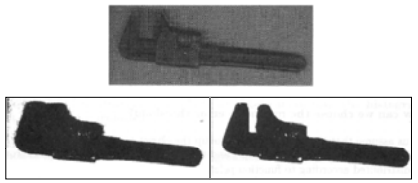
Thresholding: threshold(s) selection

- Selection of thresholds
  - manually
  - automatically
    - at local minima (requires smoothing of histogram first)
    - at intersection of fitted Gaussian functions
    - from prior knowledge (*see previous slide*)
- In general, good thresholds can be selected if the histogram peaks are
  - tall,
  - narrow,
  - symmetric,
  - and separated by deep valleys.

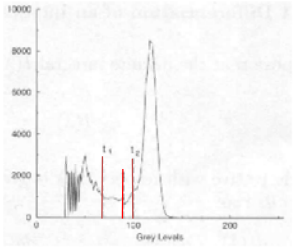
Thresholding

Hysteresis thresholding

- If there is no clear valley in the histogram of an image, it means that there are several background pixels that have similar gray level value to object pixels and vice-versa.
- Two thresholds, one at each side of the valley can be used in this case.
- Pixels above the high threshold are classified as **object** and below the low threshold as **background**.
- Pixels between the low and high thresholds are classified as object only if they are adjacent to other object pixels.



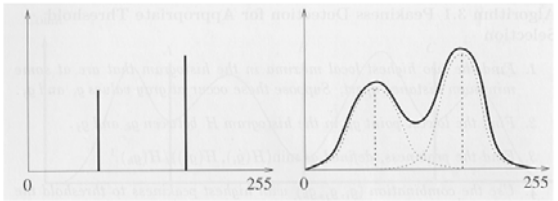
Single thresholding Hysteresis thresholding



Thresholding

Optimal thresholding

- Suppose that an image contains only two principal regions (e.g., object and background)
- We can minimize the number of misclassified pixels if we have some prior knowledge about the distributions of the gray level values that make up the object and the background.
- Assume that the distribution of gray-level values in each region follows a Gaussian distribution
- Drawbacks:
  - Prior probabilities might not be known.
  - Object/Background distributions might not be known.



Thresholding

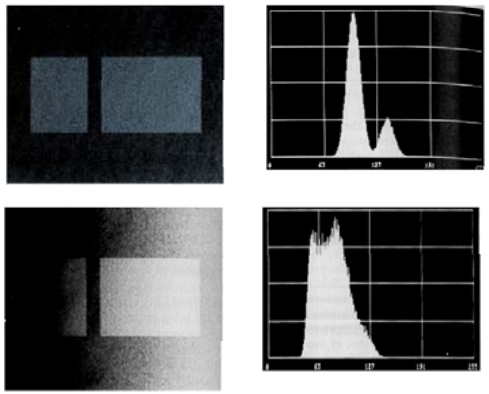
Otsu method

- A measure of region homogeneity is variance (i.e., regions with high homogeneity will have low variance).
- Otsu's method selects the threshold, T, such that
  - the intra-class variance is minimized and
  - the between-class variance is maximized .
- Iterative procedure
  - => calculate between-class variance for every possible T
- It does not depend on modeling the probability density functions, however, it assumes a bimodal distribution of gray-level values
  (i.e., if the image approximately fits this constraint, it will do a good job).
- Drawbacks:
  - Assumes that the histogram of the image is bimodal (i.e., two classes).
  - Breaks down when the two classes are very unequal (i.e., the classes have very different sizes).

Effect of illumination on thresholding

- A single threshold will not work well when we have **uneven illumination** due to shadows or due to the direction of illumination.

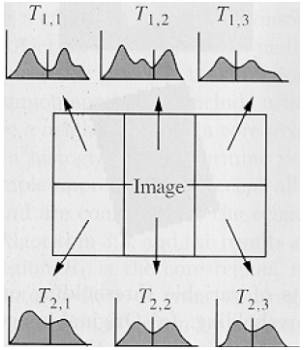
non-uniform illumination



Effect of illumination on thresholding

Handling nonuniform illumination: local thresholding

- The idea is to
  - partition the image into $m \times m$ subimages and
  - then choose a threshold $T_{i,j}$ for each subimage.
- This approach might lead to subimages having simpler histogram (e.g., bimodal)
- Drawback:
  - object contours not continuous in the transition between regions



http://docs.opencv.org/trunk/doc/py_tutorials/py_imgproc/py_thresholding/py_thresholding.html

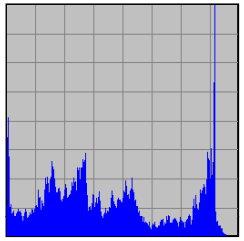
Thresholding

Drawbacks of thresholding:

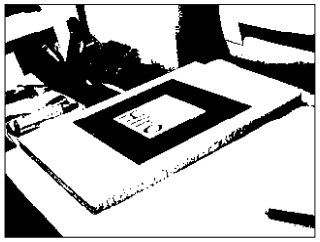
- Pixels assigned to a single class need not form coherent regions as the spatial locations of pixels are completely ignored
  - only hysteresis thresholding considers some form of spatial proximity.
- Threshold selection is not always straightforward.

Thresholding: examples

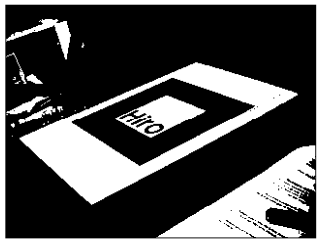
Original image



Histogram of the original image



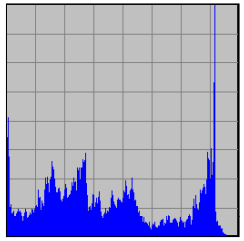
Binarized image(threshold=71)



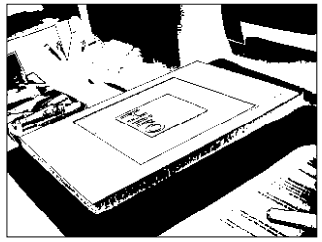
Binarized image(threshold=168)

Thresholding: examples

Original image



Histogram of the original image

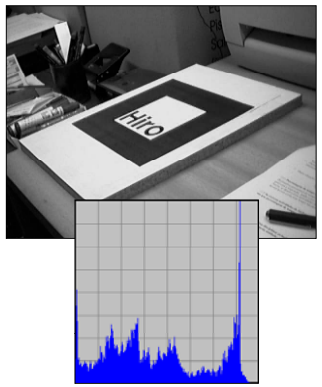


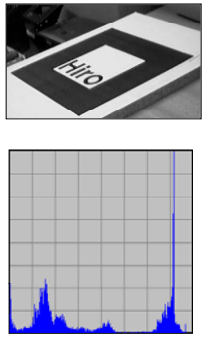
Binarization result using 2 thresholds;
pixels whose intensity is between 71 and 168 were set to black

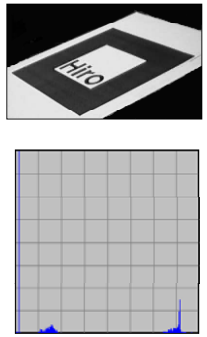
Thresholding*:*

threshold selection based on the intensity histogram

- The regions having uniform intensity give rise to peaks in the histogram.
- The separating thresholds are chosen in the valleys of the histogram.
- In general, it is possible to choose a "good threshold" if the peaks are high, narrow, symmetric and separated by deep valleys.





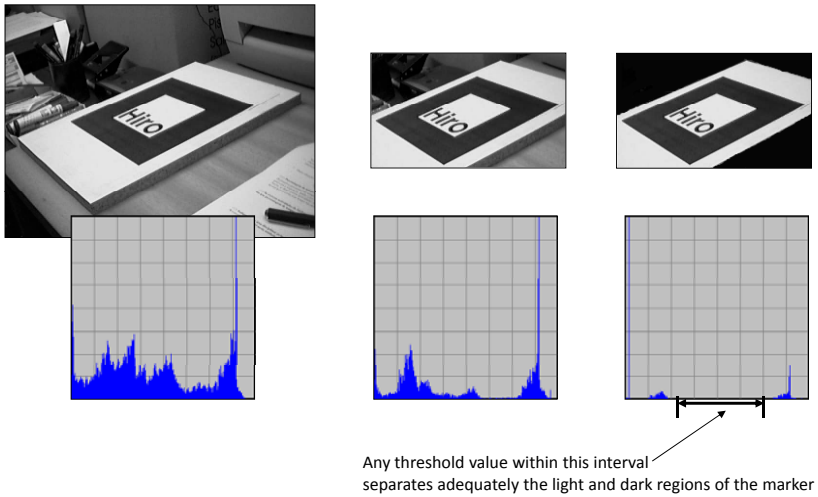


Segmentation in marker-based Augmented Reality

- If …
  - the marker is adequately chosen
  - and the environment is controlled (adequate illumination)
- … Then
  - it is possible to use a thresholding technique to segment the marker
    using a fixed threshold

- This is the solution used in ARToolkit

Segmentation
in marker-based Augmented Reality



Blob detection

- Input: binarized image
- Output: labels of each blob
- Methods: region labelling
  - recursive method
  - sequential method

Connected components (blobs) detection

- After obtaining a binary image, by thresholding, is is necessary to analyze it, in order to detect potential markers.
- For that, it is necessary to search for sets of connected pixels (white or black, it depends on the type of marker).
- A set of pixels in which each pixel is conneceted to other pixels is said to be a connected component or blob (binary large object)
  - definition: a pixel $p \in S$ is said to be connected to pixel $q \in S$ if there is a pathway from p to q consisting entirely of pixels belonging to S
  - connectivity:

Component/Region labelling

- A component/region labelling algorithm determines all the connected components of an image and attributes a unique label to all the points of the same component/region.
- Algorithms:
  - recursive
  - sequential
- In the following presentation of these algorithms it will be considered that object pixels are black and background pixels are white.

Region labelling: recursive algorithm

- **1)** Scan the image,
  from top to bottom, left to right,
  searching for a black pixel that has not yet been labelled
  and assign it a new label *L*.
- **2)** Recursively,
  assign label *L* to each one of its black neighbours.
- **3)** Stop
  when there are no more neighbours to be labelled.
- **4)** Go to step 1,
  continuing the scanning at the point where it was interrupted.

Region labelling: sequential algorithm

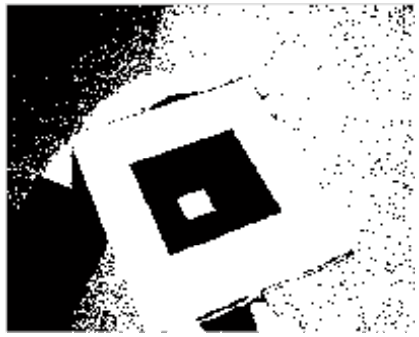
- **1)** Scan the image,
  from top to bottom, left to right
- **2)** If the pixel (X) is black, then:
  - analyze its neighbours that were already scanned
    - A,B,C,D when 8 conectivity is considered
    - A,B when 4 conectivity is considered
  - if all neighbours are white, assign X a new label
  - otherwise, analyze pixels A,B,C,D (or just A,B), by this order,
    and assign to X the label of the first one of those pixel that already has a label
  - if the *pixels* A,B,C,D (or justA,B) have different labels,
    take note that those labels are equivalent, in an equivalence list.

B	C	D
A	X	

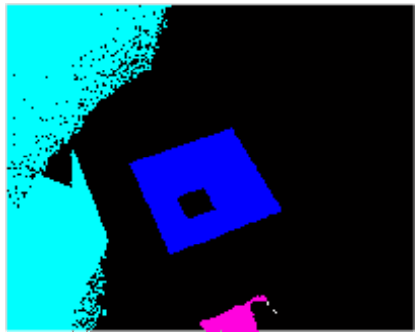
	B
A	X

- **3)** Go to step 2 until all the image is scanned
- **4)** When this first scanning ends, process the equivalence list and
  assign a single labels to all labels that are equivalent among each other
  (for exemple, the lowest of the equivalent labels)
- **5)** Make a second scanning of the image,
  assigning to each pixel the new label,
  taking into account the equivalences of the previous step.

Region labelling: example



Binary image



Labelling result
(only the black regions in the binary image were labelled;
very small regions were, after that, elliminated)

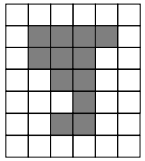
Selection of candidate (marker) regions - 1

- In order to reduce processing time as much as possible,
  it is convenient to eliminate regions that can not correspond
  to the searched marker.
- Several heuristic rules can be applied,
  by rejecting regions that verify one of the following
  conditions:
  - area too small
    - even if it corresponds to the marker,
      that means that it is too far away,
      and so the detection will probably not be accurate
  - aspect ratio of the envolving rectangle is less than a threshold

(In the following, other heuristics that can be used will be presented)

Contour detection

- Border following algorithm
  - How to determine the contour points of the follwing object ?

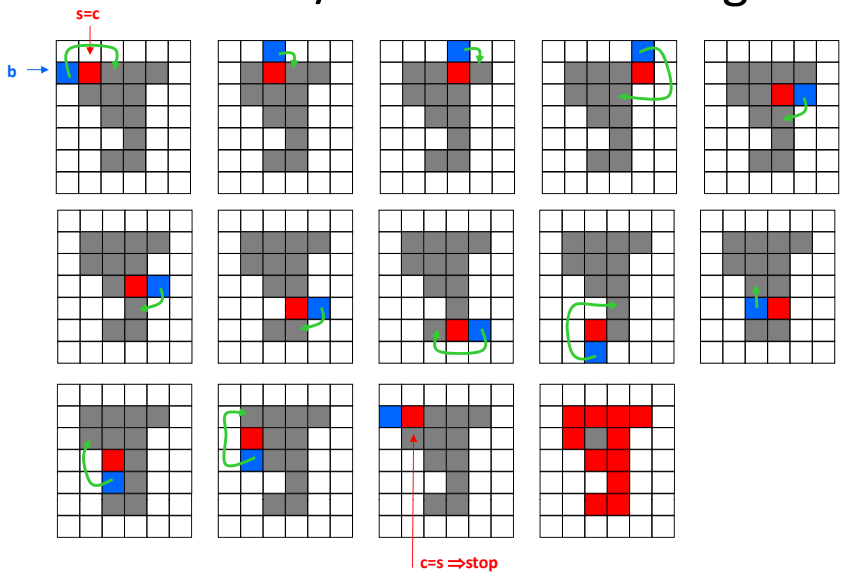


Border/Contour following

- It is necessary to determine the border/contour of the regions previously labelled.
- The border/contour of a region *S*, representing an object, is the set of pixels of *S* that are adjacent to the background.

- Countour following algorithm,
  in clockwise sense,
  considering 8 connectivity (all 8 neighbours are analyzed)
  - Let:
    - *S* – be the set of object pixel
    - *s* – be the first point of the object, detected when the image is scanned, from top to bottom, left to right
    - *b* – be the background pixel that is at the left of *s*
  - 1) let $c = s$
  - 2) considering the 8 neighbours of *c*, starting at *b*, in clockwise sense: $n_1, n_2, n_3, \ldots, n_8$
  - 3) select the first n_i such that n_i belongs to *S*
    - let $c = n_i$
    - let $b = n_{i-1}$
  - 4) repeat steps 2 and 3 until $c = s$

Border/Contour following

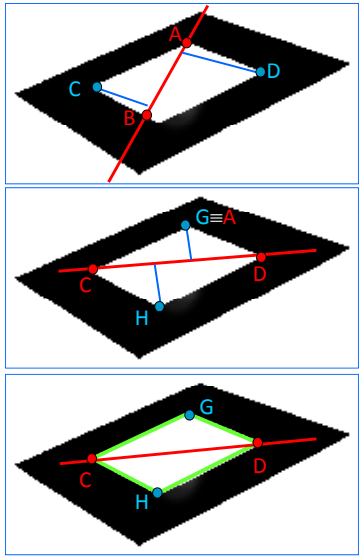


Border/Contour following

- References:
  - http://www.imageprocessingplace.com/downloads_V3/root_downloads/tutorials/contour_tracing_Abeer_George_Ghuneim/alg.html
  - http://docs.opencv.org/modules/imgproc/doc/structural_analysis_and_shape_descriptors.html
    - void **findContours(**InputOutputArray image, OutputArrayOfArrays contours, OutputArray hierarchy, int mode, int method, Point offset=Point())
    - Implements method proposed in
      Suzuki, S. and Abe, K., Topological Structural Analysis of Digitized Binary Images by Border Following. CVGIP 30 1, pp 32-46 (1985)
  - https://github.com/opencv/opencv/blob/master/modules/imgproc/src/contours.cpp

Corner detection & Quadrilateral fitting

- Corner detection:
  algorithm based on polygonal approximation:
  - select 2 points on the contour, separated by approximately half of the contour length: A e B
  - trace the line that passes trough A and B: line A-B
  - for each of the obtained contour halves, determine the contour point that is most distant from line A-B: C and D
  - trace the line that passes trough C and D: line C-D
  - for each of the obtained contour halves, determine the contour point that is most distant from line C-D: G e H
  - trace the line segments: C-G, G-D, D-H e H-C
- Quadrilateral fitting:
  verify that the contour points are close enough to each of the line segments, so that the polygon can be considered a quadrilateral
  - ~ equivalent to specify that the number of corners must be 4

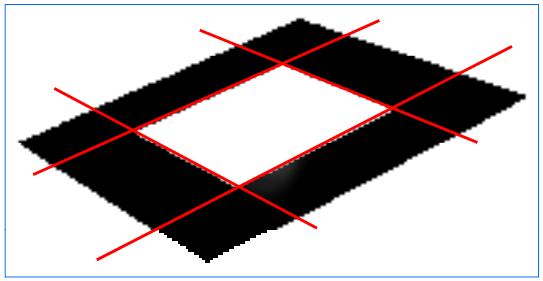


Selection of candidate (marker) regions - 2

- After corner detection and quadrilateral fitting another heuristic rule can be used to reject quadrilaterals that can not correspond to the marker:
  - the opposite sides of the quadrilateral must be approximately parallel

Corner detection: refinement

- The position of the corners can be detected more accurately (*subpixel accuracy*) in the following way :
  - fit a line to the contour points on each one of the sides of the quadrilateral,
    - do not consider the points of the contour that are closer to the corners
  - intersect the 4 lines to obtain the corners with subpixel accuracy



Marker identification & orientation determination

Marker identification & orientation determination

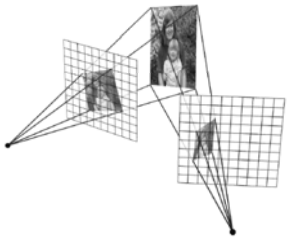
- After the 4 corners of a region potentially corresponding to a marker have been detected in the acquired image it is necessary to verify whether the region effectively corresponds to a marker.
- For that, it is necessary to correct the perspective distortion so that the region can be compared with stored markers (in frontal pose, and usually in 4 possible rotations).
  - This perspective correction operation is sometimes named *unwarping*.
  - We will also call it *normalization* or *perspective correction*.

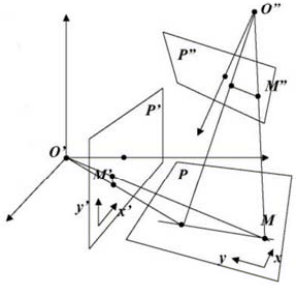
Marker identification & orientation determination

- Steps:
  - marker normalization
    - calculate the homography that relates the 4 corners of the possible marker, detected in the image, to the 4 corners of a marker in frontal view
    - apply homography(ies) to the detected marker(s)
      => geometric image transformation
  - comparison with markers in the database
    - markers are usualy stored in 4 possible rotations
    - XOR (normalized marker, stored marker)
  - for each marker, save the correspondences between the corners of the marker in the image and the corners of the stored marker

Marker normalization / Perspective correction: homographies between two views

- Homography between a 3D plane and its image
- Homography between 2 images of the same 3D plane



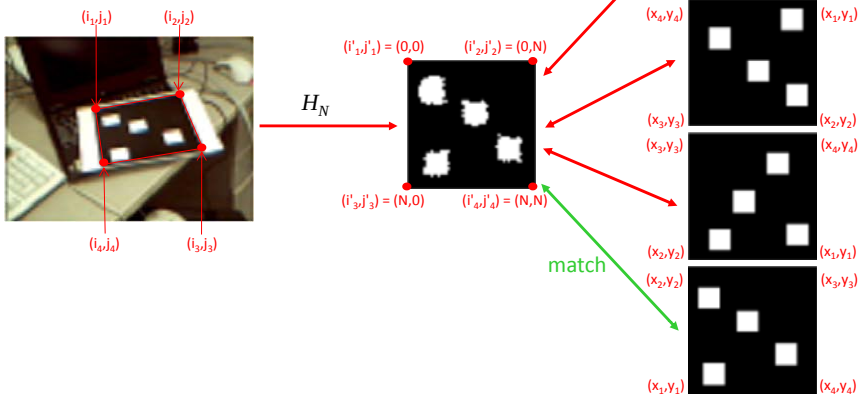


$$\left.\begin{aligned} x_i^{'} &= H^{'} x_i \\ x_i^{''} &= H^{''} x_i \end{aligned}\right| \longrightarrow x_i^{''} = H^{''} H^{'\,-1} x_i^{'} = H x_i^{'}$$

Marker normalization / Marker identification

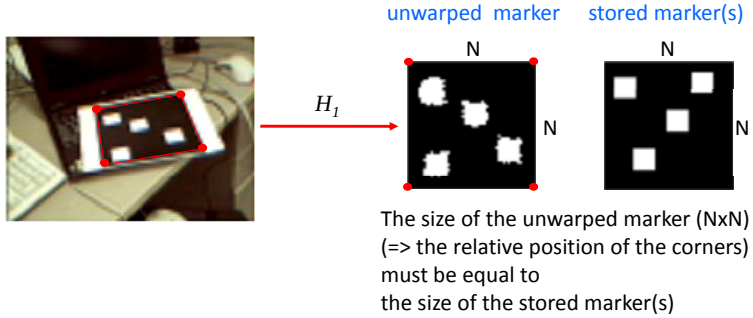
homography between 2 images:
• selected region and
• NxN quadrilateral, in frontal view

$$\begin{bmatrix} wi' \\ wj' \\ w \end{bmatrix} = [H_N] \cdot \begin{bmatrix} i \\ j \\ 1 \end{bmatrix}$$



Marker normalization: geometric transformation

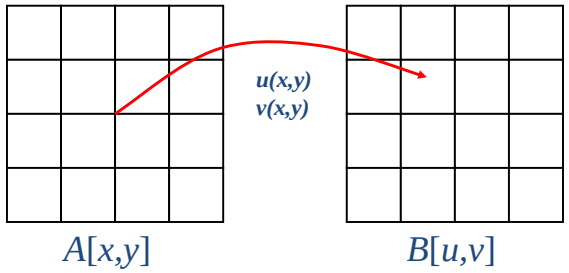
- How to transform the acquired image, to generate a frontal view of the marker that can be compared with stored markers ?



The size of the unwarped marker (NxN) (=> the relative position of the corners) must be equal to the size of the stored marker(s)

Geometric image transformations

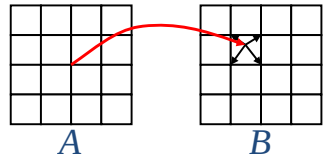
Forward mapping



- ***Note***:
  the pixel position is, in this case, indicated by the grid intersections

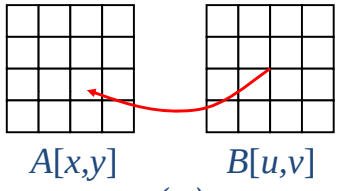
Forward mapping: problems

- The coordinates of the transformated point may fall outside of the image
- The transformed coordinates can be non-integer values
  - solution:
    "to spread" the value of each *pixel*

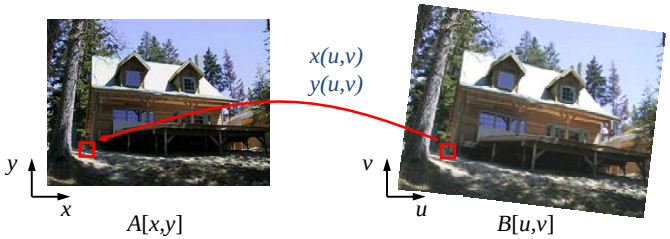


Backward mapping

- To reduce the computation time, guaranteeing that each pixel of the resulting image has an assigned value, a backward mapping is usually used to determine the position of the resulting pixel in the original image.

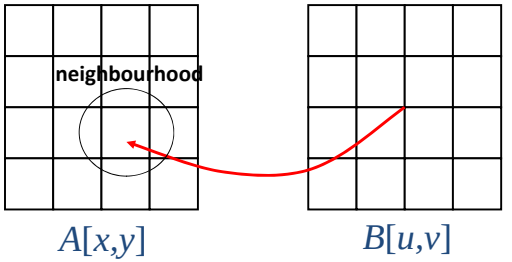


u(x,y)
v(x,y)



Backward mapping: problems

- The coordinates of the point in the original image rarely are integer values.
  - Solution: interpolation
  - The mapping from A[x,y] to B[u,v] must be invertible.

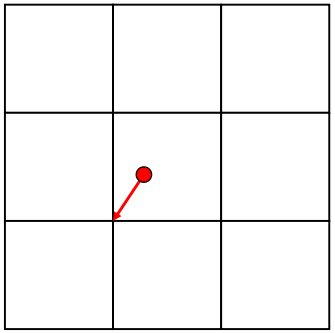


Interpolation

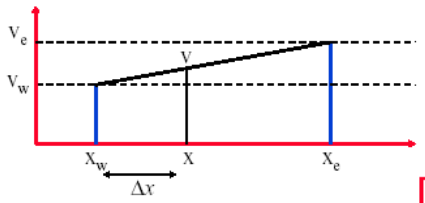
- Objective:
  - to generate a new *pixel* through the analysis of neighbour *pixels*
- A function of the closest neighbours (or a larger neighbourhood)
- Methods:
  - Nearest neighbour
  - Bilinear
  - Bicubic or higher orders

Interpolation: nearest neighbour

- The easiest form of interpolation
- The coordiante values, *x* and *y*, are rounded to the nearest integer
- Problem: the resulting image has not a continuous aspect



Interpolation: linear



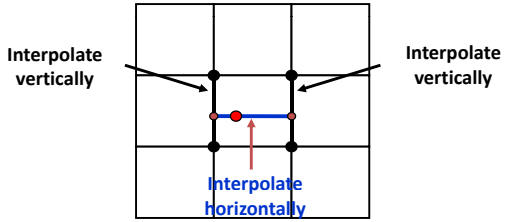
$$\frac{x - x_w}{x_e - x_w} = \frac{v - v_w}{v_e - v_w}$$

$$v = \frac{x - x_w}{x_e - x_w} v_e + \frac{x_e - x}{x_e - x_w} v_w$$

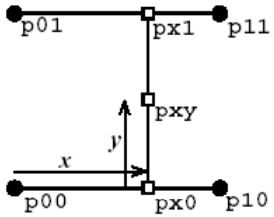
$$v = \Delta_x v_e + (1 - \Delta_x) v_w$$

Interpolation: bilinear

- Interpolate linearly in one direction (ex: vertically)
- Interpolate linearly the results of the previous operation in the other direction (ex: horizontally)



Interpolation: bilinear



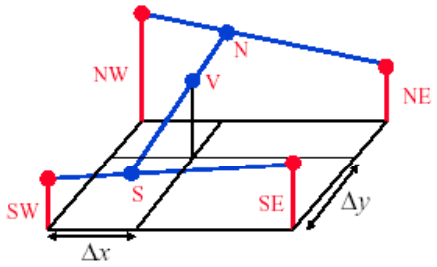

```
pxy = (1-x)*(1-y)*p00 + x*(1-y)*p10
    + (1-x)*y*p01 + x*y*p11
```

or

```
px0 = p00 + x*(p10-p00)
px1 = p01 + x*(p11-p01)
pxy = px0 + y*(px1-px0)
```

← Reduces the no. of products from 8 to 3

Interpolation: bilinear



$$S = SE \cdot \Delta x + SW \cdot (1 - \Delta x)$$
$$N = NE \cdot \Delta x + NW \cdot (1 - \Delta x)$$
$$V = N \cdot \Delta y + S \cdot (1 - \Delta y)$$

Interpolation: example

Interpolation: higher orders

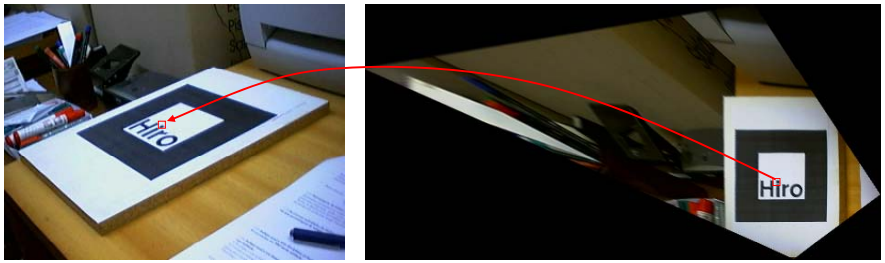
- Higher degree polynomials:
  - ex: bicubic(cubic interpolation in one direction followed by cubic interpolation in the other direction)
- Requires larger neighbourhoods
  - ex: bicubic requires a neighbourhood size of 4x4
- More heavy computacionally

Geometric image transformations

)

Marker normalization / Perspective correction

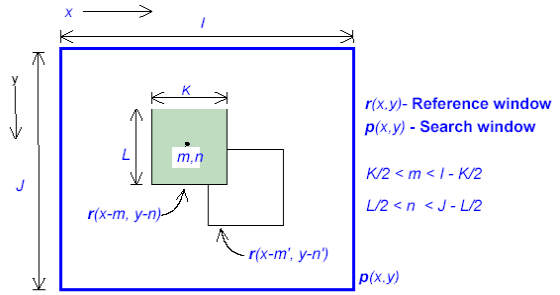
- Generation of a"frontal view" of the reference marker, to be compared to the stored marker
- Alignment (*registration*)
  - Determination of the homography between 2 images (the acquired image and the stored "image" – square in frontal view)
  - Image transformation (*unwarping*)



(Template matching in grayscale images

- Define a reference window (template) and a search window (sometimes, the whole image)
- Define a similarity measure
- For all possible positions of the reference window, calculate the similarity measure
- Choose the position that leads to the highest value of the similarity measure

Searching …



- Problems / questions:
  - the scale of the image may not be known *a priori*
  - the searched "object" must be in identical pose in the search and reference windows
  - which similarity measure to use and what threshold to use for the similarity measure

Similarity measures

Sum of Square Differences

$$SSD(m,n) = \sum_x \sum_y \left(p(x,y) - r(x-m, y-n)\right)^2$$

Sum of Absolute Differences

$$SAD(m,n) = \sum_x \sum_y \left|p(x,y) - r(x-m, y-n)\right|$$

Sum of Absolute Differences Normalized

$$SADN(m,n) = \sum_x \sum_y \left|\left[p(x,y) - \overline{p(x,y)}\right] - \left[r(x-m, y-n) - \bar{r}\right]\right|$$

- Properties:
  - Perfect match: SSD=0, SAD=0, SADN=0
  - SSD and SAD are sensitive to illumination variations
  - All the measures are sensitive to contrast variations

Similarity measures

Correlation

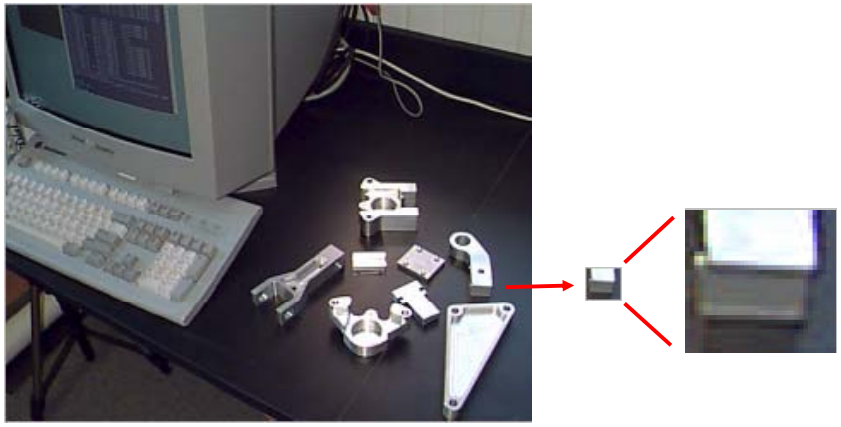
$$C(m,n) = \sum_x \sum_y p(x,y)\, r(x-m, y-n)$$

Normalized Cross-correlation

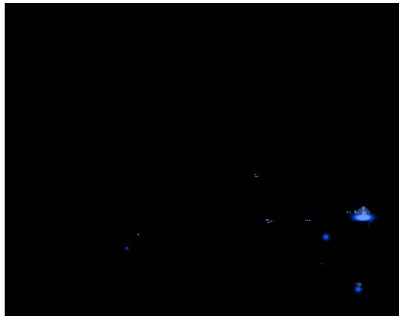
$$CN(m,n) = \frac{\sum_x \sum_y p(x,y)\, r(x-m, y-n)}{\left(\sum_x \sum_y p(x,y)^2\right)^{1/2} \left(\sum_x \sum_y r(x-m, y-n)^2\right)^{1/2}}$$

- Properties:
  - Perfect match: C maximum, CN=1

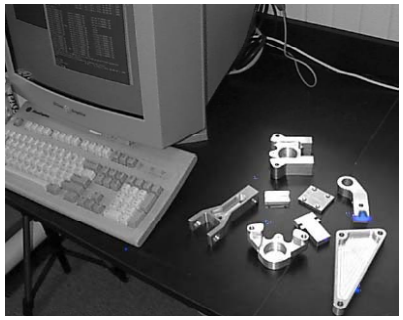
Similarity measures



Similarity measures)



Correlation results (pseudo-color); highest intensity corresponds to highest correlation value



Correlation result superimposed to the original image

AR marker identification

- Taking into account that the candidate region and the stored marker(s) are binary images the folowing similarity measure can be used:

$$C = \sum_{\text{every point}} I_R(i,j) \oplus I_N(i,j)$$

where $\oplus$ represent the operator EXCLUSIVE-OR.

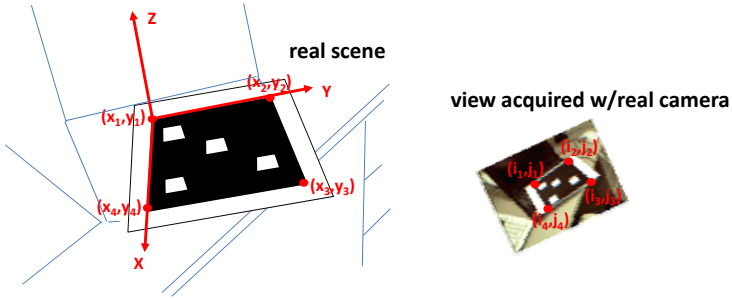
- The less the value of *C* the better is the matching.
- A threshold for the value of *C*, above which the matching is rejected must be selected.

- The dimensions of the stored reference markers I_R are frequently small (just 16x16, in ARToolkit).

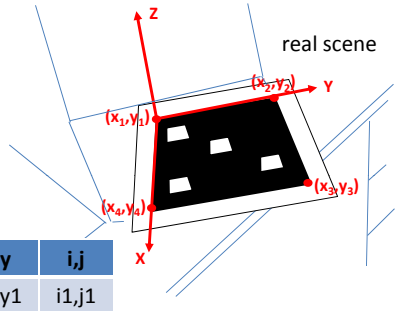
Pose estimation of the real camera

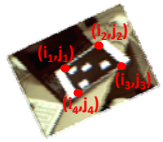
Pose from homography

- As explained previously,
  the pose of the camera used to acquire the image of the scene relatively to the marker,
  can be obtained from the homography between the 4 corners of the marker, in (x,y) coordinates,
  and their position in the image, in (i,j) coordinates



Pose from homography





x,y	i,j
x1,y1	i1,j1
x2,y2	i2,j2
x3,y3	i3,j3
x4,y4	i4,j4

compute homography → $[H2]=\begin{bmatrix} H_{11} & H_{12} & H_{13} \\ H_{21} & H_{22} & H_{23} \\ H_{31} & H_{32} & H_{33} \end{bmatrix}$ →

→ compute pose → $[R]=\begin{bmatrix} R_{11} & R_{12} & R_{13} \\ R_{21} & R_{22} & R_{23} \\ R_{31} & R_{32} & R_{33} \end{bmatrix}$ $[T]=\begin{bmatrix} T_X \\ T_Y \\ T_Z \end{bmatrix}$

Notes on pose estimation

- Pose from homography makes a good starting point
  but is numerically sensitive
- To obtain a more stable linear system for computing H2,
  the coordinates of the points (x,y) and (i,j) should be normalized,
  as explained in [1]
- Pose refinement: after computing H2,
  the reprojection error of the keypoints should be minimized,
  based on
  - Newton iteration
  - Levenberg-Marquardt method
  - … (other)

[1] - R. Hartley. In Defense of the Eight-Point Algorithm, PAMI, Vol. 19, pp. 133-135, 1997
(http://ieeexplore.ieee.org/xpl/login.jsp?tp=&arnumber=601246&url=http%3A%2F%2Fieeexplore.ieee.org%2Fiel1%2F34%2F13258%2F00601246)

Virtual image creation
+
Junction of real & virtual images

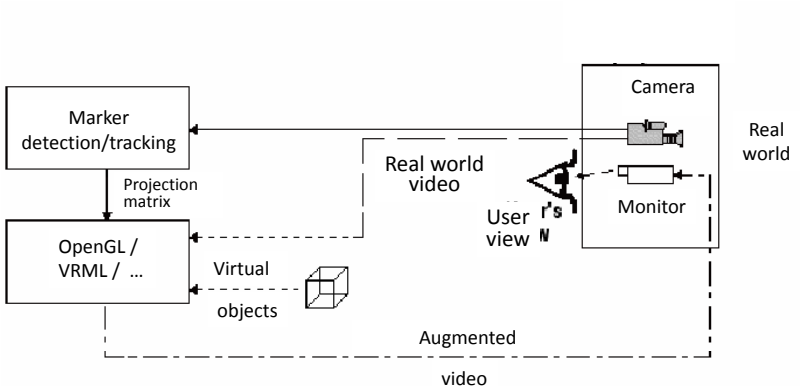
Virtual image rendering

- The first step for rendering the virtual objects is to create a virtual camera, having the same intrinsic and extrinsic parameters as the real camera
- The position and size of the virtual objects must be specified in the (x,y,z) coordinates of the world coordinate system attached to the marker
- In this way, the rendered virtual objects will be aligned/registered with the real objects

Junction of the virtual objects

- The virtual camera used for rendering the virtual objects must have the same intrinsic and extrinsic parameters as the real camera
- Determinação dos parâmetros da câmara virtual
  - intrinsic parameters
    - "focal distances": K_x, K_y; image center: I_0,J_0
    - can be determined previously provided the focal distance of the real camera is kept constant (⇒ no *zoom* lens)
  - extrinsic parameters
    - determined from the homography that relates the (x,y) coordinates of the corners of the marker and the corresponding (i,j) coordinates
- Creation of the virtual image
  - OpenGL
  - VRML
  - …
- Junction of virtual objects to the real image

Junction of the virtual objects



AR Toolkit: creation of the virtual camera

simple.c

```
...
void mainLoop(void)
{
  ...
  /* get the transformation between the marker and the real camera */
   if(mode == 0)  //mode: continuous or one-shot
      arGetTransMat(&marker_info[k], patt_center, patt_width, patt_trans);
    else
      arGetTransMatCont(&marker_info[k], patt_trans, patt_center, patt_width,
                                                    patt_trans);
    draw(patt_trans);
   ...
}

void draw(double patt_trans[3][4])
{
   ...
   /* load the camera transformation matrix */
    argConvGlpara(patt_trans, gl_para);
    glMatrixMode(GL_MODELVIEW);
    glLoadMatrixd(gl_para);
   ...
}
```

http://www.hitl.washington.edu/artoolkit/documentation/index.html

The **arGetTransMatCont** function uses information from the previous image frame to reduce the jittering of the marker. With **arGetTransMat**, only the information from the current image frame is used to compute the position of the marker.
When using the history function, the result will be less accurate because the history information increases performance at the expense of accuracy.

Marker tracking

Tracking

- After the marker is detected and identified it is necessary to track it.
- The reference points used for tracking are the corners of the marker
  - the corners can be easily detected
    and the detection is faily insensitive to changes in illumination
  - for each image it is only necessary to
    analyse a small region around the corner position in th previous image
    (as long as the motion of the marker in the image is not too fast)
- As the marker or the camera moves,
  the homography *Hp*
  (between the 3D plane of the marker and the image plane)
  is recalculated, and
  from *Hp*, the new pose of the camera is recalculated.
- It would be convenient to deal with
  the missing of some corners of the marker
  - due to occclusions caused by other objects
  - or when some corners are out of the field of view of the camera

Tracking

- The tracking phase has
  4 main steps:
  - prediction of the position of the corners
  - detection of the corners
  - updating of the homography H_P
  - updating of the extrinsic parameters of the camera

- NOTE: the updating of homography H_P can be done iteratively
  once the initial homography 3D-2D and
  the homographies 2D-2D between the images of the image sequence are known
  (v. "Markerless Tracking using Planar Stuctures in the Scene", G. Simon, A. Fitzgibbon, A. Zisserman)

- When tracking fails it is necessary to go back to
  the marker detection + recognition phase

Predicting corner position

Predicting corner position

- One possible algorithm:
  - Search for the position of the corners inside a window placed near the position of each corner in the previous image, after correcting the position taking into account the predicted motion of the marker

$$\hat{p}_i = p_i + v_i \Delta t$$

p_i - position of corner *i* in previous image

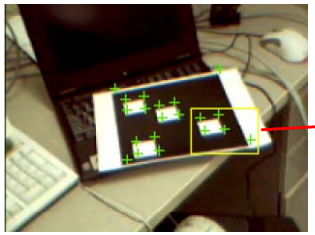
$\hat{p}_i$ - estimated next position

v_i - speed of corner *i*

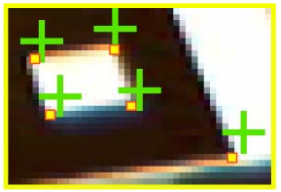
Δt - time passed since last image acquisition

- Dificulty:
  - unpredicted motion of the camera or marker

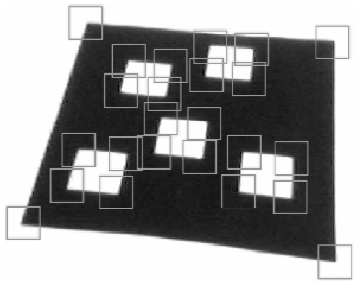
Predicting corner position



zoom



Green marks = predicted positions.
Yellow points = effective corner positions.



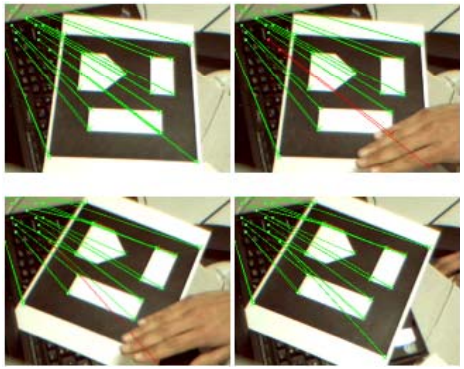
Search window for each corner

Predicting corner position

- Size of the search window
  - should be proportional to the expected displacement of the corners between 2 images acquired consecutively
  - window too small ⇒
    - greater probability of failure in the presence of fast movements
  - window too large ⇒
    - greater probability of detecting several corners inside each window
  - best option: dynamic size
    - the size varies as a function of the distance of the marker to the camera
      - marker distant ⇒ smaller window
    - ex: calculate the size of the window as a function of
      - the area occupied by the elements of the marker
      - or the distance among the corners
  - a great slope of the marker may cause problems (see next slides)

Recovering the corners in occlusion situations

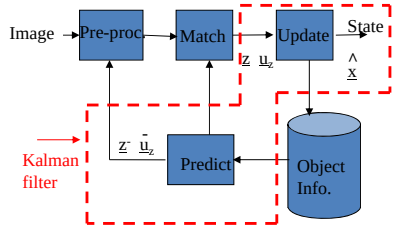
- Using the homography that relates the tracked corners (between 2 acquired images) it is possible to predict where the corners that were temporarily occluded should appear (see the red lines).

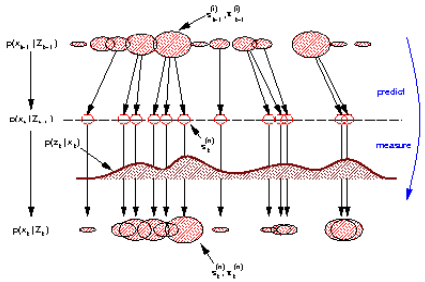


Predicting corner position - alternative methods

- Alternative methods
  - Kalman filter (estimator)
    (http://www.cs.unc.edu/~welch/kalman/)
    - Optimal estimator that infers the parameters of a model from indirect, inexact and uncertain observations/measurements.
    - It is recursive in the sense that the observations/measurements are processed as they are obtained (there is no need to store them).
    - (*v. Kalman filter for dummies*)
  - *CONDENSATION* Algorithm *(*)*
    (*Conditional Density Propagation*)
    (CONDENSATION – Conditional Density Propagation for Visual Tracking, Michael Isard and Andrew Blake, Int. J. Computer Vision, 29, 1, pp. 5-28, 1998)
    - allows the use of more complex models of motion than those usually used in Kalman filters.

    (*) – also known as Particle Filter or Monte-Carlo methods
  - Kanade-Lucas-Tomasi (KLT) *tracker*
    - based on the computation of the optical flow of the image
  - others: simple correlation(-), *mean-shift*, ...



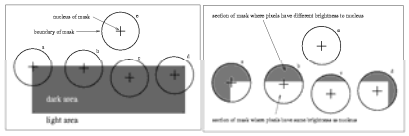


Corner detection

Corner detection: algorithms

- Many algorithms have been proposed

  (http://epubs.surrey.ac.uk/726872/1/Tuytelaars-FGV-2008.pdf)
  - one of the firts:
    Moravec detector
    (http://rfv.insa-lyon.fr/~jolion/PAPIERS/ptint/node2.html)
  - *SUSAN*
    (*Smallest Univalue Segment Assimilating Nucleus*) detector
    (http://www.fmrib.ox.ac.uk/~steve/susan/)
  - Harris / Shi-Tomasi detector
    much used in Computer Vision
    (v. *Introductory Techniques for 3-D Computer Vision, E. Trucco & al.*)
  - ...



a) b)

SUSAN
a) máscaras circulares
colocadas em diversos pontos da imagem
b) *USAN*s são as partes brancas das máscaras
que se assemelham ao núcleo (p.to central);
o valor mínimo ocorre nos cantos

Corner detection: Harris / Shi-Tomasi corner detector

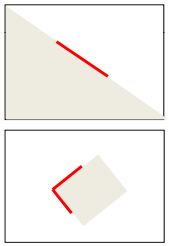
- Basic idea:
  - find points where two edges meet—
    i.e., high gradient in orthogonal directions
- **Harris corner detector**
  - Allows the calculation of the "strength of a corner", λ_p, for each pixel, *p*, of an image window
  - The calculation is based on the gradient inside the window

  - gradient = $[E_x\ E_y]^T$,
    where E_x E_y are the spatial derivatives
    - $E_x = \partial E/\partial x$; $E_y = \partial E/\partial y$
    - calculated using, for example, Sobel filters
  - *p* – image pixel
  - *Q* – neighborhood of (2*N*+1) x (2*N*+1) *pixels*, around *p*
  - *C* – the following matrix, calculated in the neighborhood *Q* of *p*

$$C = \begin{bmatrix} \sum E_x^2 & \sum E_x E_y \\ \sum E_x E_y & \sum E_y^2 \end{bmatrix}$$

Corner detection: Harris / Shi-Tomasi corner detector

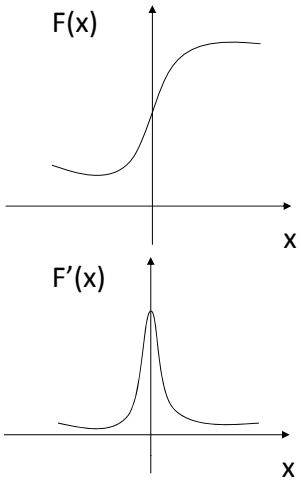
- Matrix *C* caracterizes the graylevel "structure".
- Using the eigenvalues and eigenvectors of *C* it is possible to detect the existence of corners in the windows
  - let λ_1 and λ_2 be the eigenvalues of *C* (both positive)
  - if the intensity of the pixels inside the window is approximately constant, then
    - $\lambda_1 \approx \lambda_2 \approx 0$
  - if the window contains an ideal step edge (transition black↔white) we have:
    - $\lambda_1 > 0$ and $\lambda_2 = 0$
    - eigenvector associated with λ_1 is parallel to the gradient (perpendicular to the edge)
  - if the window contains a corner, then:
    - $\lambda_1 >= \lambda_2 > 0$
    - the greater the λ's, the greater the contrast
    - eigenvectors are parallel to the gradients
- In conclusion:
  - eigenvectors of *C* codify edge orientation
  - eigenvalues of *C* codify edge magnitude



(

Gradient computation

Gradient methods



Edge= sharp variation

Large first derivative

Gradient of a Function

- Assume f is a continuous function in (x,y). Then

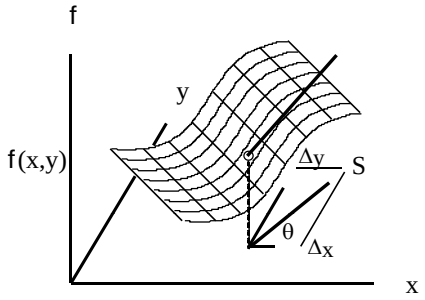
$$\Delta_x = \frac{\partial f}{\partial x}, \quad \Delta_y = \frac{\partial f}{\partial y}$$

- are the rates of change of the function f in the x and y directions, respectively.
- The vector (Δ_x, Δ_y) is called the gradient of f.
- This vector has a magnitude: $s = \sqrt{\Delta_x^2 + \Delta_y^2}$

and an orientation: $\theta = \tan^{-1}\left(\frac{\Delta_y}{\Delta_x}\right)$

- θ is the direction of the maximum change in f.
- S is the size of that change.

Geometric Interpretation

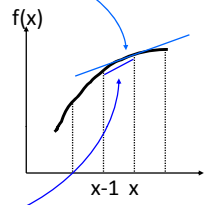


- But
  - I(i,j) is not a continuous function.
- Therefore
  - look for discrete approximations to the gradient.

Discrete approximations

$$\frac{df(x)}{dx} = \lim_{\Delta x \to 0} \frac{f(x+\Delta x) - f(x)}{\Delta x}$$

$$\frac{df(x)}{dx} \cong \frac{f(x) - f(x-1)}{1}$$

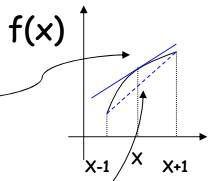


➡ Convolve with

-1	1

Digital approximation

$$\frac{df(x)}{dx} = \lim_{\Delta x \to 0} \frac{f(x+\Delta x) - f(x)}{\Delta x}$$



➡ Convolve with:

-1	1

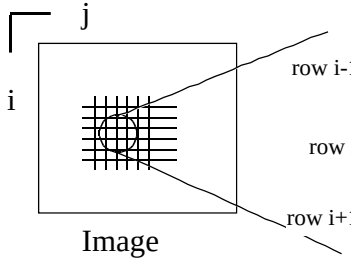
$$\frac{df(x)}{dx} \cong \frac{f(x+1) - f(x-1)}{2}$$

➡ Convolve with:

-1	0	1

In 2 dimensions

- Discrete image function I



	col j-1	col j	col j+1
row i-1	I(i-1,j-1)	I(i-1,j)	I(i-1,j+1)
row i	I(i,j-1)	I(i,j)	I(i,j+1)
row i+1	I(i+1,j-1)	I(i+1,j)	I(i+1,j+1)

- Derivatives ➡ Differences

$\Delta_j I =$

-1	1

$\Delta_i I =$

-1
1

1x2 Example

Horizontal gradient

Vertical gradient

Combined

Smoothing

- Derivatives are sensitive to noise
- Averaging reduces noise
  - spatial averages can be computed using masks

$1/9 \times$

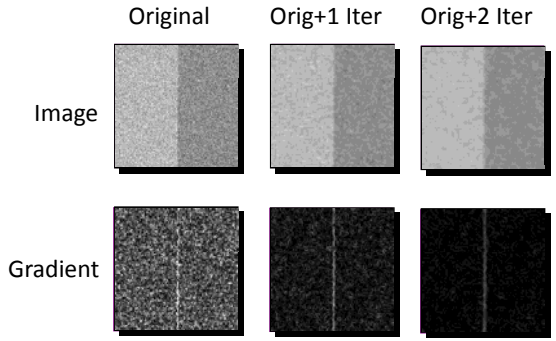
1	1	1
1	1	1
1	1	1

$1/8 \times$

1	1	1
1	0	1
1	1	1

- Solution: combine smoothing with derivative computation

Effect of smooting



Combining the two

- Applying this mask is equivalent to taking the difference of averages on either side of the central pixel.

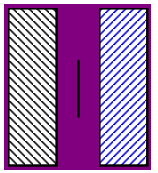
-1	-1	-1
0	●	0
1	1	1

Average

—

Average

-1	0	1
-1	●	1
-1	0	1



Sobel operator

S_1=

-1	-2	-1
0	0	0
1	2	1

S_2 =

-1	0	1
-2	0	2
-1	0	1

Gradient Magnitude = $\sqrt{S_1^2 + S_2^{\ 2}}$

Gradient Direction = $\tan^{-1}\left(\frac{S_1}{S_2}\right)$

)

(

Eigenvalues
and
eigenvectors

Eigenvalues and eigenvectors

- Consider that
  - A is a square matrix
  - v is a column vector
  - λ is a scalar
- If $A.v = \lambda v$
  - assuming a non-trivial solution ($v \neq 0$)
  - v is an *eigenvector* of A
  - λ is an *eigenvalue* of A

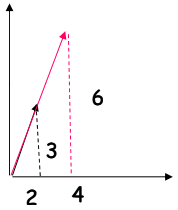
(http://mathworld.wolfram.com/Eigenvalue.html)

Eigenvalues and eigenvectors

- Example:

$$A = \begin{bmatrix} 5 & -2 \\ 6 & -2 \end{bmatrix}$$

$$v = \begin{bmatrix} 2 \\ 3 \end{bmatrix}$$



$$Av = \begin{bmatrix} 5 & -2 \\ 6 & -2 \end{bmatrix} \begin{bmatrix} 2 \\ 3 \end{bmatrix} = \begin{bmatrix} 4 \\ 6 \end{bmatrix} = 2 \begin{bmatrix} 2 \\ 3 \end{bmatrix}$$

v is an eigenvector with eigenvalue 2

Determining the eigenvalues

$$Av = \lambda v$$

$$Av - \lambda v = 0$$

$$(A - \lambda I)v = 0$$

homogeneous system of linear equations

v is an eigenvector if it is not the trivial solution, *v*=0;

according to Cramer's rule,
a system of linear equations has non-trivial solutions
if and only if the determinant is zero

$$det(A - \lambda I) = 0$$

Calculating the eigenvalues

- Exemplo:

$$A = \begin{bmatrix} 5 & -2 \\ 6 & -2 \end{bmatrix}$$

$$\det(A - \lambda I) = \det \begin{vmatrix} 5-\lambda & -2 \\ 6 & -2-\lambda \end{vmatrix} = 0$$

$$(5 - \lambda)(-2 - \lambda) - (6)(-2) = 0$$

$$\lambda^2 - 3\lambda + 2 = 0 \;\Rightarrow \lambda = \frac{3 \pm \sqrt{9 - 8}}{2}$$

$$\lambda_1 = 1, \lambda_2 = 2$$

Determining the eigenvectors

- Eigenvector for λ=1

$$\begin{aligned} (5-1)v_{11} - 2v_{12} &= 0 \\ 6v_{11} + (-2-1)v_{12} &= 0 \end{aligned} \quad \Rightarrow \quad \begin{aligned} 4v_{11} - 2v_{12} &= 0 \\ 6v_{11} - 3v_{12} &= 0 \end{aligned}$$

$$\Rightarrow 2v_{11} = v_{12} \;\Rightarrow\; v_1 = \begin{bmatrix} 1 \\ 2 \end{bmatrix}$$

Determining the eigenvectors

- Eigenvector for λ=2

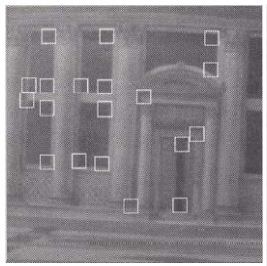
$$\begin{aligned}(5-2)v_{21}-2v_{22} &= 0\\ 6v_{21}+(-2-2)v_{22} &= 0\end{aligned} \quad \Rightarrow \quad \begin{aligned}3v_{21}-2v_{22} &= 0\\ 6v_{21}-4v_{22} &= 0\end{aligned}$$

$$\Rightarrow 3v_{21} = 2v_{22} \Rightarrow \quad v_2 = \begin{bmatrix}2\\3\end{bmatrix}$$

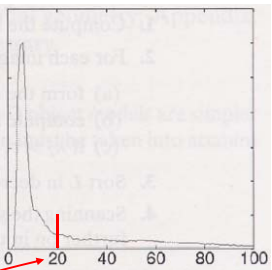
Corner detection: Harris / Shi-Tomasi detector

- A corner is identified by two "strong" edges
- So, when $\lambda_1 >= \lambda_2$, a corner is the point where the smallest eigenvalue, λ_2, is still large enough
  (Shi-Tomasi formulation, based on the detector previously proposed by Harris)
- A threshold, τ, for the value of λ_2 must be established
- This value could be determined by analyzing the histogram λ_2 values

Original image



Neighborhood (15x15) of the points for which λ_2>20



Histogram of λ_2 values for the whole image

Corner detection: Harris / Shi-Tomasi detector

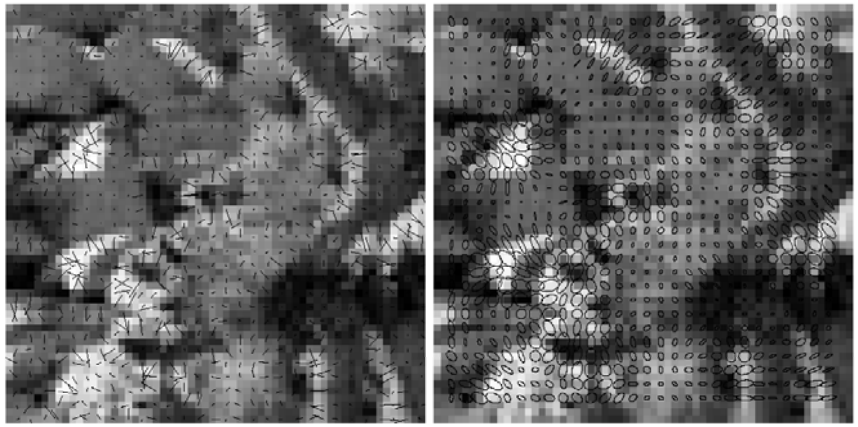
- Algorithm for detecting the corners of an image:
  - Calculate the image gradient
  - For each point of the image, *p*
    - calculate matrix *C* in a neighborhood of *p* - (2*N*+1)x(2*N*+1) points
    - calculate λ_2, the smallest eigenvalue of *C*
    - if $\lambda_2 > \tau$, save the coordinates of *p* in a list *L*
  - Sort *L* by descending order of the λ_2
  - Scan the sorted list, from begin to end, and, for each point, *p*, eliminate all the points that are below it in the list and that are in the neighborhood of *p*
- This algorithm has 2 fundamental parameters:
  - the size *N* of the neighborhood
    - there is no criterium for an optimal choice of *N*
    - values of *N* in the range 2 - 10 are appropriate in many situations (*Trucco's book*)
  - the value of τ
    - this value can be estimated from the histogram of λ_2 values
- For detecting the corners inside a window of the image the above algorithm must be applied to all points inside the window
  - the "strongest" corner will be the one for which λ_2 has the highest value

Harris detector: example

Original image

Image gradient

Harris detector: example



Zooming of a region of the previous image.
The ellipses indicate the eigenvalues and eigenvectors of *C* matrices.

Corner detection(Harris) with *subpixel* accuracy

- For calculating the homographies with the highest possible accuracy the position of the corners must also be accurately detected
- For that, the values of λ_2 calculated in the neighborhood of a corner can be used:
  - let ***s***=(*x*,*y*) be the "strongest" corner in a given neighborhood
  - its position with *subpixel* accuracy can be obtained using the formulas on the right, where λ_i represents the "strength" of a corner (value of λ_2) calculated for *pixel i*

(S. Malik, G. Roth, C. McDonald. "Robust 2D Tracking for Real-time Augmented Reality", in Proceedings of Vision Interface (VI) 2002, Calgary, Alberta, Canada)

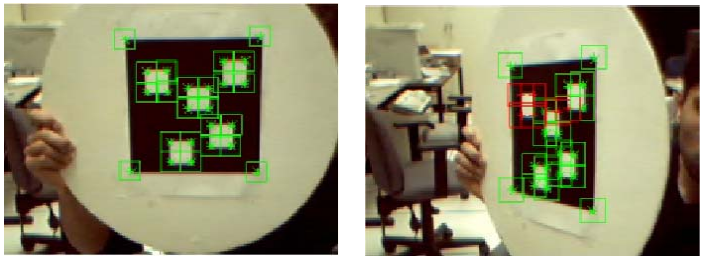
	c	
a	**s**	b
	d	

$$sub\_x = x + 0.5 + \frac{(\lambda_a - \lambda_b)}{2(\lambda_a + \lambda_b - 2\lambda_s)}$$

$$sub\_y = y + 0.5 + \frac{(\lambda_c - \lambda_d)}{2(\lambda_c + \lambda_d - 2\lambda_s)}$$

Corner detection (Harris) final remarks

- When the orientation of the markers is such that the perspective distortion is high, problems can arise in the detection of the corners, not due the the Harris detector but due the window size
  - several corners may be inside each window



green windows – corners successfully tracked
red windows – corners whose tracking failed

- Harris detector produces good results even when the lighting conditions are not the best ones (as long as there is a visible contrast between the black and pixels around the corners)

Robust detectors/descriptors

- Robust
  - may be used to identify (more os less) uniquely feature points (/ interest points), even in adverse conditions, such as:
    - variations in the size of the objects (image scale)
    - rotation and orientation / slope
    - variations in the illumination
    - noise
  - they generate feature point "descriptors" that may be used in matching
- Exampless:
  - SIFT- Scale Invariant Feature Transform, Lowe (1999)
  - SURF - Speeded Up Robust Features, Bay et al. (2006)
  - FAST- Features from Accelerated Segment Test, Rosten et al. (2005)
  - ... GLOH (2005), LESH (2008), ...
- Dificulty:
  - computationally "heavy" (GPU implementations start being available)

Robust computation of homographies

Homography computation

- The computation of homographies must be robust enough to deal with
  - errors in the detection of some corners
  - wrong matches
  - occluded corners
- For that, the *RANSAC* (*RANdom SAmple Consensus*) may be used

(*M. A. Fischler, R. C. Bolles. Random Sample Consensus: A Paradigm for Model Fitting with Applications to Image Analysis and Automated Cartography. Comm. of the ACM, Vol 24, pp 381-395, 1981*)
(*RANSAC for dummies, http://vision.ece.ucsb.edu/~zuliani/Research/RANSAC/docs/RANSAC4Dummies.pdf*)

  - this method allows a robust model fitting in the presence of many *outliers.*

The *RANSAC* algorithm

- The problem:
  - given a fitting problem with parameters $[x_1..x_n]$, estimate the parameters values
- Assume that:
  - the parametersmay be estimated from *N* data values
  - there are a total of *M* data values
- The algorithm:
  - 1) Select, randomly, *N* samples
  - 2) Estimate the parameters $[x_1..x_n]$ (obtain a model)
  - 3) Determine how many of the *M* samples fit the obtained model, given a tolerance; let *K* be the number of samples
  - 4) If *K* is large enough, accept the model parameters and end with SUCESS
  - 5) Repeat 1) to 4), *L* times
  - 6) If this step is reached, end with FAILURE
- L is difficult to be determined... Formule for determining *L :*

$$L = \frac{\log(p_{fail})}{\log(1 - p_g^N)}$$

p_{fail} - probability of failing the selection of N *inliers* during *L* repetitions

p_g - probability of finding a good sample; in general, it is unknown at the beginning, being updated along the iterations

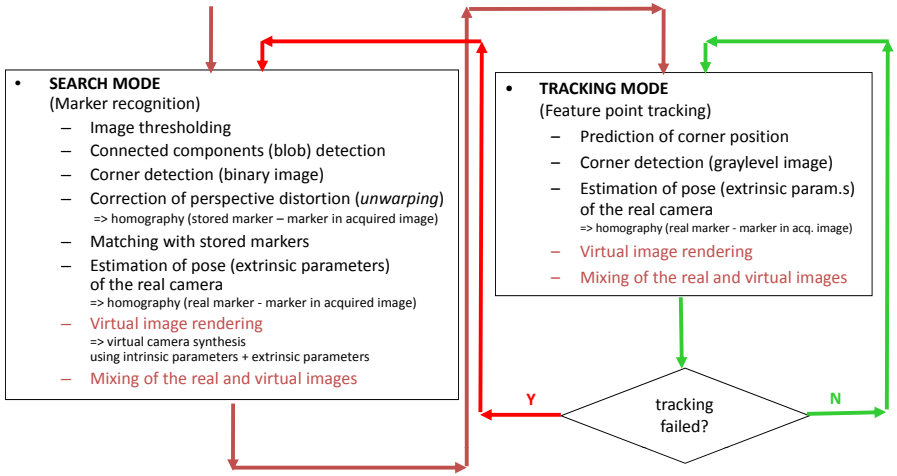
Homography calculation using *RANSAC*

- Algorithm:
  1. select, randomly, 4 non-collinear, matched points in two images of the same marker or in the marker and in an acquired image of the marker: $\{ [x\ y\ 1]^T \leftrightarrow [x'\ y'\ 1]^T \}$
  2. calculate the homography, H, using this random sample
  3. for all the matched points, calculate the distance between $[x'\ y'\ 1]^T$ and $[H]\ [x\ y\ 1]^T$
  4. count the number of matched pairs for which the distance is below a threshold; these pairs are considered as *inliers* (I_H=no. of *inliers*) and the remaining pairs are considered as *outliers*
  5. a maximum number of iterations was reached STOP and EXIT with FAILURE
  6. if the number of *outliers* is very high or the ratio *outliers/inliers* is above a threshold, go to step 1,
  7. recalculate the homography using the *inliers*

Final remarks

Marker-based AR System

Block diagram



What is missing

for a realistic integration
of virtual objects in real world images ?

What is missing…

- **Markerless tracking**
  - fairly more complex than *marker-based tracking*
  - see next slides + several papers in RVAu Moodle page
- Interaction with users
  - virtual objects manipulation,
    using special devices, gestures, etc…
  - collision detection
    (between real and virtual objects or between virtual objects)
  - force feedback, audio feedback, etc…
- Deal with occlusion btween real and virtual objects
  - an exact 3D model of the real world may be needed
- Virtual lighting
  - presently, in most cases, virtual obejcts are rendered using pre-set lighting
  - para uma visualização realista é necessário recriar a iluminação real
    - extrair a posição e características das fontes de luz
      a partir das imagens adquiridas
    - em tempo real …