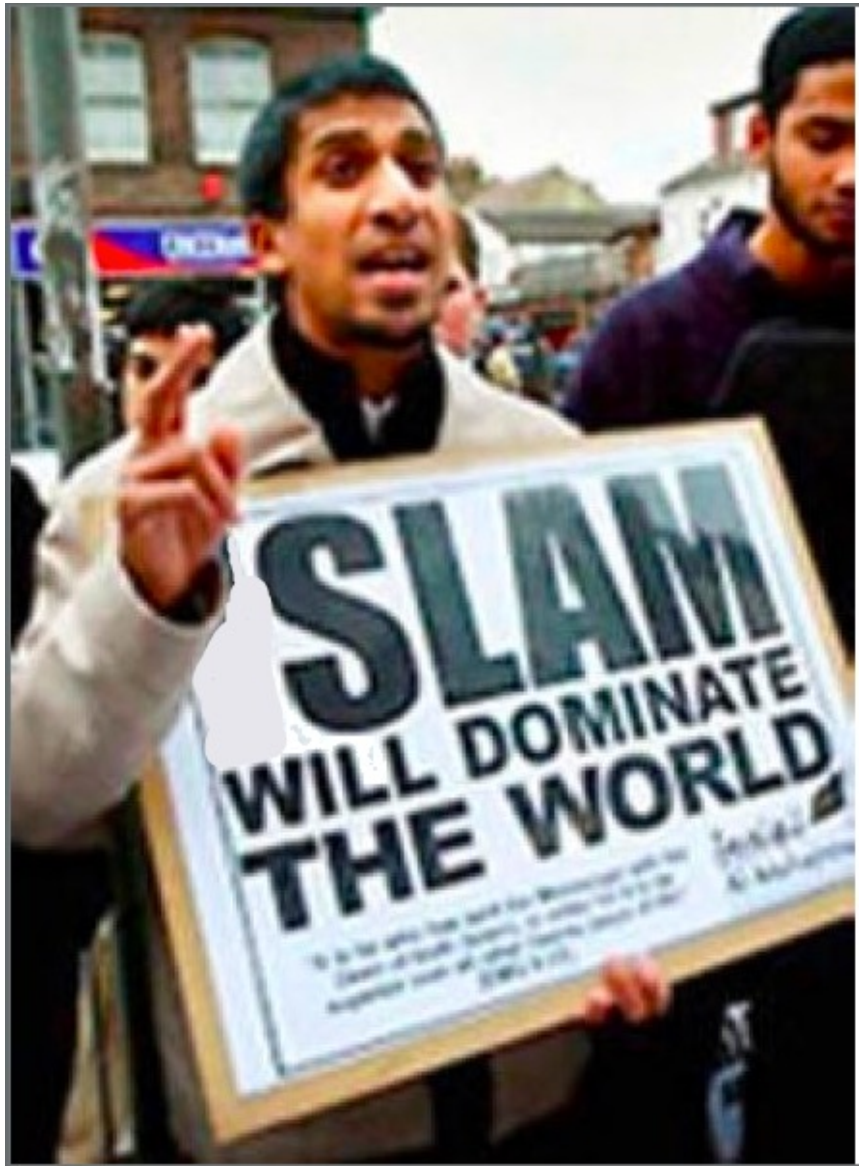
VAN course
Lesson 2

Dr. Refael Vivanti
vivanti@gmail.com

Rafi’s intro

- Why SLAM?
  - Cool stuff
    - Accurate stuff!
  - Unique niche
  - Industry:
    - Homeland security
    - AR/VR
    - Engineering (drones, mines, ports exc..)
    - Companies in Israel:
      - Rafael, Elbit, Meta, Mobileye, Magic leap, Ception, Shopic, 6DOF, 643.ai …
- Why this course?
  - Project you can put in your CV
  - David is a very good lecturer



Courtesy of AsiaNews

Image Features - intro

- What is so Deep in Deep Learning?
  - Yann LeCun prophecy (2012):
  - ”In five years, you’ll all use …
    Feature Learning”

Courtesy of historyofdatascience.com

- Features are a key concept in Computer Vision:
  - Some small patch gives a lot of info

- Research history:

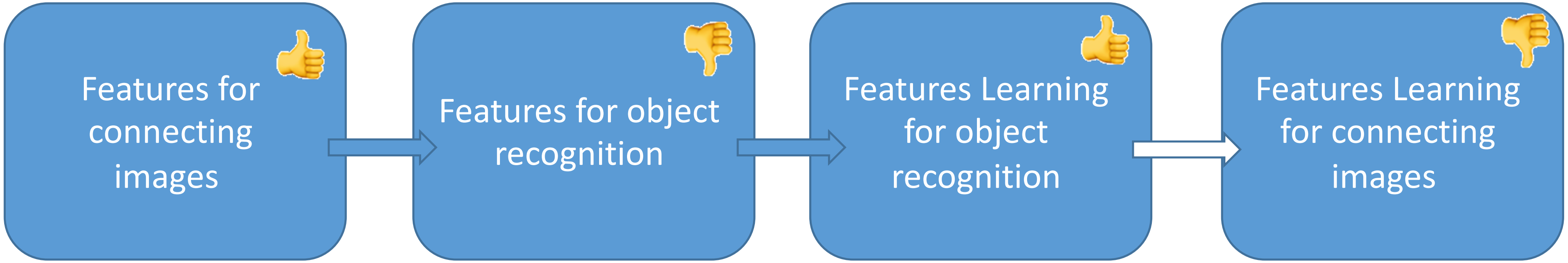
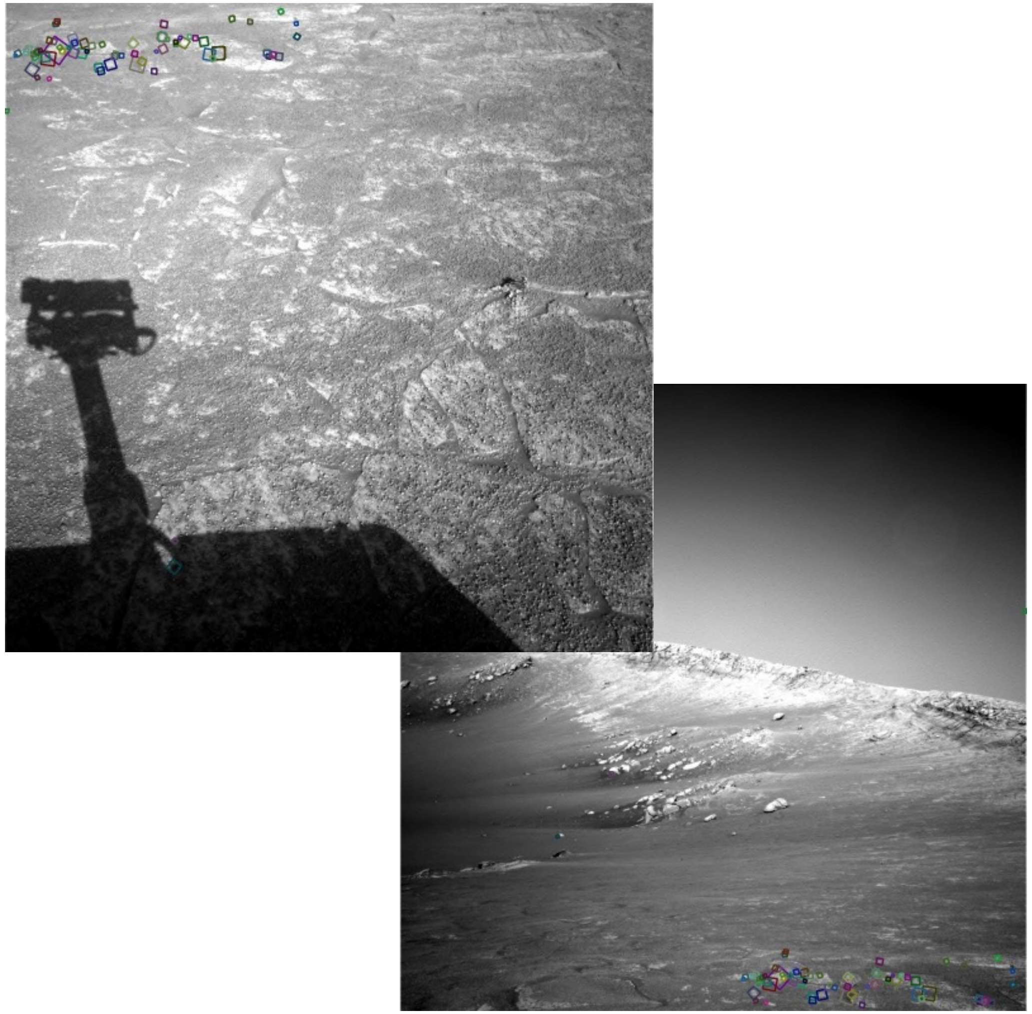


Image Features - intro

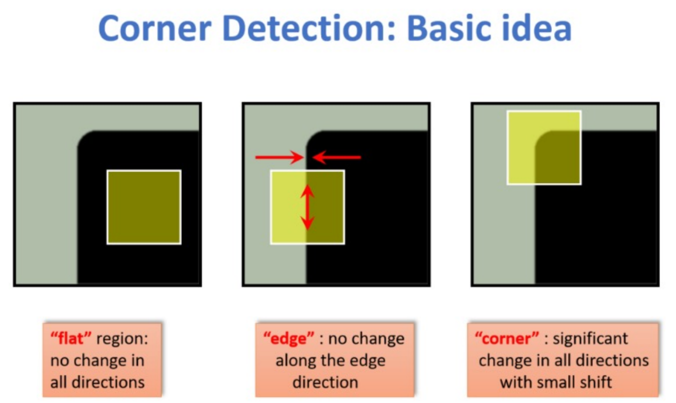
- Features: Locations with texture in image
  - which are easy to recognize in other images
- What makes a good feature?
  - Unique – distinguished from other locations
  - Invariant to changes – color, perspective, noise
  - It’s a tradeoff!
- For feature matching we need:
  - Feature extraction method
  - Feature description
  - Feature matching
  - They are correlated



NASA Mars Rover images with SIFT feature matches
Figure by Noah Snavely

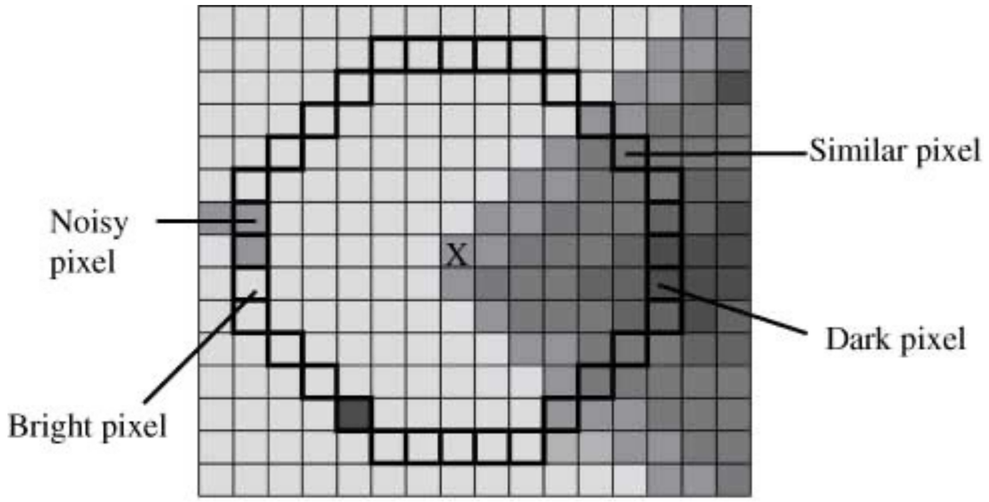
Feature extraction

- Extraction/detection: finding the best locations
  - Locally unique : outstanding from local environment
    - Has strong gradients in both directions
  - Globally unique : has low prevalence
- Types:
  - Harris corner detection
    - Using second moments matrix
  - FAST
    - Counting radius pixels
  - DOG
    - Scale invariant
  - Many more
  - Very correlated with chosen descriptor



Harris

Courtesy of datahacker.rs

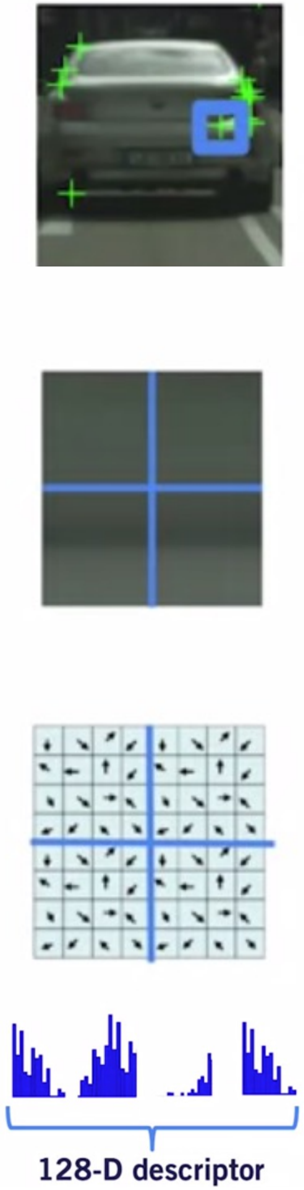


FAST

Courtesy of Yuanxiu Xing

Feature descriptor

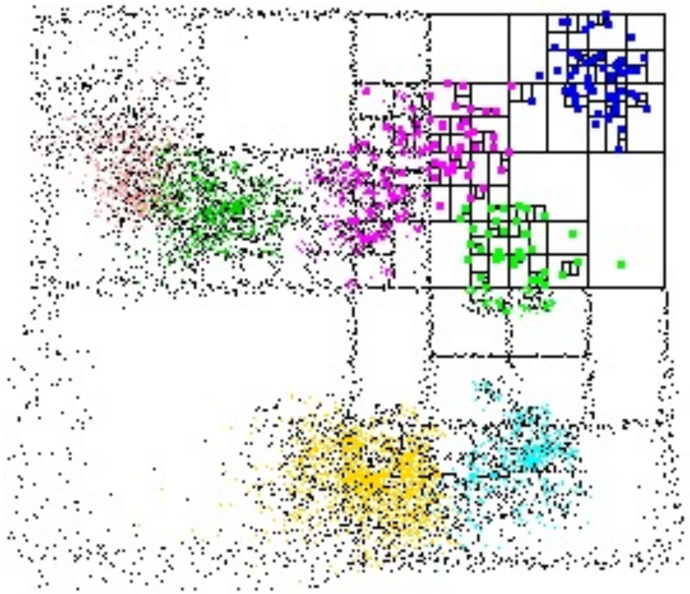
- A vector which represent the feature environment
- Desired features:
  - Repeatability - Similar for same location
    - Invariant to color and perspective changes
  - Distinctive - different locations should be distictable
  - Compact – fast to use
- Types:
  - Naïve – a local patch
  - Gradients based - SIFT, SURF, GLOH, HOG, KAZE, A-KAZE
  - Binary – BRISK, BRIEF, ORB
  - Deep learning – SuperPoint, D2-Net, LF-Net



SIFT descriptor

Feature Matching

- Finding couples of locations
  - Using their description
- Components:
  - Metric – distance between descriptors
    - L_1, L_2, L_∞ , Hamming, cosine, …
  - Matching algorithm
    - Brute force, Approximate Nearest Neighbor
  - Regularization
    - Matching grade threshold
    - Cross matching
    - Significance threshold



Approximate Nearest Neighbor

Courtesy of David M. Mount and Sunil Arya

Open-CV feature matching example

```
algorithm = cv2.KAZE_create()

# Extraction:
Key_points = algorithm.detect(image)

# Descriptors:
kps, dsc = alg.compute(image, kps)

# Matching:
brute_force = cv2.BFMatcher(cv2.NORM_HAMMING, crossCheck=True)
matches = bf.match(des1,des2)
```

Why Homogeneous Coordinates?

- Common transformations are affine but not linear
  - But are linear in Homogeneous Coordinates
- Allows us use matrix multiplication to calculate transformations – extremely efficient!

Homogeneous Coordinates

- A point (x, y) can be re-written in homogeneous coordinates as (x_h, y_h, h)
- The homogeneous parameter h is a non-zero value such that:

$$x = \frac{x_h}{h} \qquad y = \frac{y_h}{h}$$

- We can then write any point (x, y) as (hx, hy, h)
- We can conveniently choose $h = 1$ so that (x, y) becomes $(x, y, 1)$
- $(x, y, 1) = (5x, 5y, 5) = (hx, hy, h) \neq (0x, 0y, 0)$
    - This removes one DOF, hence it is still a 2D representation

Homography

- Combine the geometric transformation into a single matrix with 3x3 matrices
- Two-Dimensional translation matrix:

$$\begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & t_x \\ 0 & 1 & t_y \\ 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$$

$$x' = 1x + 0y + 1t_x = x + t_x$$

$$y' = 0x + 1y + 1t_y = y + t_y$$

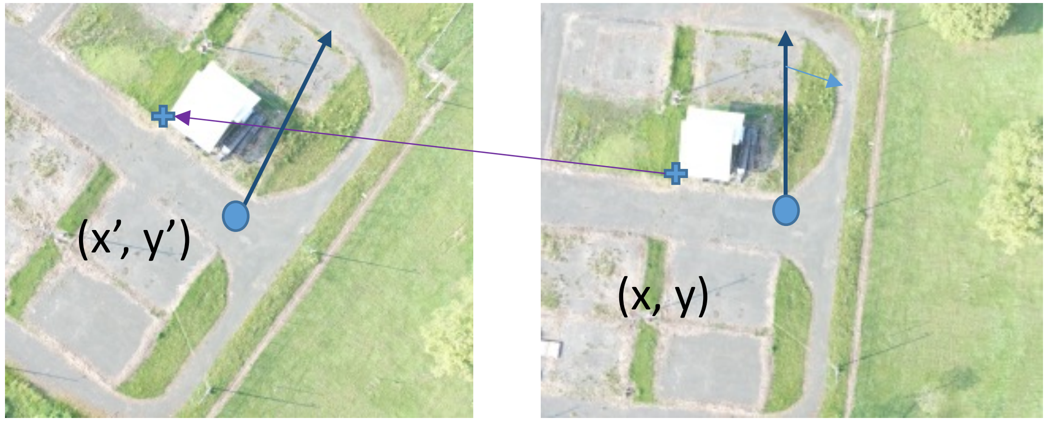
$$h = 0x + 0y + 1 = 1$$

$$\begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix} = \begin{bmatrix} x + t_x \\ y + t_y \\ 1 \end{bmatrix}$$

Homography

- Two-Dimensional rotation matrix

$$\begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix} = \begin{bmatrix} \cos\theta & -\sin\theta & 0 \\ \sin\theta & \cos\theta & 0 \\ 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$$



$$x' = x\cos\theta - y\sin\theta + 1\cdot 0 = x\cos\theta - y\sin\theta$$

$$y' = x\sin\theta + y\cos\theta + 1\cdot 0 = x\sin\theta + y\cos\theta$$

$$h = 0x + 0y + 1 = 1$$

Example: $\theta = 90^0$

$$x' = x\cos 90^0 - y\sin 90^0 + 1\cdot 0 = -y$$

$$y' = x\sin 90^0 + y\cos 90^0 + 1\cdot 0 = x$$

$$h = 0x + 0y + 1 = 1$$

Homography

- Two-Dimensional **scaling** matrix

$$\begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix} = \begin{bmatrix} s_x & 0 & 0 \\ 0 & s_y & 0 \\ 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$$

$$x' = s_x x + 0y + 1 \cdot 0 = s_x x$$

$$y' = 0x + s_y y + 1 \cdot 0 = s_y y$$

$$h = 0x + 0y + 1 = 1$$

$$\begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix} = \begin{bmatrix} s_x x \\ s_y y \\ 1 \end{bmatrix}$$

Homography

- Linear transformation - a combination of:

Scale, Rotation and Translation transformations

$$\begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix} = \begin{bmatrix} s_x & 0 & 0 \\ 0 & s_y & 0 \\ 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} \cos\theta & -\sin\theta & 0 \\ \sin\theta & \cos\theta & 0 \\ 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} 1 & 0 & t_x \\ 0 & 1 & t_y \\ 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$$

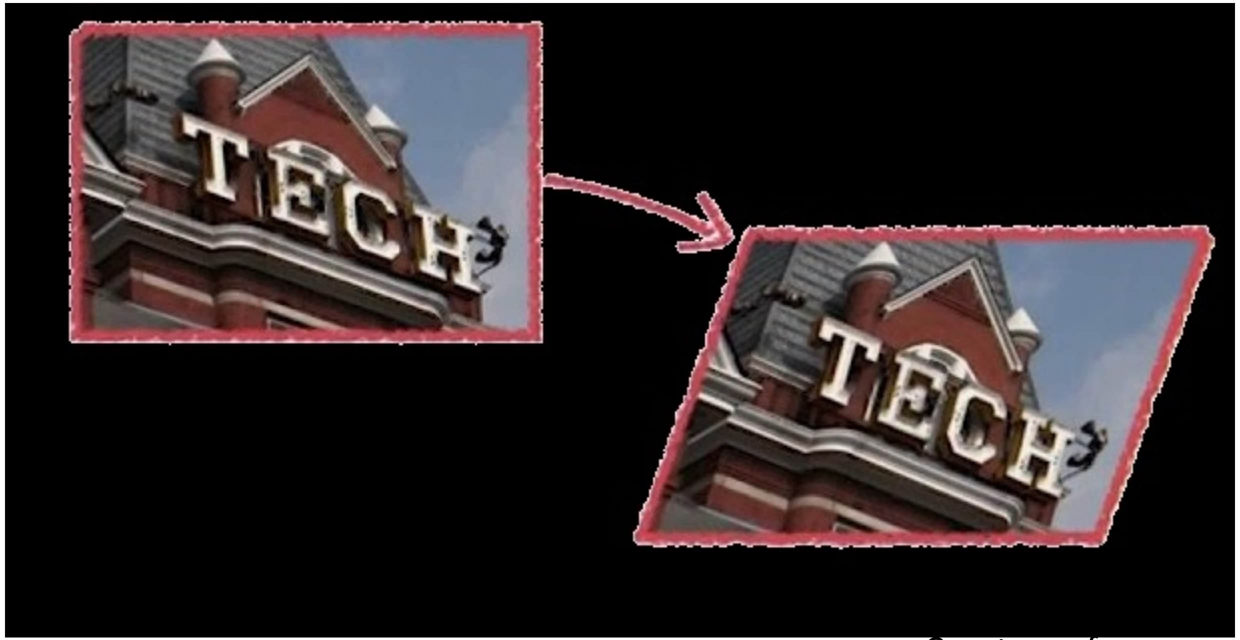
- Also called “similarity”
- 5 DOF: S_x, S_y, Θ, t_x, t_y

Homography

- Affine Transformation:
- 6 DOF: a, b, c, d, e, f

$$\begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix} = \begin{bmatrix} a & b & c \\ d & e & f \\ 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$$

- First transformation to change angles!



Courtesy of coursera

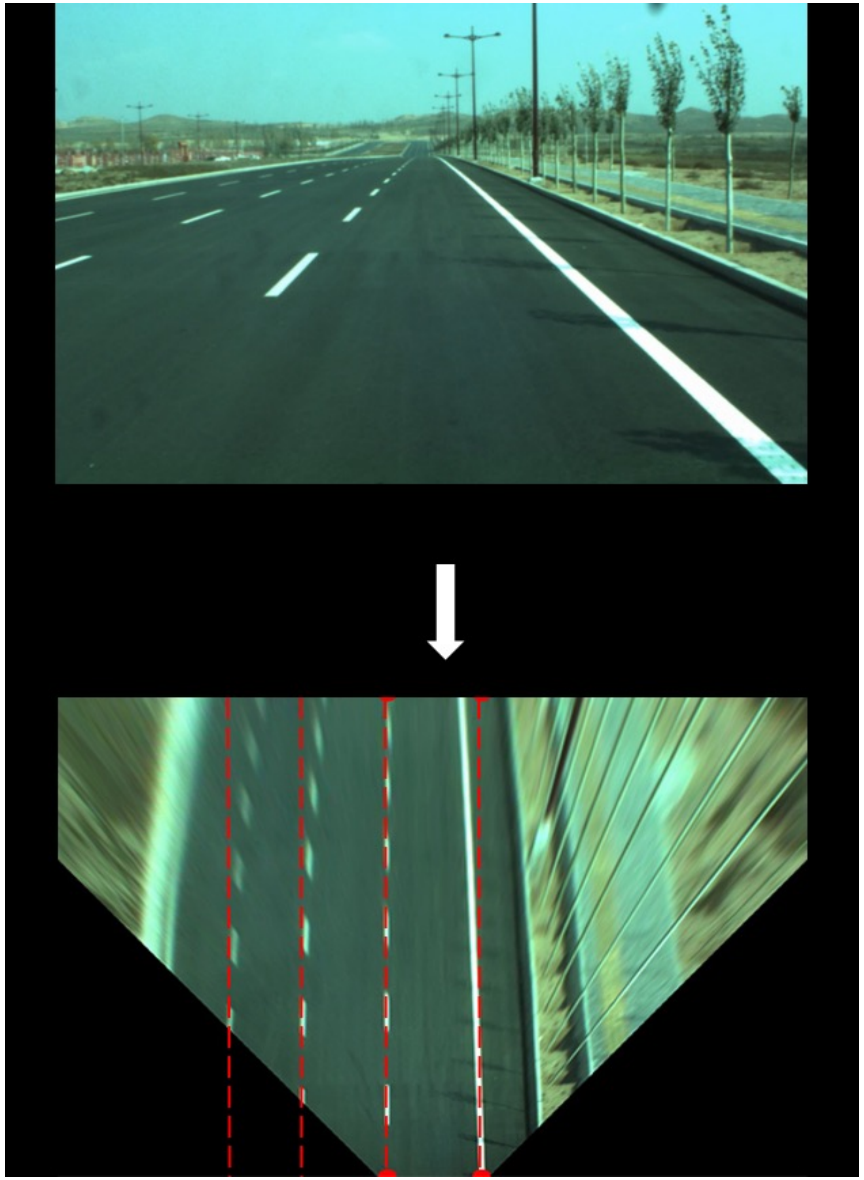
- Turns squares into parallelogram
- Any affine trans is equal to:
  - Rotation -> uneven scale -> another rotation -> translation

Homography

- Perspective Transformation:
- 8 DOF: a, b, c, d, e, f, h, g

$$\begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix} = \begin{bmatrix} a & b & c \\ d & e & f \\ h & g & 1 \end{bmatrix} \cdot \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$$

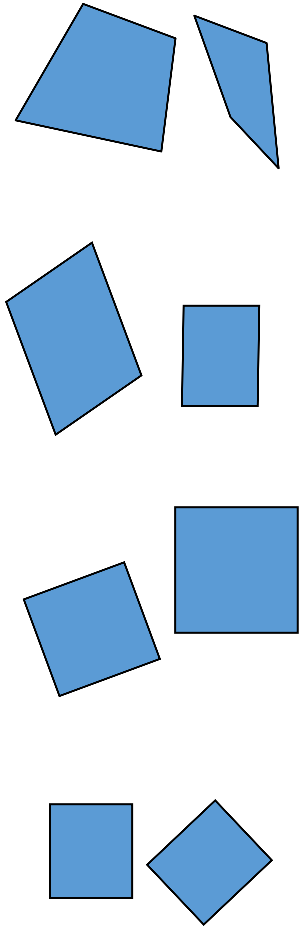
- Note that h, g are typically very small (~0.00001)
- Turns squares into a Quadrilateral
  - Usually quazi-trapezoids
- Can describe change of perspective on planes



Courtesy of line.17qq.com/

Hierarchy of 2D transformations

transformed squares



invariants

Projective
8dof

$$\begin{bmatrix} h_{11} & h_{12} & h_{13} \\ h_{21} & h_{22} & h_{23} \\ h_{31} & h_{32} & h_{33} \end{bmatrix}$$

Concurrency, collinearity, order of contact (intersection, tangency, inflection, etc.), cross ratio

Affine
6dof

$$\begin{bmatrix} a_{11} & a_{12} & t_x \\ a_{21} & a_{22} & t_y \\ 0 & 0 & 1 \end{bmatrix}$$

Parallellism, ratio of areas, ratio of lengths on parallel lines (e.g midpoints), linear combinations of vectors (centroids).
The line at infinity $\mathbf{l}_\infty$

Similarity
4dof

$$\begin{bmatrix} sr_{11} & sr_{12} & t_x \\ sr_{21} & sr_{22} & t_y \\ 0 & 0 & 1 \end{bmatrix}$$

Ratios of lengths, angles.
The circular points I,J

Euclidean
3dof

$$\begin{bmatrix} r_{11} & r_{12} & t_x \\ r_{21} & r_{22} & t_y \\ 0 & 0 & 1 \end{bmatrix}$$

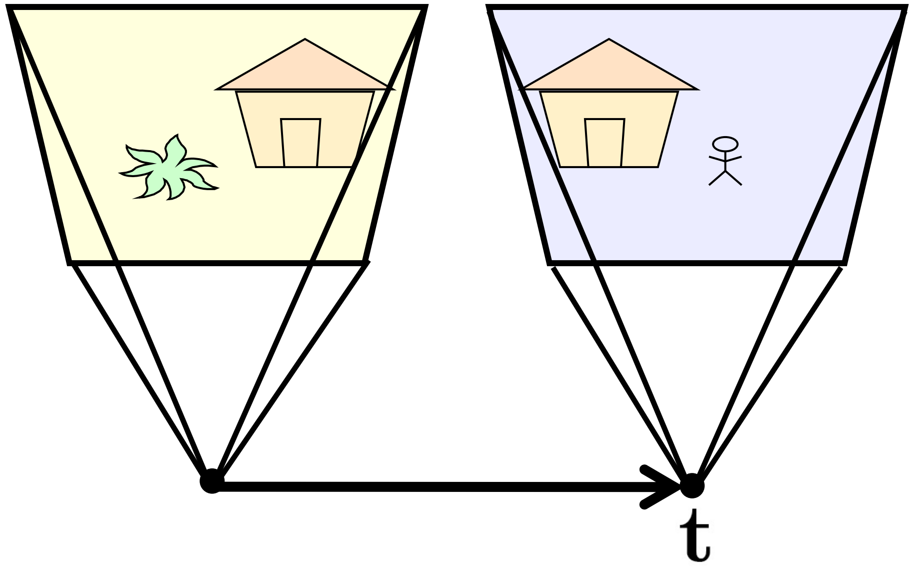
lengths, areas.

Courtesy of Marc Pollefeys

Stereo

- In stereo cameras-couple there is only translation along the 3D X axes, with no 3D rotation.

$$\mathbf{R} = \mathbf{I}_{3\times 3}$$
$$\mathbf{t} = [\ 1 \quad 0 \quad 0\ ]^T$$



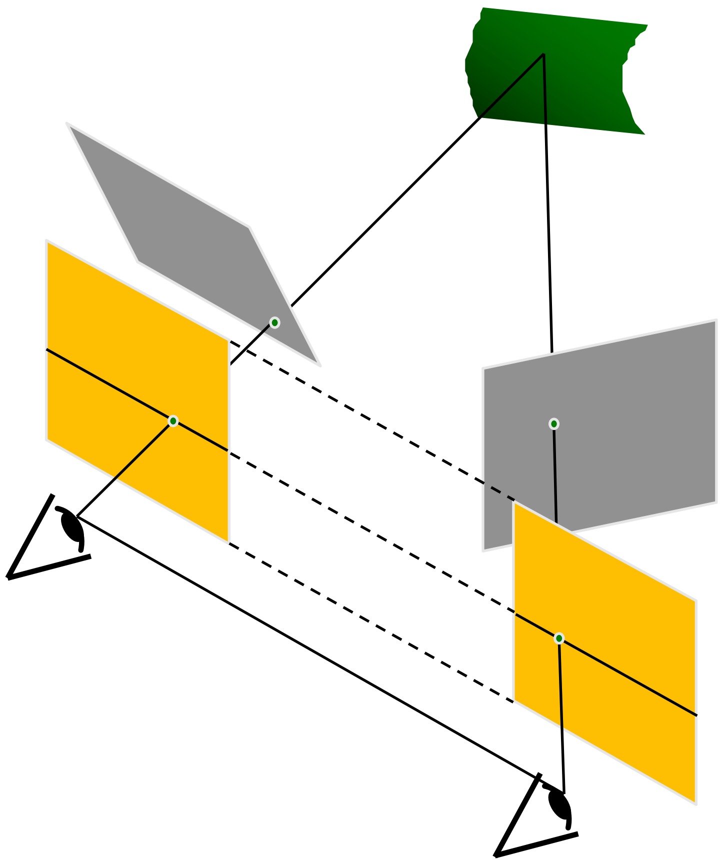
- Therefore, all pixel-matches has same Y coordinate
- And the depth is a function of the disparity (x_1-x_2)
- This makes matching process much easier:
  - Faster – small search area, good initial guess
  - Robust – less possible false matches

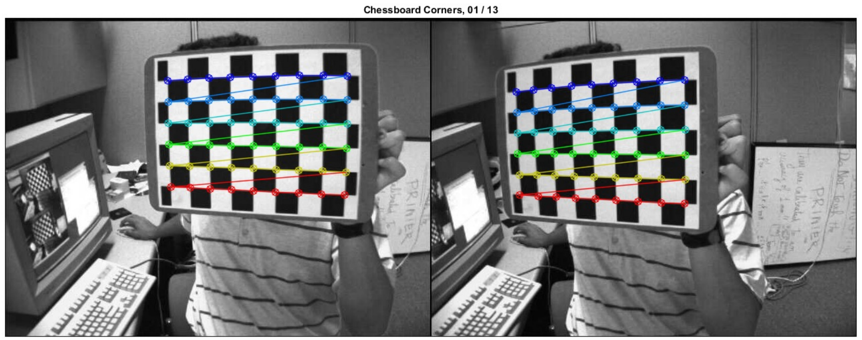
- But is highly unlikely to get!
  - Nano-movements break the stereo-assumption
  - Homography to the rescue: stereo rectification

Stereo image rectification

- Re-project image planes:
  - onto a common plane
  - parallel to the line between optical centers
- Pixel motion is horizontal after this transformation
- The rectification is two homographies:
  - Two 3x3 homographic transformations
  - One for each input image re-projection

    C. Loop and Z. Zhang. Computing Rectifying Homographies for Stereo Vision CVPR 1999
  - We usually warp the images using the transformations
- Why it works?
  - True rectification can be achieved using two 3D rotations
  - Perspective transformation model camera rotation
- Stereo Calibration:
  - finding the transformations
  - Usually also includes lens-distortion calibration
  - Kitti already did this for us





Courtesy of amroamroamro.github.io/

Camera Matrix

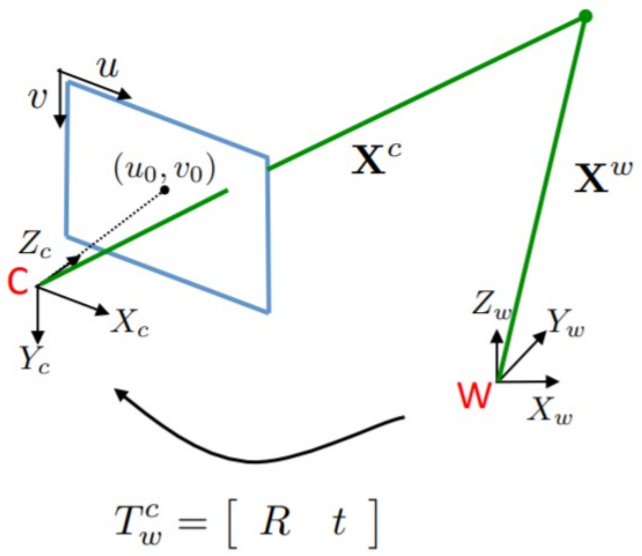
$$P = K[R \mid t]$$

K_{3x3} - calibration matrix, intrinsic params

$$K = \begin{bmatrix} f_x & & p_x \\ & f_y & p_y \\ & & 1 \end{bmatrix}$$

f *- focal length,*

p *- principal point*



In order to apply the camera model, objects must be expressed in camera coordinates.

- Transform from world to camera coords:

$[R|t]_{3x3}$ – extrinsic params

Where:

$R = R_{w \to c}$ and $t = t_{c \to w}$

Projection: $\begin{pmatrix} x' \\ y' \\ z' \end{pmatrix} = K[R \mid t] \begin{pmatrix} X_w \\ Y_w \\ Z_w \\ 1 \end{pmatrix}$ and then as earlier: $\begin{pmatrix} x \\ y \end{pmatrix} = \begin{pmatrix} \frac{x'}{z'} \\ \frac{y'}{z'} \end{pmatrix}$

Triangulation

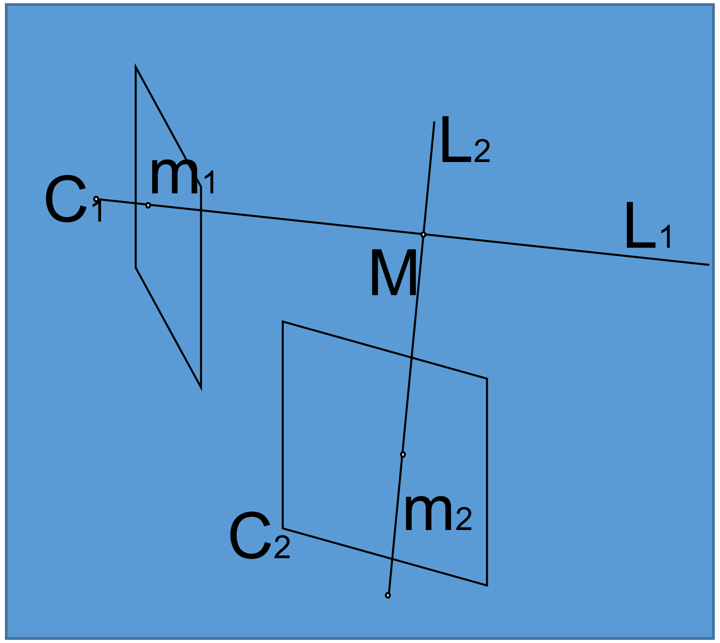
- Triangulation is finding 3D location
  - Using at least 2 views of the point
- Backprojection: $\lambda x = PX$

$$\begin{pmatrix} \lambda x' \\ \lambda y' \\ \lambda \end{pmatrix}_{3x1} = K[R\,|\,t]X = PX = \begin{bmatrix} \cdots P_1 \cdots \\ \cdots P_2 \cdots \\ \cdots P_3 \cdots \end{bmatrix}_{3x4} X_{4x1}$$

$$\begin{pmatrix} \lambda x \\ \lambda y \\ \lambda \end{pmatrix} = \begin{matrix} P_1 X \\ P_2 X \\ P_3 X \end{matrix} \Rightarrow \begin{matrix} P_3 X x = P_1 X \\ P_3 X y = P_2 X \end{matrix} \Rightarrow \begin{bmatrix} P_3 x - P_1 \\ P_3 y - P_2 \end{bmatrix} X = 0$$

- We have 2 cameras: P and P’, so:

$$AX = \begin{bmatrix} P_3 x - P_1 \\ P_3 y - P_2 \\ P_3' x' - P_1' \\ P_3' y' - P_2' \end{bmatrix} X = 0$$



SVD

- X is the kernel of A. How can we find it?
  - A simple pseudo-inverse will return X=0
- SVD matrix decomposition:
  - U_{MxM}, V^T_{NxN} are orthonormal $A_{MxN} = U_{MxM} \Sigma_{MxN} V^T_{NxN}$
  - Σ_{MxN} is semi-diagonal
  - Descending singular values σ_i on the diagonal

$$\Sigma_{MxN} = \begin{bmatrix} \sigma_1 & & & & \\ & \sigma_2 & & & \\ & & \sigma_3 & & \\ & & & \ddots & \\ & & & & \sigma_N \\ & & & & \end{bmatrix}$$

- We do SVD to A_{3x4}
- If $\sigma_3=0$
  - It means the rays intersect
  - We take X to be the associated vector V_3
- Else
  - The rays don't intersect. We wish to get the closest point.
  - We still takes $X=V_3$, it's the least-squares solution: $\arg\min_{\mathtt{X}} \sum_i \left(\mathtt{x}_i - \lambda^{-1}\mathbf{P}_i\mathtt{X}\right)^2$

Iterative least squares

- Problem: our equations were unevenly weighted.
  - Each camera's λ is arbitrary and different
- Solution: Iterative least squares
  - In each iteration, we set X~ to be the last solution

$$\begin{bmatrix} \frac{1}{\mathtt{P}_3\tilde{\mathtt{X}}} \begin{pmatrix} \mathtt{P}_3 x - \mathtt{P}_1 \\ \mathtt{P}_3 y - \mathtt{P}_2 \end{pmatrix} \\ \frac{1}{\mathtt{P}'_3\tilde{\mathtt{X}}} \begin{pmatrix} \mathtt{P}'_3 x - \mathtt{P}'_1 \\ \mathtt{P}'_3 y - \mathtt{P}'_2 \end{pmatrix} \end{bmatrix} \mathtt{X} = 0$$

- To summarize our shortcuts:
  - We use linear triangulation
  - We used SVD even if there is no kernel
  - The λ is arbitrary