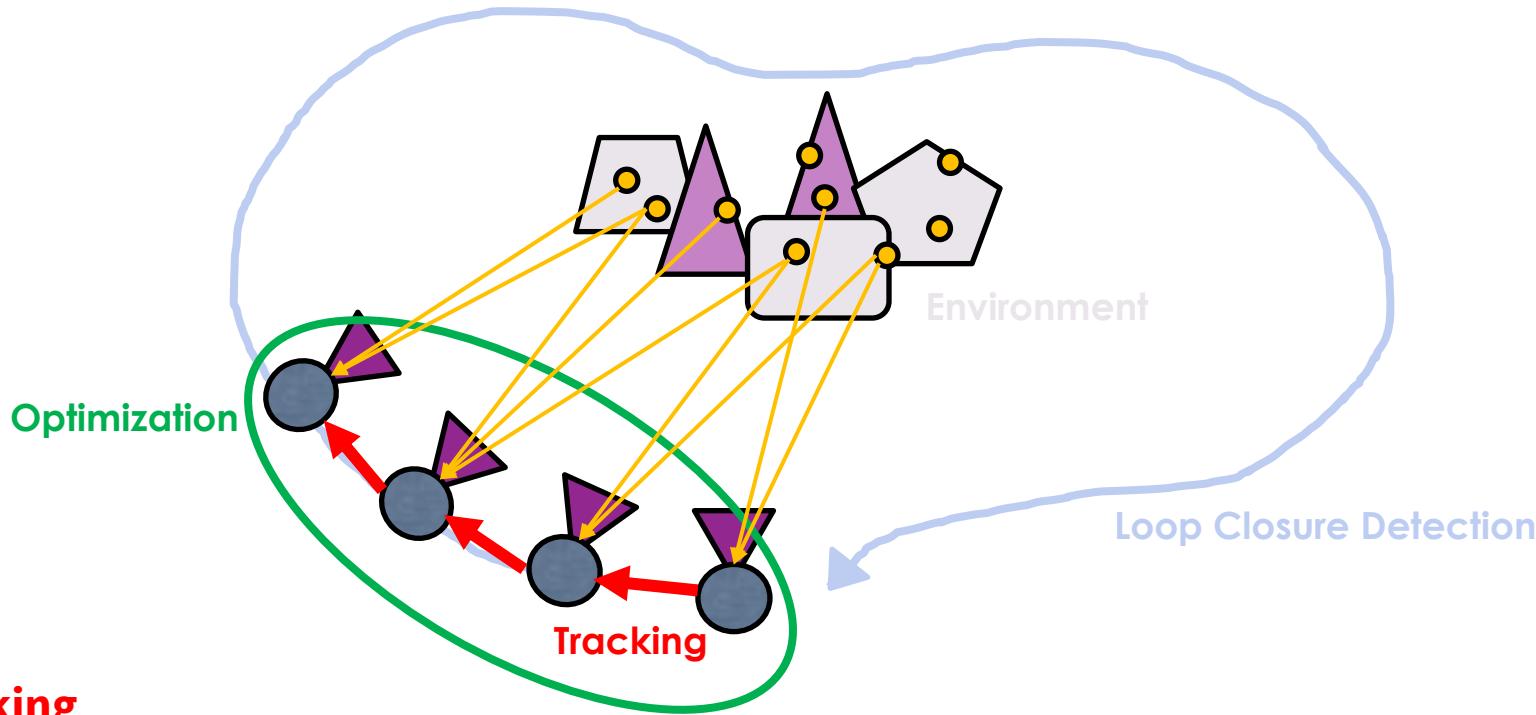


# Robotic Navigation and Exploration

Week 8: Multi-view Geometry & Transformation Optimization

Min-Chun Hu [anitahu@cs.nthu.edu.tw](mailto:anitahu@cs.nthu.edu.tw)  
CS, NTHU

# SLAM Overview



## Pose Tracking

Using continuous measurement to estimate the movement

## Local Optimization

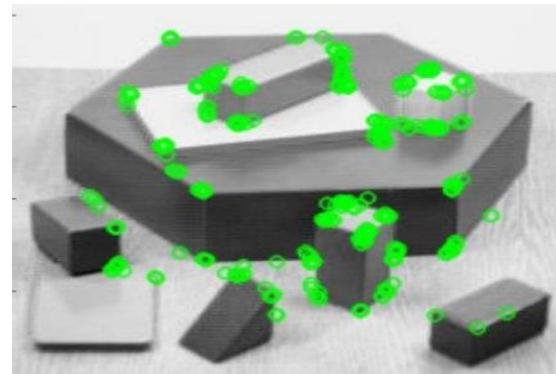
Using several measurement to optimize the error of the map

## Loop Closure Detection

Detecting the loop to stabilize the global structure

# Information from Image Data

Sparse



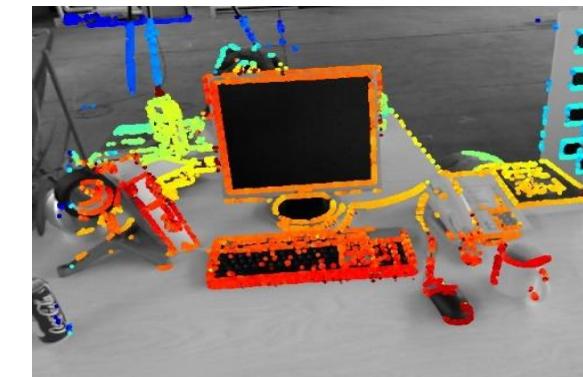
Sparse Feature Points

Dense



All Points

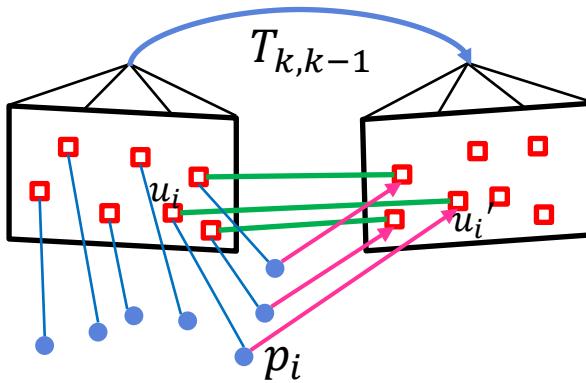
Semi-Dense



Important Points

# Objective Function

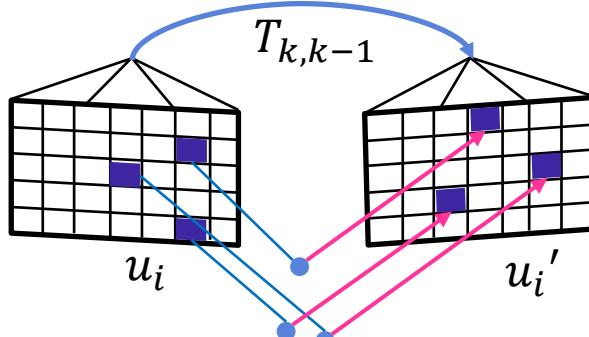
## Indirect Method



Minimize Geometric Error (Reprojection)

$$T_{k,k-1} = \underset{T}{\operatorname{argmin}} \sum_i^N \|u_i' - \pi p_i\|^2$$

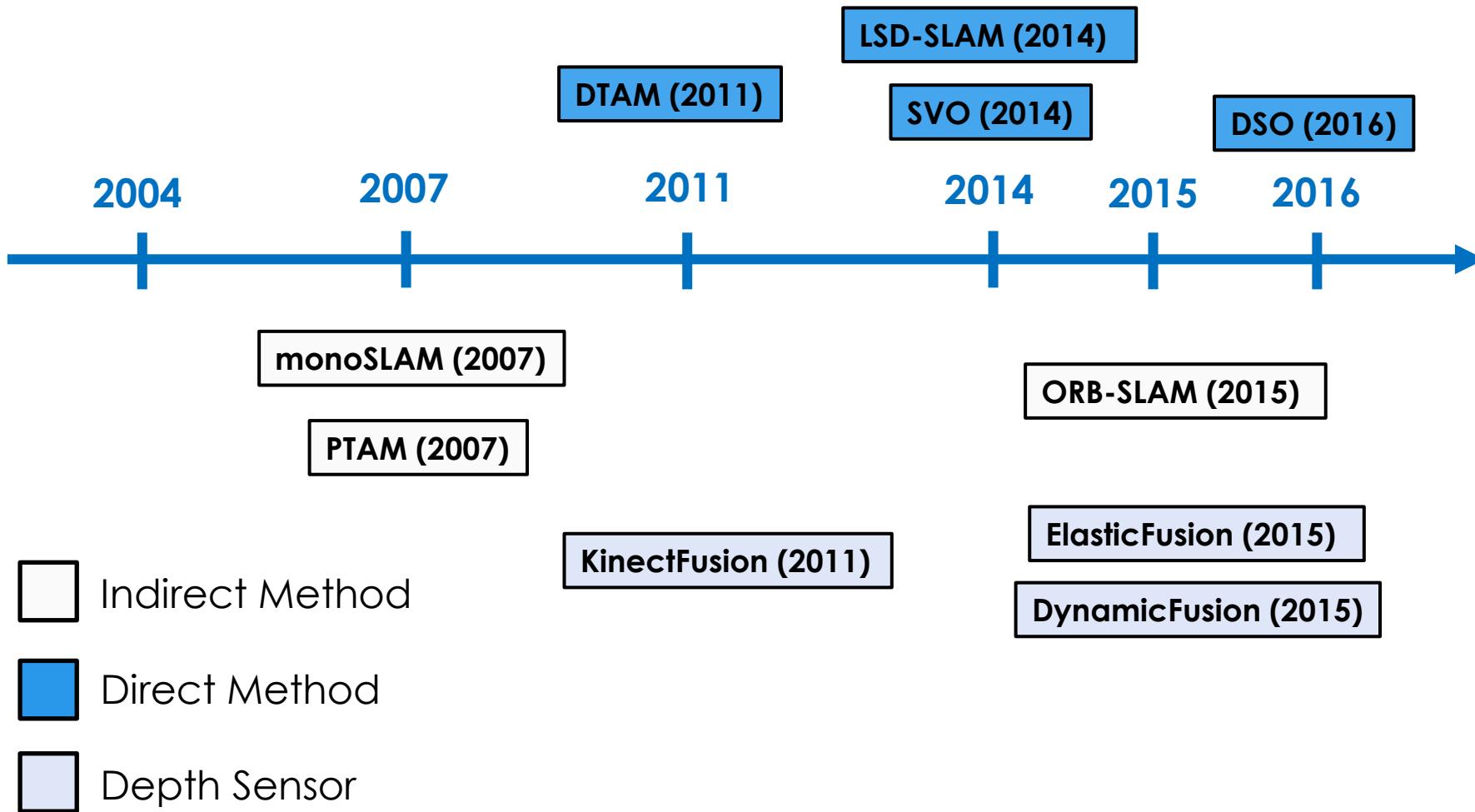
## Direct Method



Minimize Photometric Error (Pixel Grayscale)

$$T_{k,k-1} = \underset{T}{\operatorname{argmin}} \sum_i^N \|I_k(u_i') - I_{k-1}(u_i)\|^2$$

# History of Visual SLAM



# History of Visual SLAM

## First dense monocular SLAM algorithm.

Using GPU to accelerate the computation and build dense point cloud.

DTAM (2011)

Improve the speed of DTAM by only building the semi-dense map of whole image.

LSD-SLAM (2014)

2004

2007

2011

2014

2015

2016

## First real-time monocular SLAM algorithm.

Separate the system into two thread: tracking and mapping. The pipeline is the basis of modern SLAM system.

PTAM (2007)

KinectFusion (2011)

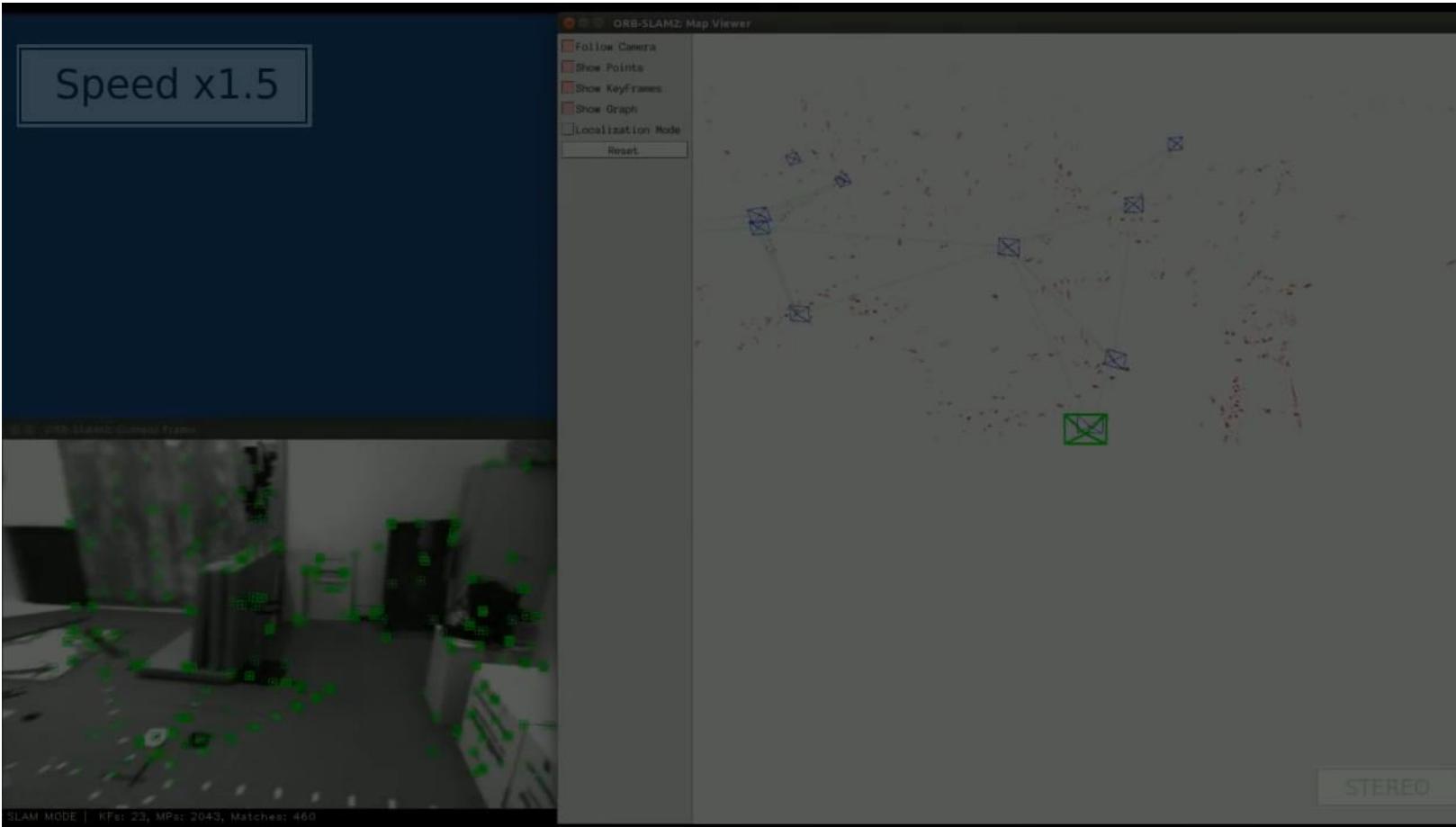
## First depth SLAM algorithm.

Using the volumetric fusion map to construct complete and beautiful dense 3D point cloud.

ORB-SLAM (2015)

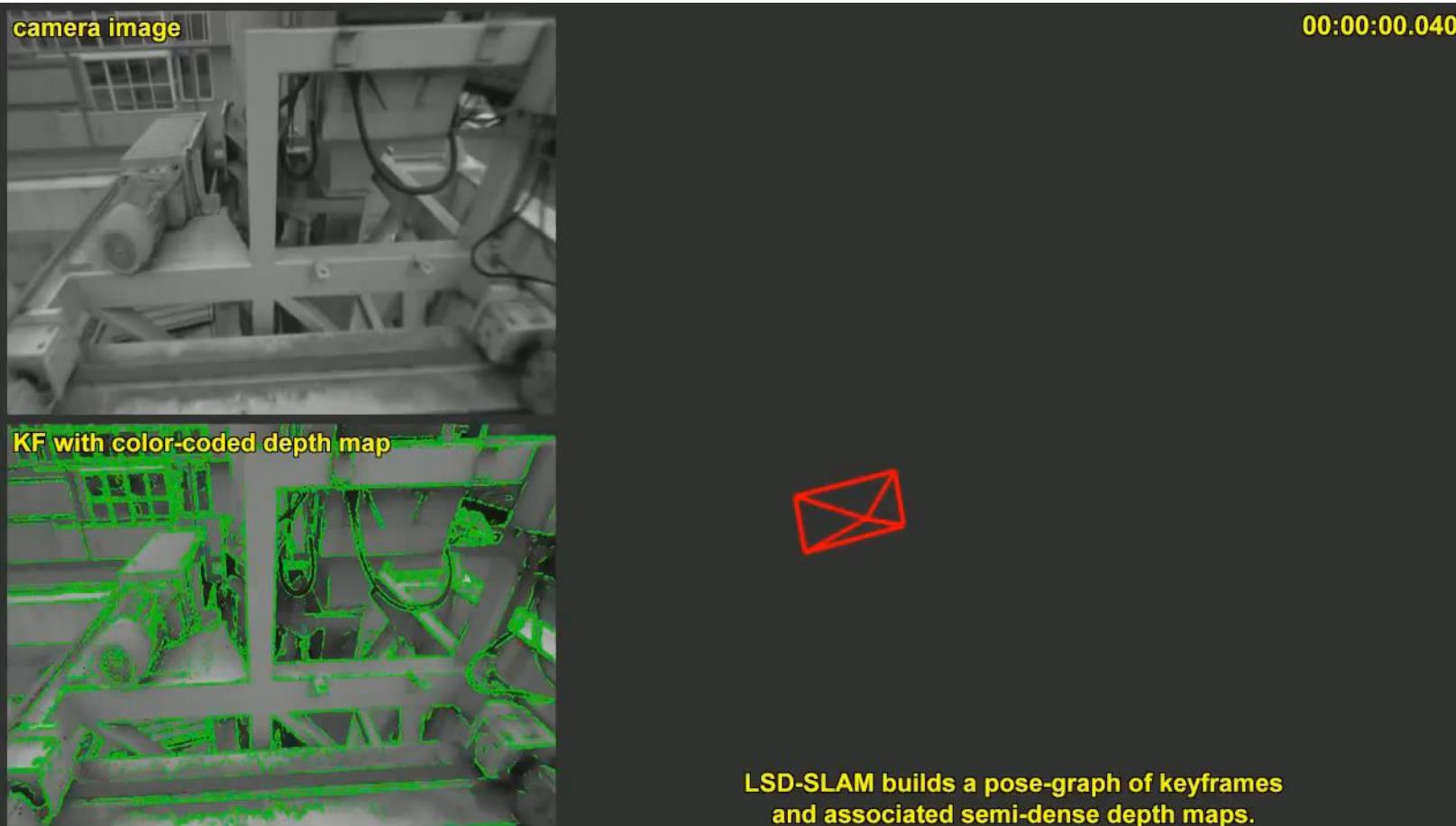
Assembles recent researches of feature-based SLAM. Use similar pipeline as PTAM. A stable and reliable monocular SLAM system.

# ORB-SLAM



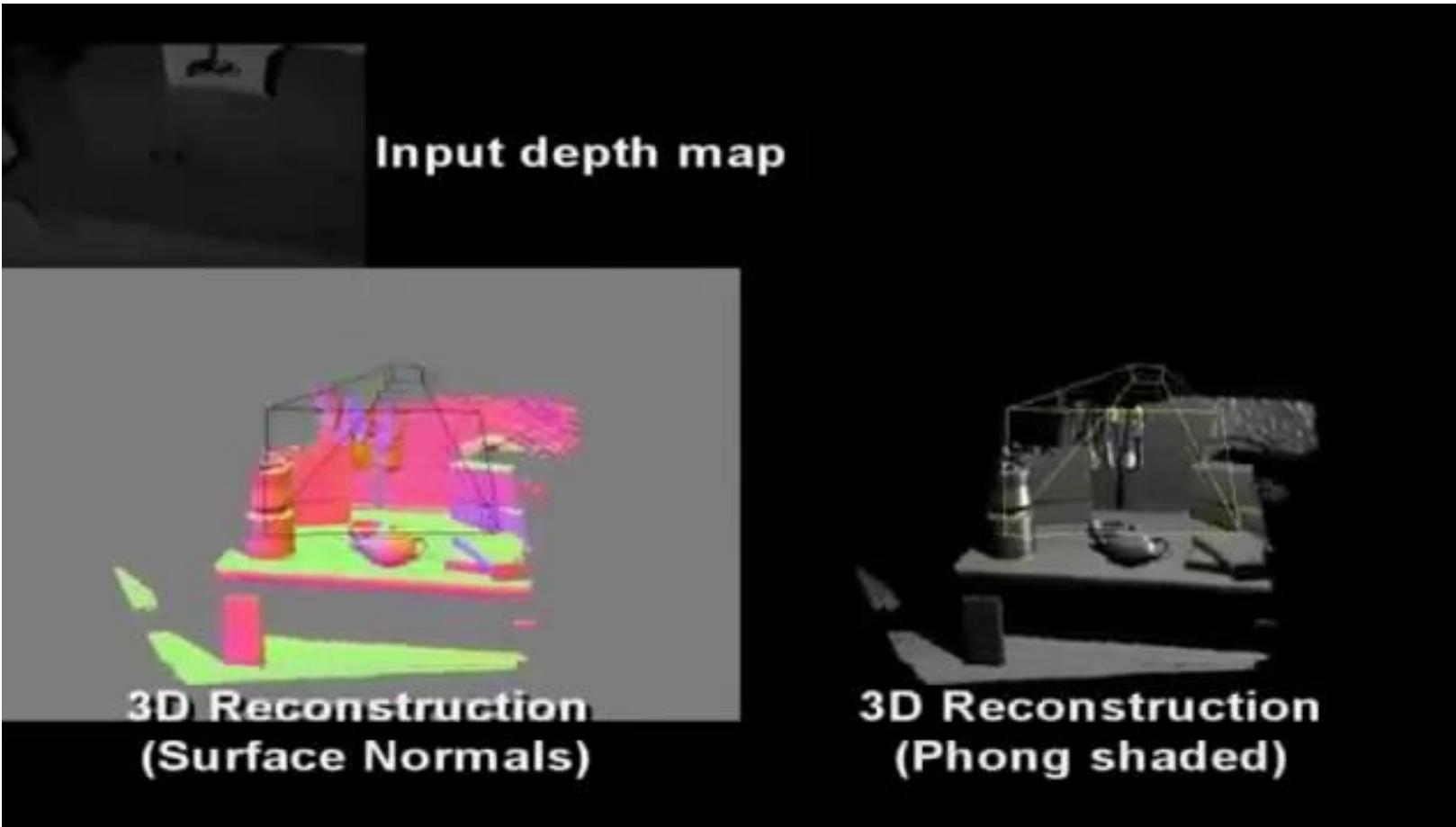
<https://www.youtube.com/watch?v=luBGKxgaxS0>

# LSD-SLAM



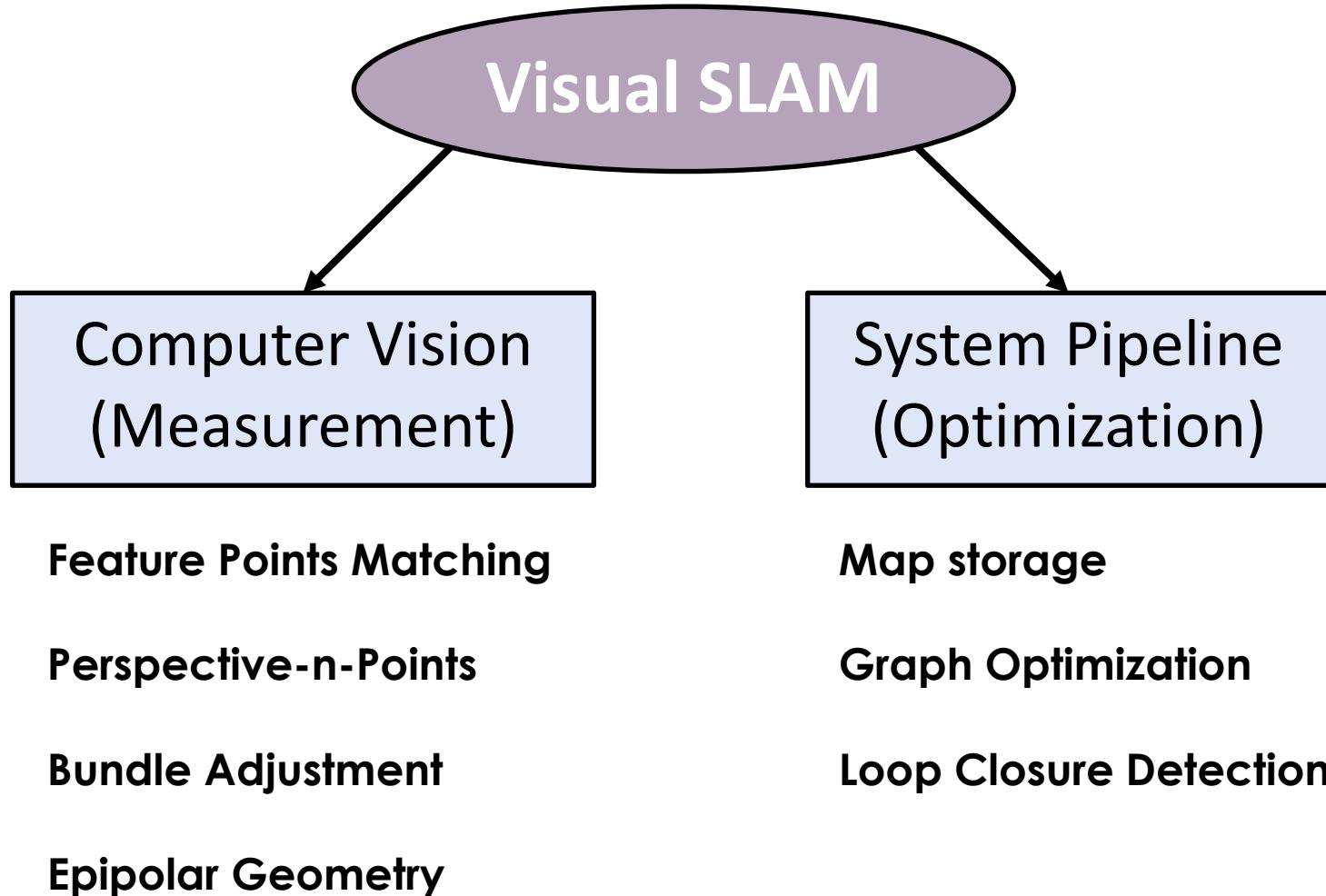
<https://www.youtube.com/watch?v=GnuQzP3gty4>

# Kinect Fusion



<https://www.youtube.com/watch?v=KOSSIKUJ-A>

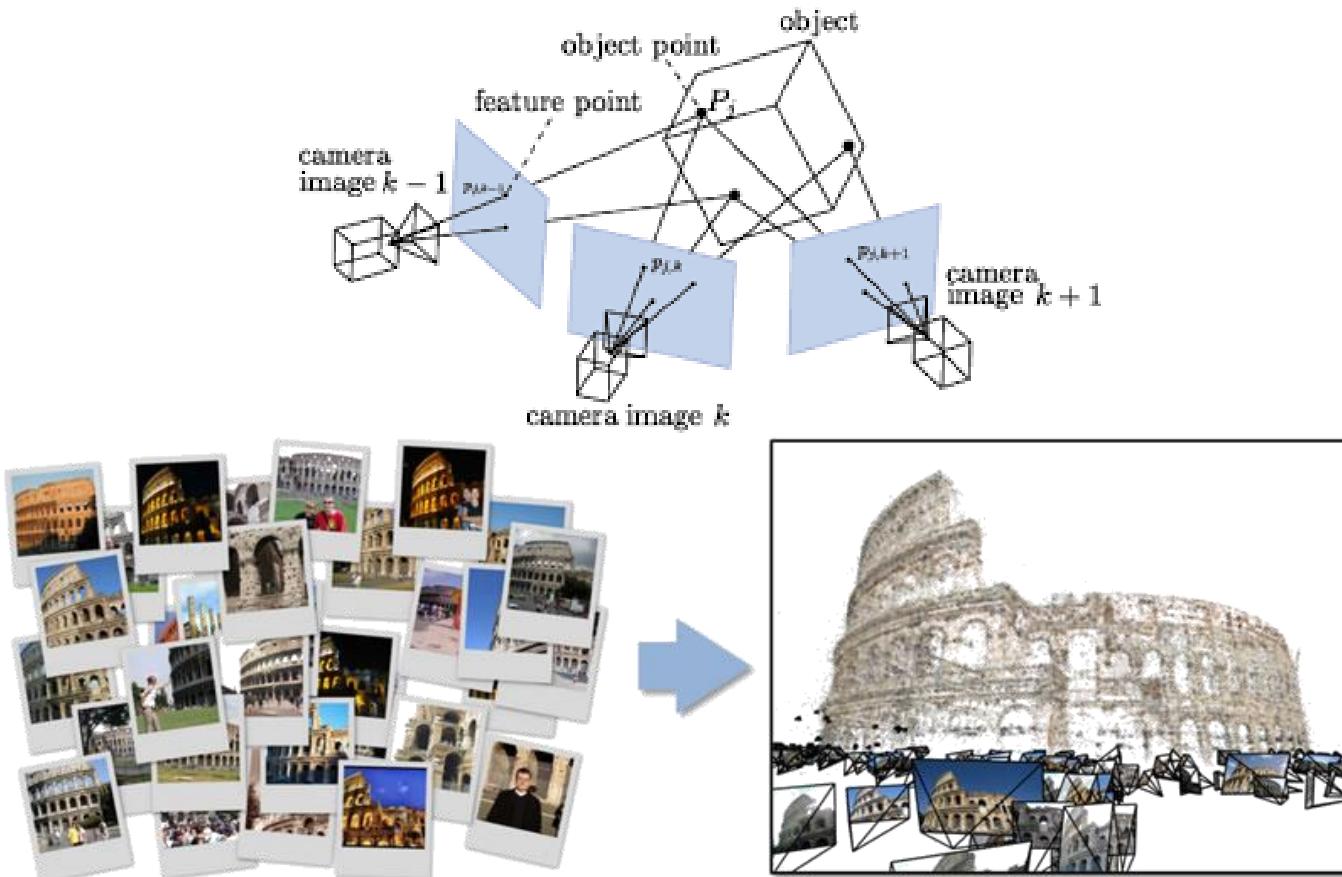
# Feature-based Visual SLAM



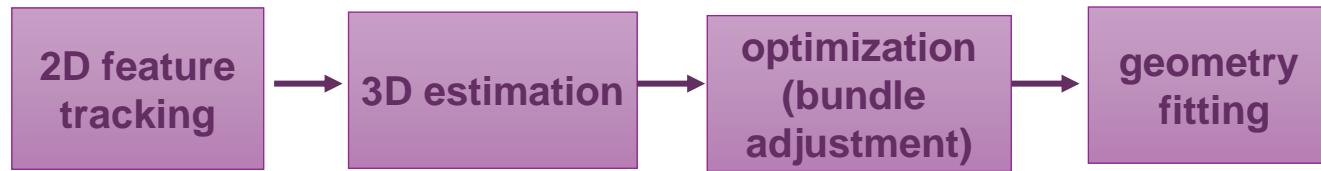
# Computer Vision / Multi-view Geometry

# Structure from Motion (SfM)

- Structure from motion: automatic recovery of **camera motion** and **scene structure** from two or more images.



# SfM Pipeline



- Step 1: Track Features
  - Detect good features (SIFT)
  - Find correspondences between frames
    - Lucas & Kanade-style motion estimation
    - window-based correlation
    - SIFT matching



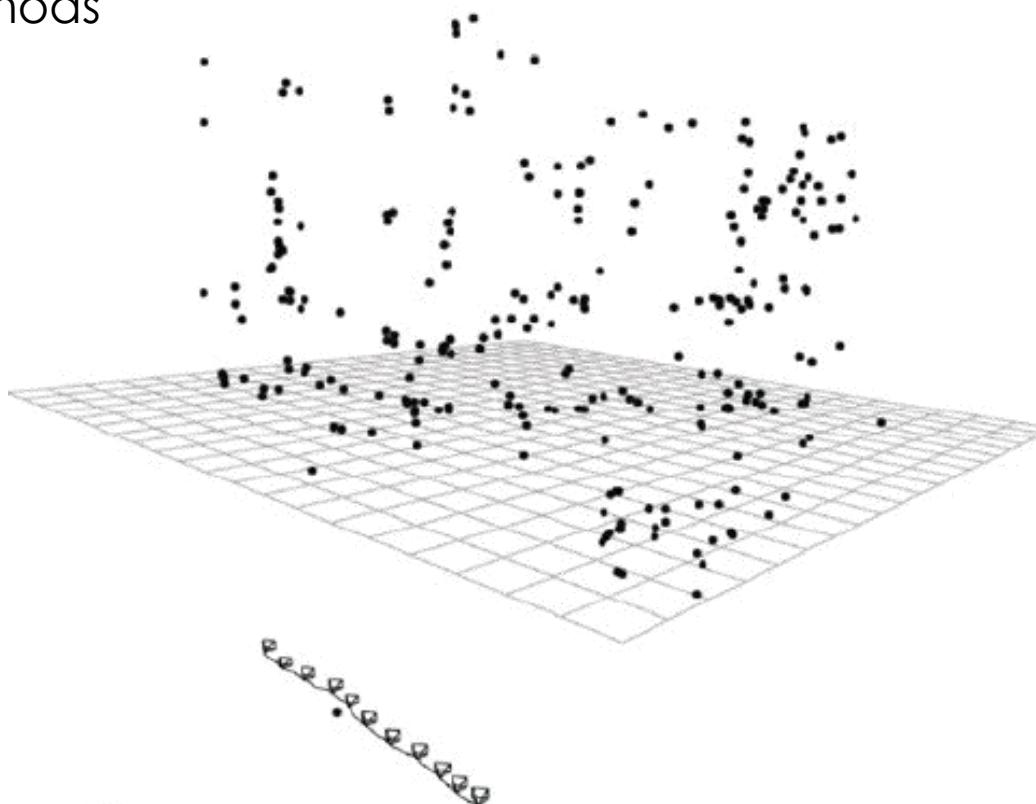
# SfM Pipeline

- Step 2: Estimate Motion and Structure
  - Simplified projection model
  - 2 or 3 views at a time



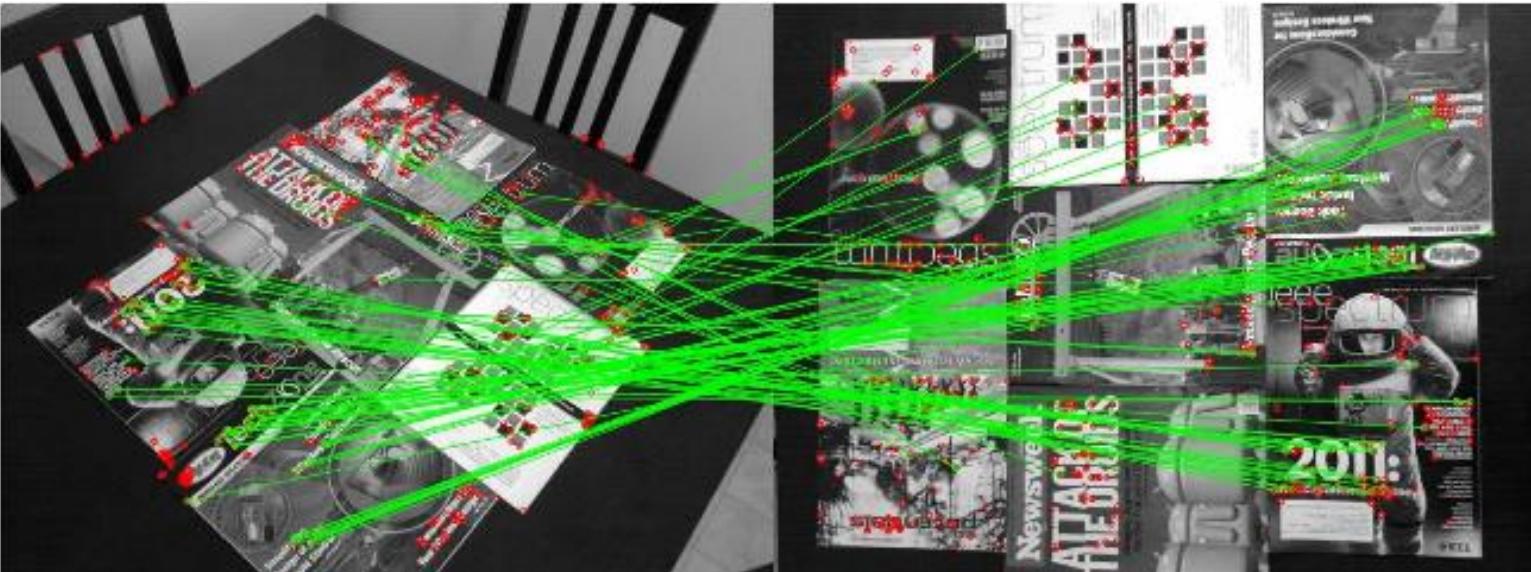
# SfM Pipeline

- Step 3: Optimization to refine estimation
  - “Bundle adjustment” in photogrammetry
  - Other iterative methods

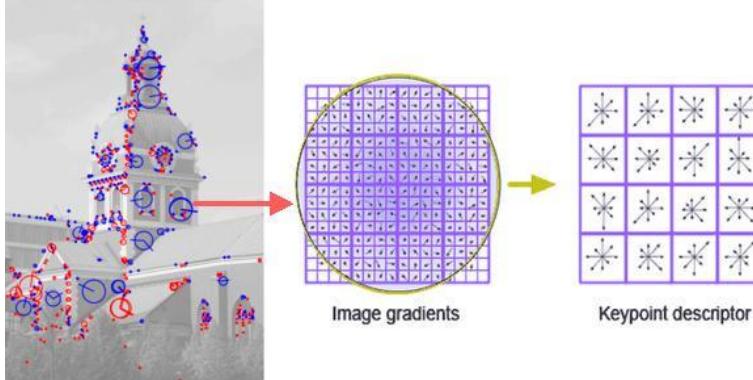


# Feature Points Matching

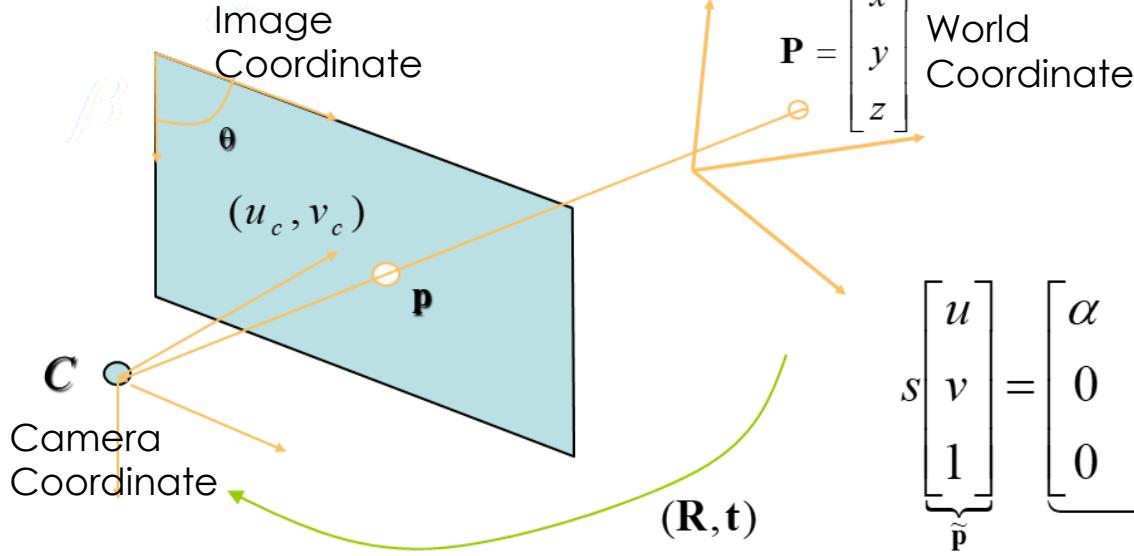
## Feature Points Detection/Description



SIFT, SURF, ORB



# Camera Calibration



$$s \begin{bmatrix} u \\ v \\ 1 \end{bmatrix} = \underbrace{\begin{bmatrix} \alpha & \gamma & u_c \\ 0 & \beta & v_c \\ 0 & 0 & 1 \end{bmatrix}}_{\tilde{\mathbf{P}}} \underbrace{\begin{bmatrix} \mathbf{r}_1 & \mathbf{r}_2 & \mathbf{r}_3 & \mathbf{t} \end{bmatrix}}_{[\mathbf{R} \quad \mathbf{t}]} \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix} \underbrace{\mathbf{P}}$$

## □ Intrinsics:

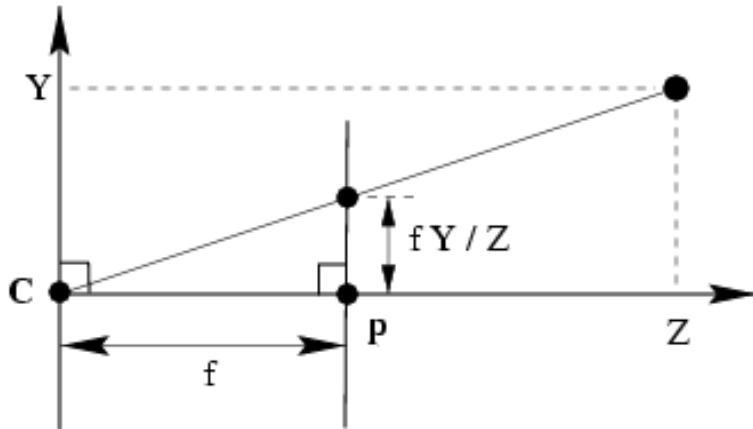
- scale factor
- focal length
- aspect ratio
- principle point
- radial distortion

## □ Extrinsics

- optical center
- camera orientation

A camera is calibrated when intrinsics/extrinsics are known.

# Pinhole Camera Projection Model



$$x = \frac{fX}{Z} \quad y = \frac{fY}{Z}$$

$$\begin{pmatrix} x \\ y \\ 1 \end{pmatrix} \sim \begin{pmatrix} fX \\ fY \\ Z \end{pmatrix} = \begin{bmatrix} f & 0 & 0 & 0 \\ 0 & f & 0 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix} \begin{pmatrix} X \\ Y \\ Z \\ 1 \end{pmatrix}$$

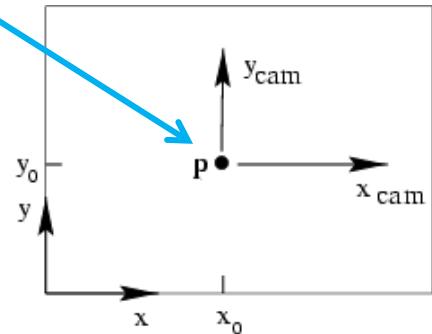
$$\begin{pmatrix} x \\ y \\ 1 \end{pmatrix} \sim \begin{pmatrix} fX \\ fY \\ Z \end{pmatrix} = \begin{bmatrix} f & 0 & 0 \\ 0 & f & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix} \begin{pmatrix} X \\ Y \\ Z \\ 1 \end{pmatrix}$$

intrinsic matrix  
extrinsic matrix  
(Camera Coordinate  
=World Coordinate)

$$\mathbf{x} \sim \mathbf{K}[\mathbf{I}|0]\mathbf{x}$$

# Principal Point Offset

principal  
point



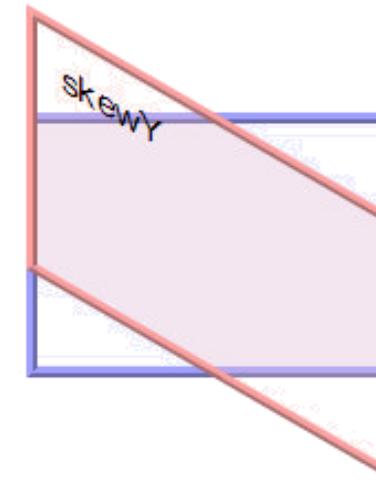
$$\begin{pmatrix} x \\ y \\ 1 \end{pmatrix} \sim \begin{pmatrix} fX \\ fY \\ Z \end{pmatrix} = \begin{bmatrix} f & 0 & x_0 \\ 0 & f & y_0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{pmatrix} X \\ Y \\ Z \\ 1 \end{pmatrix}$$

$$\mathbf{x} \sim \mathbf{K}[\mathbf{I}|0]\mathbf{x}$$

# Intrinsic Matrix

$$\mathbf{K} = \begin{bmatrix} f & 0 & x_0 \\ 0 & f & y_0 \\ 0 & 0 & 1 \end{bmatrix} \rightarrow \mathbf{K} = \begin{bmatrix} fa & s_y & x_0 \\ 0 & f & y_0 \\ 0 & 0 & 1 \end{bmatrix}$$

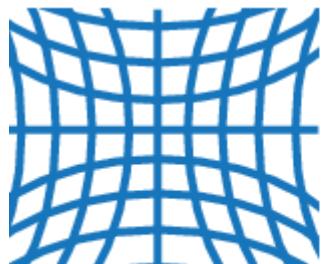
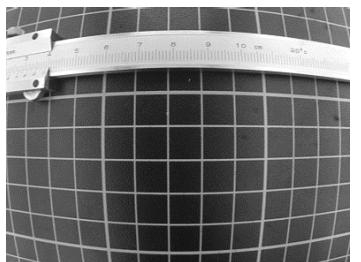
Good enough for modeling the camera projection?



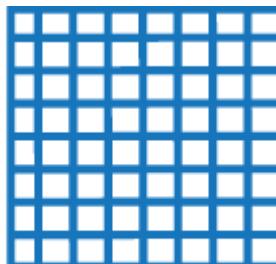
$a$  : aspect ratio (for non-square pixels)  
 $s_y$  : skew (for non-rectangular pixels)

$$x_{\text{distorted}} = x + [2 * p_1 * x * y + p_2 * (r^2 + 2 * x^2)]$$

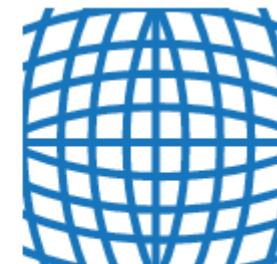
$$y_{\text{distorted}} = y + [p_1 * (r^2 + 2 * y^2) + 2 * p_2 * x * y]$$



Negative radial distortion  
"pincushion"

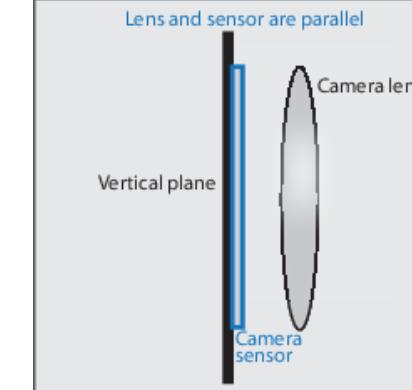


No distortion

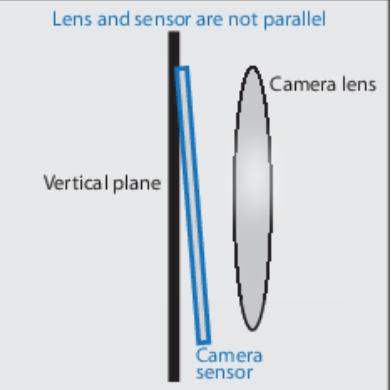


Positive radial distortion  
"barrel"

Zero Tangential Distortion

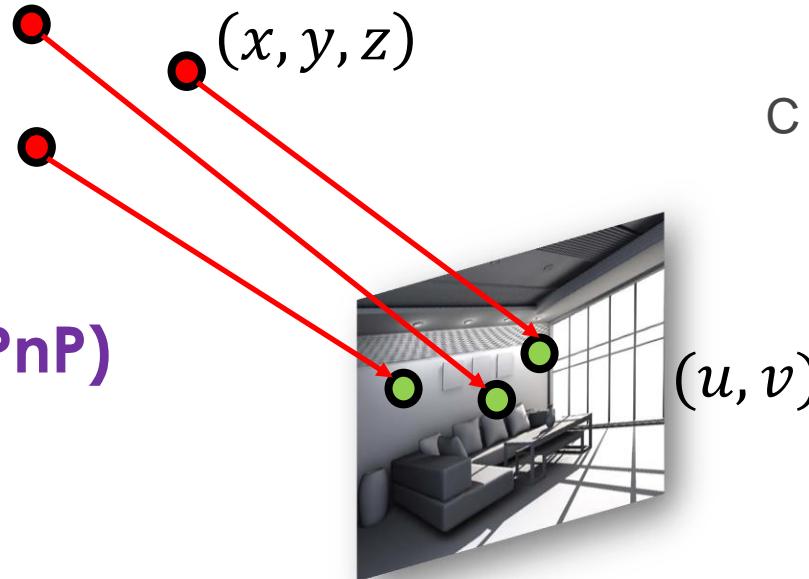


Tangential Distortion



# Transformation Matrix Estimation by Reprojection

Perspective-n-Point (PnP)

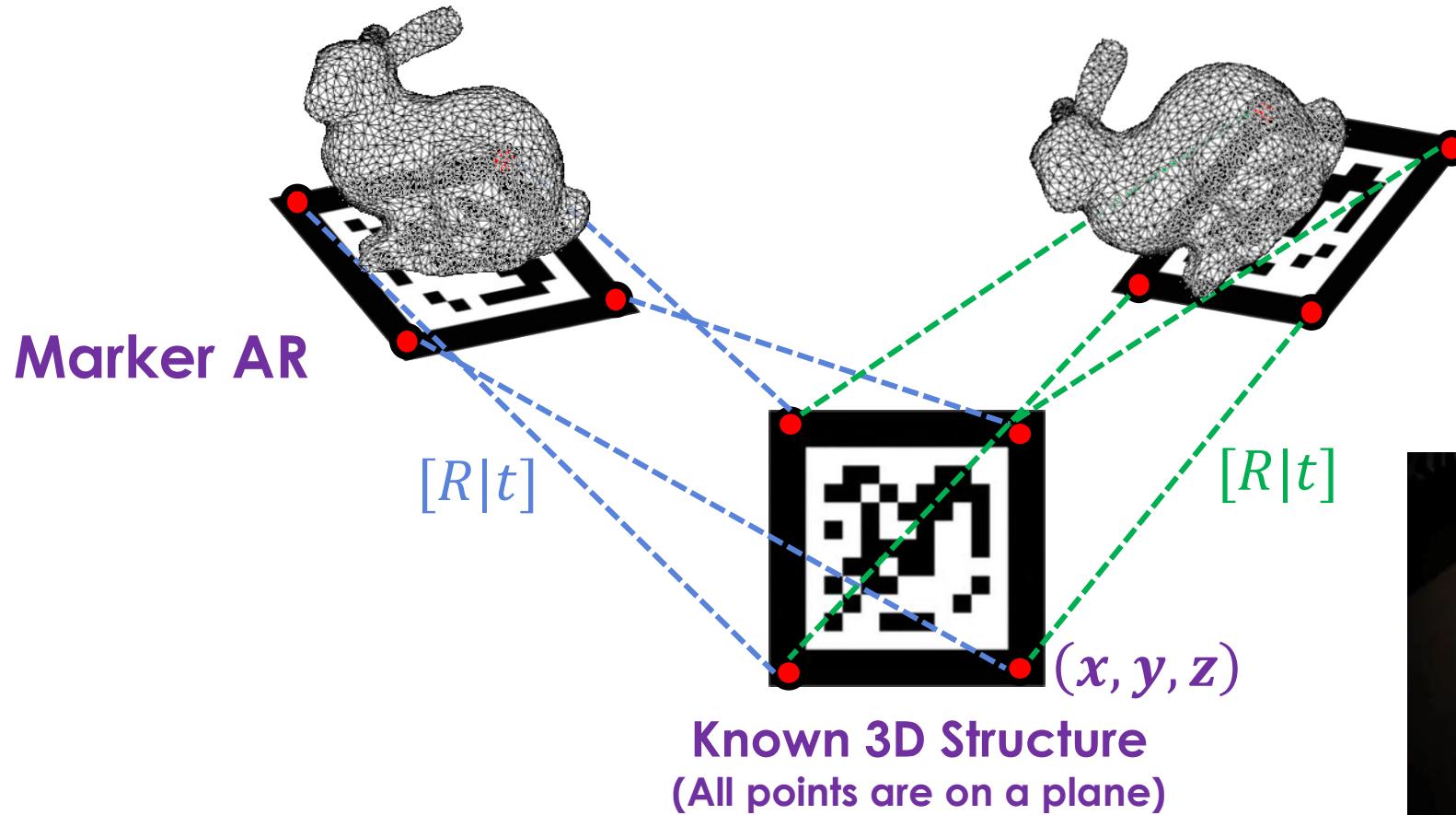


$$C = K [R \mid t] = [KR \mid KRT] = [A \mid AT]$$

$$s \begin{bmatrix} u \\ v \\ 1 \end{bmatrix} = \begin{bmatrix} f_x & \gamma & u_0 \\ 0 & f_y & v_0 \\ 0 & 0 & 1 \end{bmatrix} \boxed{\begin{bmatrix} r_{11} & r_{12} & r_{13} & t_1 \\ r_{21} & r_{22} & r_{23} & t_2 \\ r_{31} & r_{32} & r_{33} & t_3 \end{bmatrix}} \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix}$$

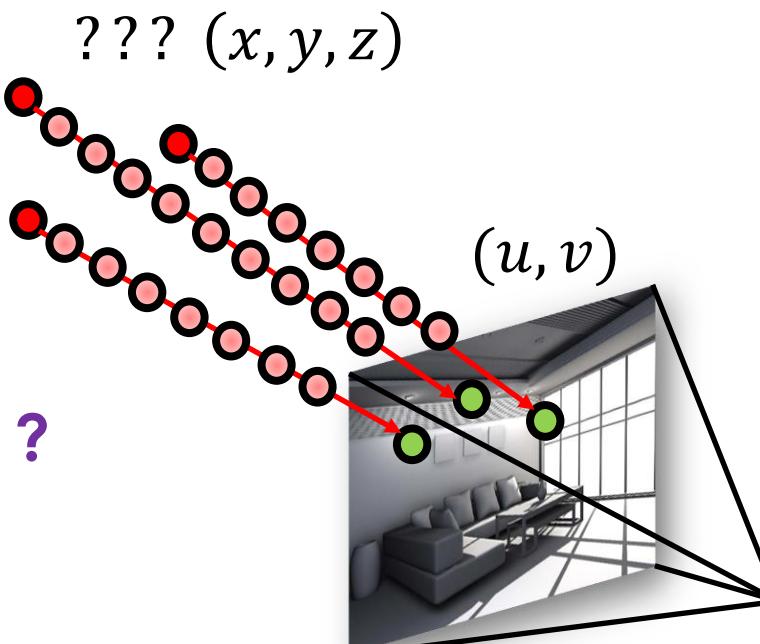
*Unknown* Need 3 Points to Solve (P3P)

# Transformation Matrix Estimation by Reprojection



# Transformation Estimation by Reprojection

Unknown Structure ?


$$s \begin{bmatrix} u \\ v \\ 1 \end{bmatrix} = \begin{bmatrix} f_x & \gamma & u_0 \\ 0 & f_y & v_0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} r_{11} & r_{12} & r_{13} & t_1 \\ r_{21} & r_{22} & r_{23} & t_2 \\ r_{31} & r_{32} & r_{33} & t_3 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix}$$

## Transformation Matrix Estimation by Reprojection

$$sx_1 = KM_1P$$

$$sx_2 = KM_2P$$

⋮

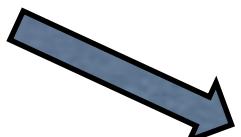
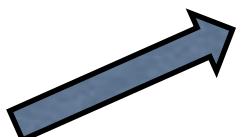
**Solve M**

**Direct  
Optimization**

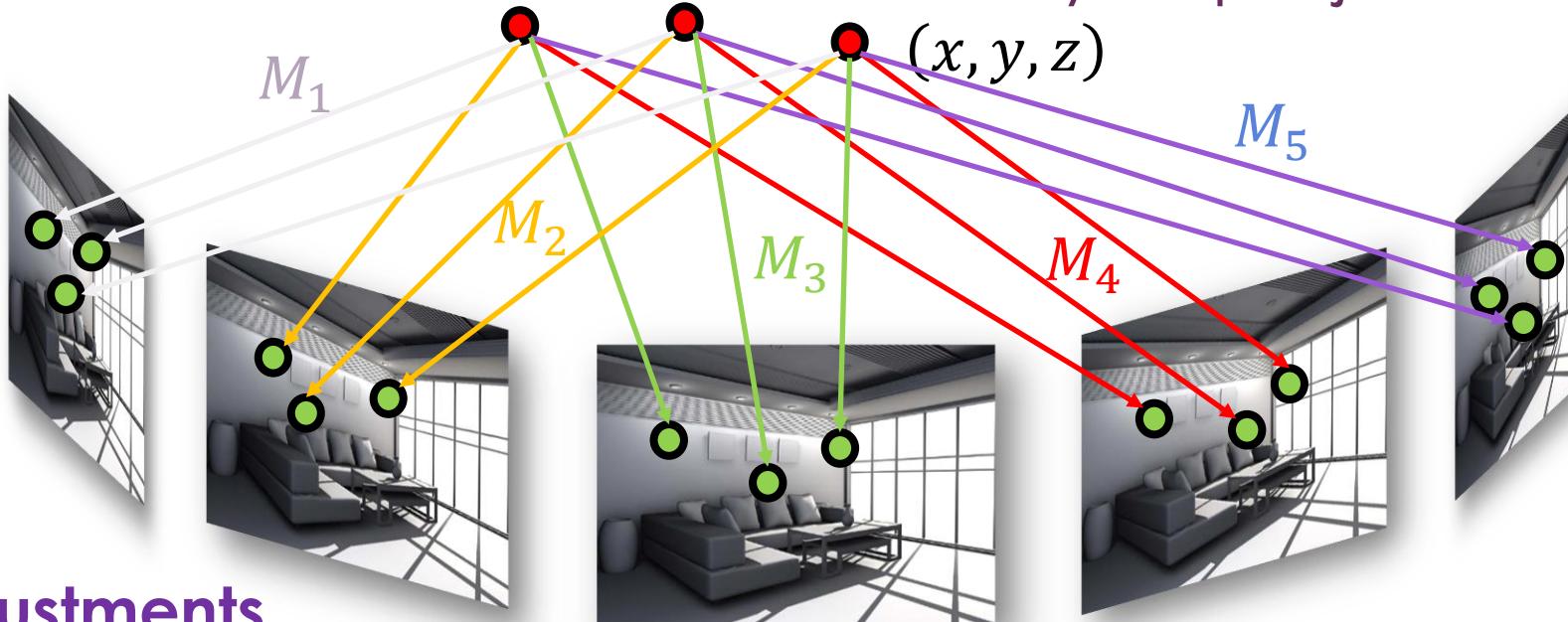
**Bundle Adjustment**

**Reduce P**

**Epipolar Geometry**



# Transformation Matrix Estimation by Reprojection



## Bundle Adjustments

Optimize

Camera Matrix

$$\begin{bmatrix} u \\ v \\ 1 \end{bmatrix} = \begin{bmatrix} f_x & \gamma & u_0 \\ 0 & f_y & v_0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} r_{11} & r_{12} & r_{13} & t_1 \\ r_{21} & r_{22} & r_{23} & t_2 \\ r_{31} & r_{32} & r_{33} & t_3 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix}$$

$K$                                      $M$

# Unknown Structure Initialization

$$\overrightarrow{C_0p_0} \cdot (\overrightarrow{C_0C_1} \times \overrightarrow{C_1p_1}) = 0$$

$$p_0 \cdot (t \times Rp_1) = 0$$

$$p_0^T [t]_{\times} Rp_1 = 0$$

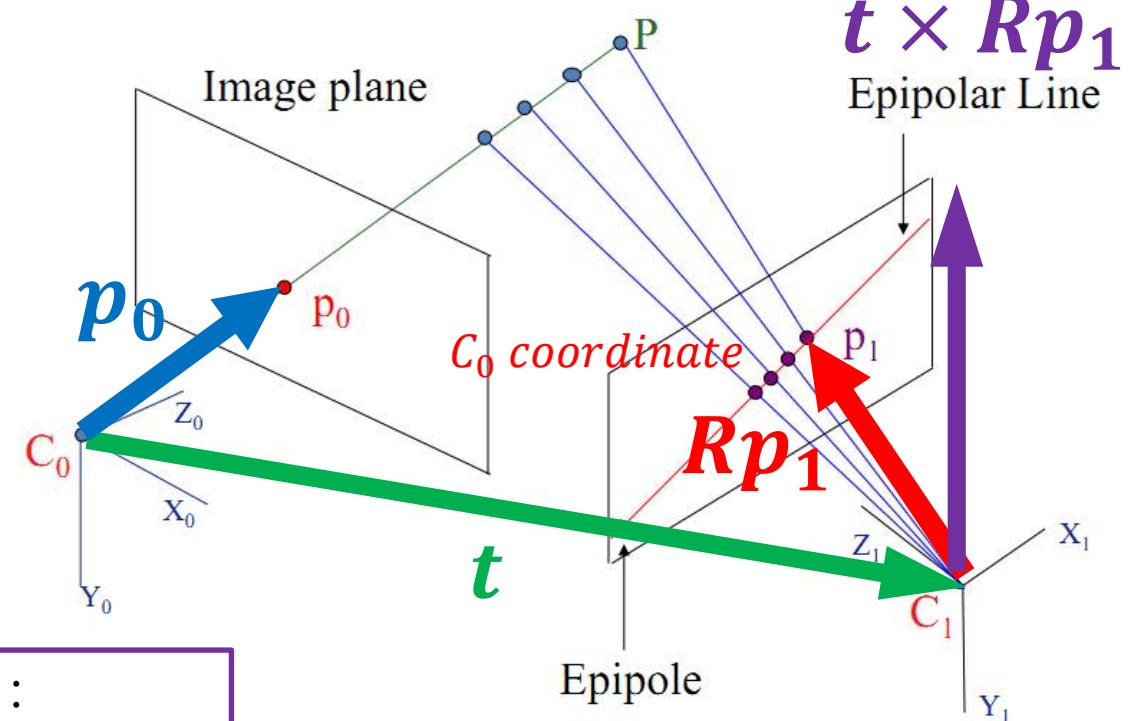
$$p_0^T E p_1 = 0$$

**Essential Matrix**

Cross Matrix :

$$[t]_{\times} = \begin{bmatrix} 0 & -t_3 & t_2 \\ t_3 & 0 & -t_1 \\ -t_2 & t_1 & 0 \end{bmatrix}$$

## Epipolar Geometry



## Unknown Structure Initialization

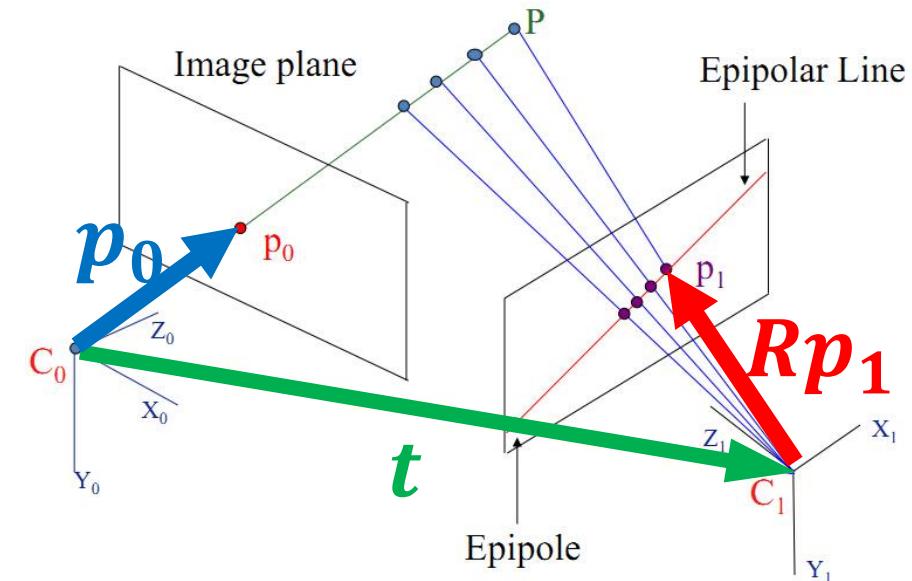
$$\mathbf{p}_0^T \mathbf{E} \mathbf{p}_1 = 0$$

$$(x_0 \quad y_0 \quad 1) \begin{pmatrix} E_{11} & E_{12} & E_{13} \\ E_{21} & E_{22} & E_{23} \\ E_{31} & E_{32} & E_{33} \end{pmatrix} \begin{pmatrix} x_1 \\ y_1 \\ 1 \end{pmatrix} = 0$$

Write as  $\mathbf{A} \mathbf{x} = \mathbf{0}$ , where  $\mathbf{x} = (E_{11}, E_{12}, E_{13}, \dots, E_{33})$

$$(x_0x_1 \quad x_0y_1 \quad x_0 \quad y_0x_1 \quad y_0y_1 \quad y_0 \quad x_1 \quad y_1 \quad 1) \begin{pmatrix} E_{11} \\ E_{12} \\ E_{13} \\ \vdots \\ E_{33} \end{pmatrix} = 0$$

## Epipolar Geometry



## Unknown Structure Initialization

### Essential Matrix Decomposition

$$E = [t]_{\times}R = U\Sigma V^T = U \begin{bmatrix} \sigma & 0 & 0 \\ 0 & \sigma & 0 \\ 0 & 0 & 0 \end{bmatrix} V^T$$

*Scale* (blue arrow pointing to the diagonal elements)

*Loss of Rank* (red box around the bottom-right element)

$$Z = \begin{bmatrix} 0 & 1 & 0 \\ -1 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix}, W = \begin{bmatrix} 0 & -1 & 0 \\ 1 & 0 & 0 \\ 0 & 0 & 1 \end{bmatrix}, ZW = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 0 \end{bmatrix}$$

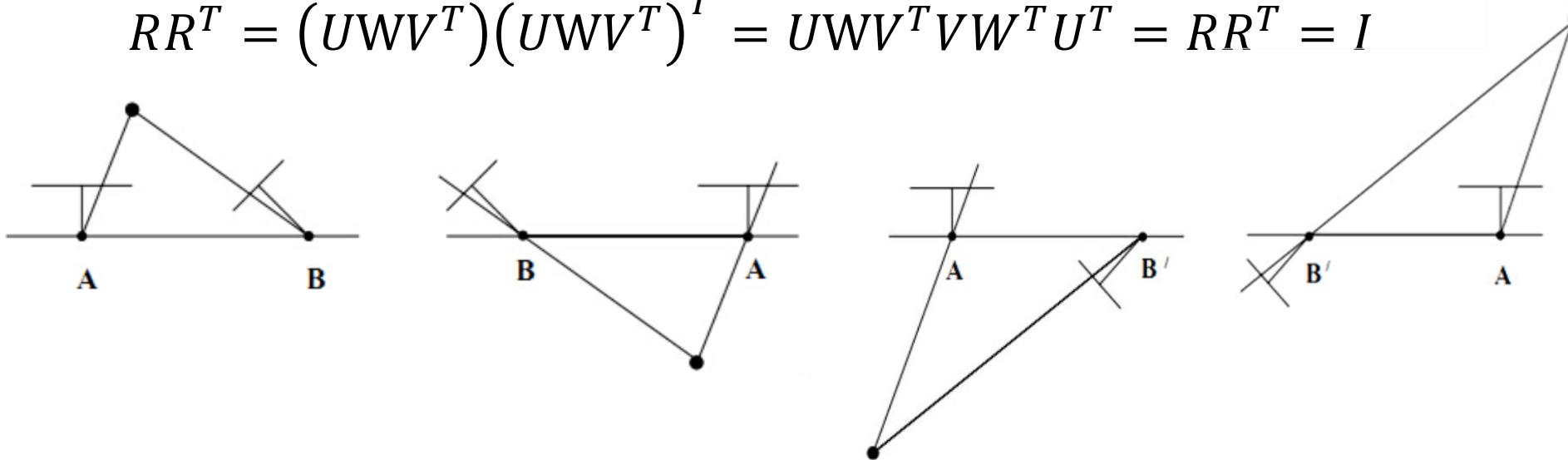
$$E = UZ WV^T = \underbrace{(UZU^T)}_{[t]_{\times}} \underbrace{(UWV^T)}_R = [t]_{\times}R$$

# Unknown Structure Initialization

## Essential Matrix Decomposition

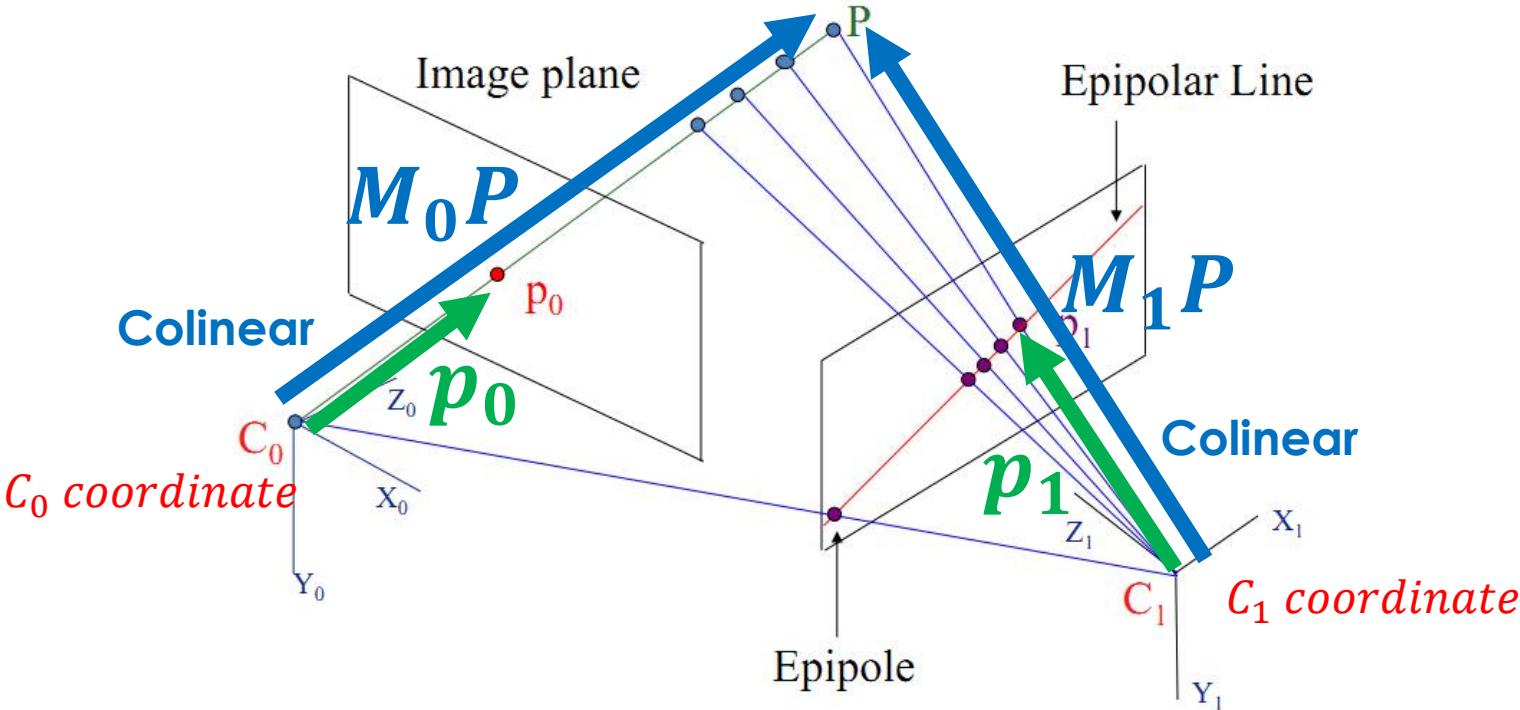
$$\begin{bmatrix} 0 & u_{33} & -u_{23} \\ -u_{33} & 0 & u_{13} \\ u_{23} & -u_{13} & 0 \end{bmatrix} \longrightarrow [\mathbf{t}]_\times$$

$$RR^T = (UWV^T)(UWV^T)^T = UWV^TVW^TU^T = RR^T = I$$



# Unknown Structure Initialization

## 3D Structure Recovering (Triangulation)

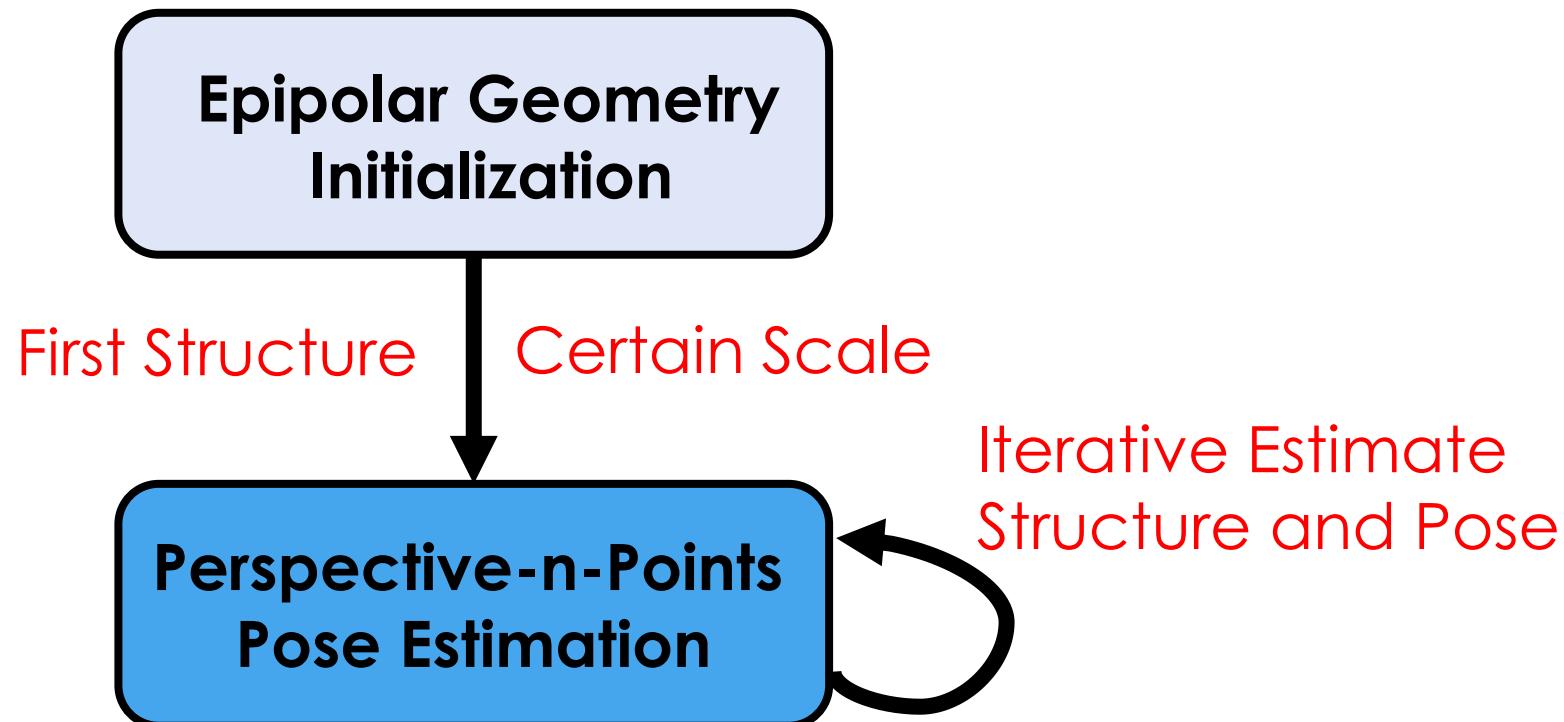


$$M_0 = [R_0 | t_0] = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix}, M_1 = [R_1 | t_1]$$

$$\begin{aligned} p_0 \times M_0 P &= 0 \\ p_1 \times M_1 P &= 0 \end{aligned} \rightarrow \begin{pmatrix} [(p_0)_\times] M_0 \\ [(p_1)_\times] M_1 \end{pmatrix} P = 0$$

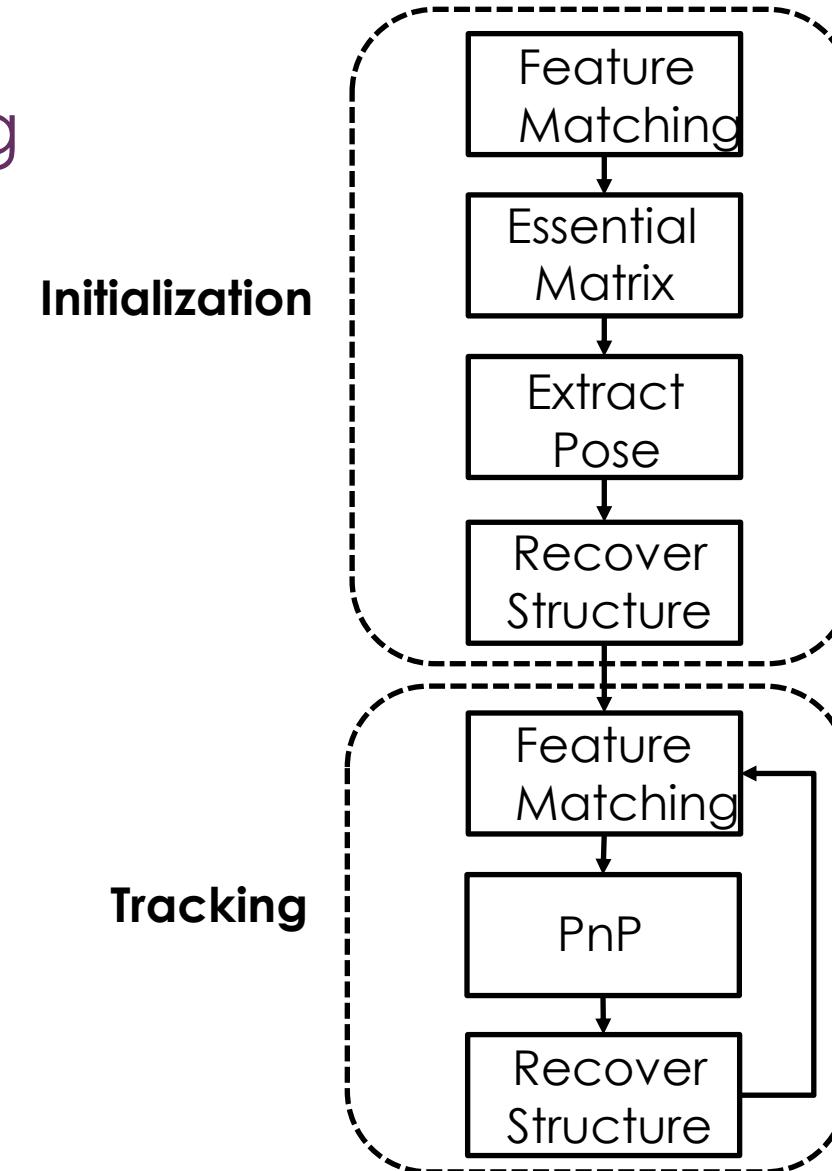
# Basic Initialization and Tracking

**Concept**



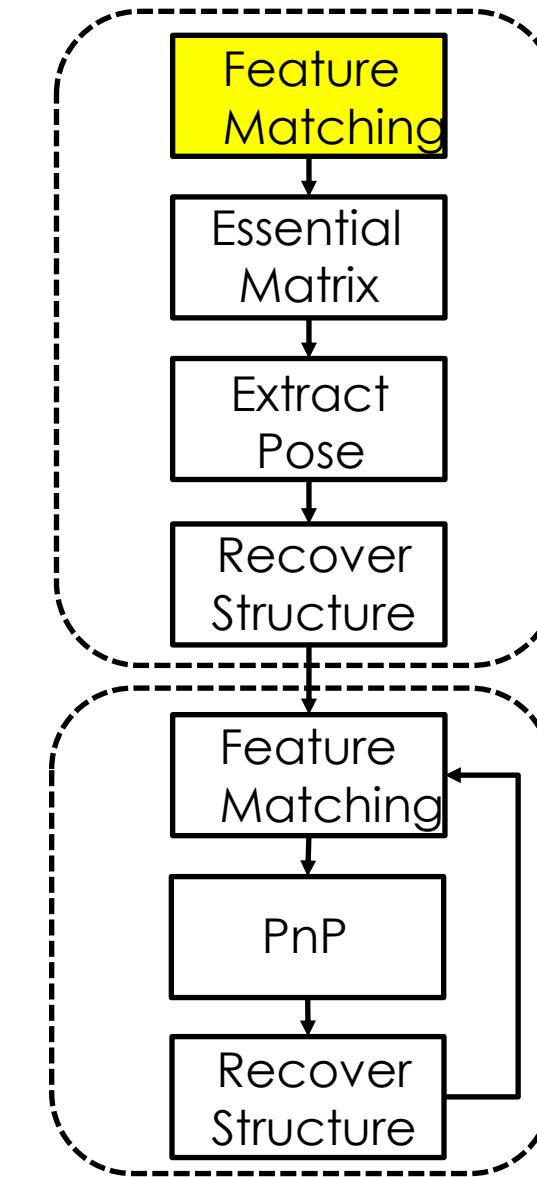
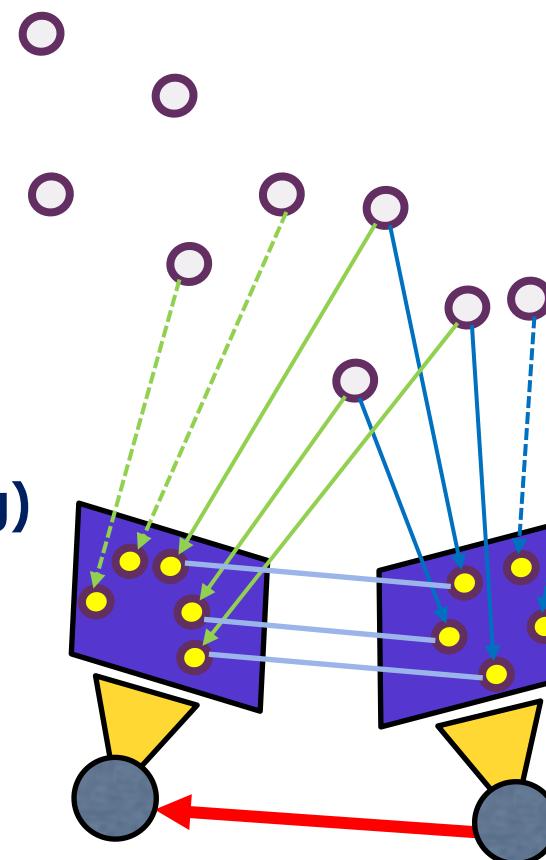
Use known structure points to estimate the pose then compute new points by triangulation.

# Basic Initialization and Tracking



# Basic Initialization and Tracking

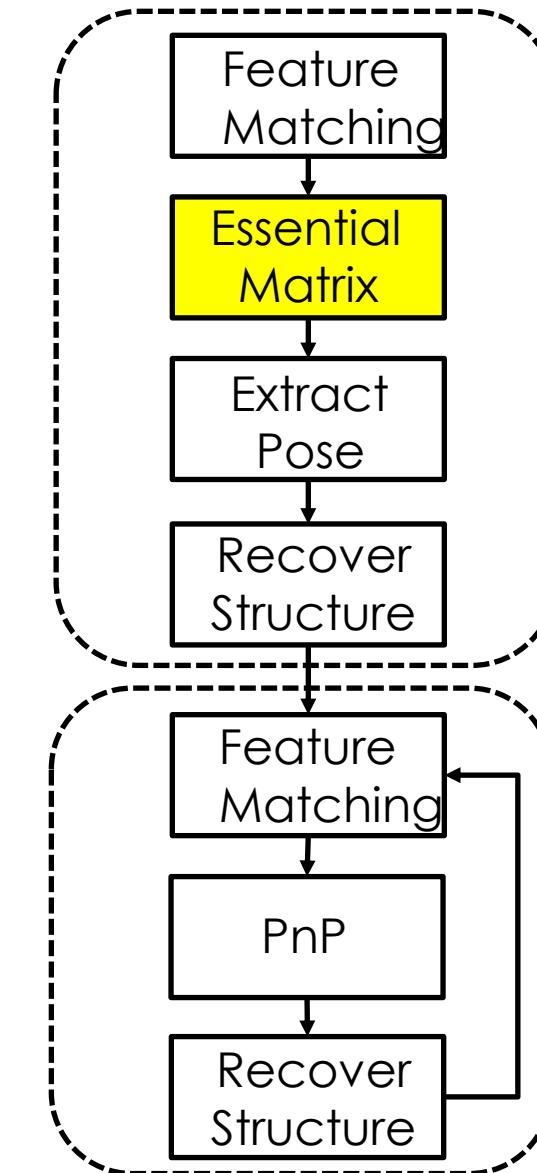
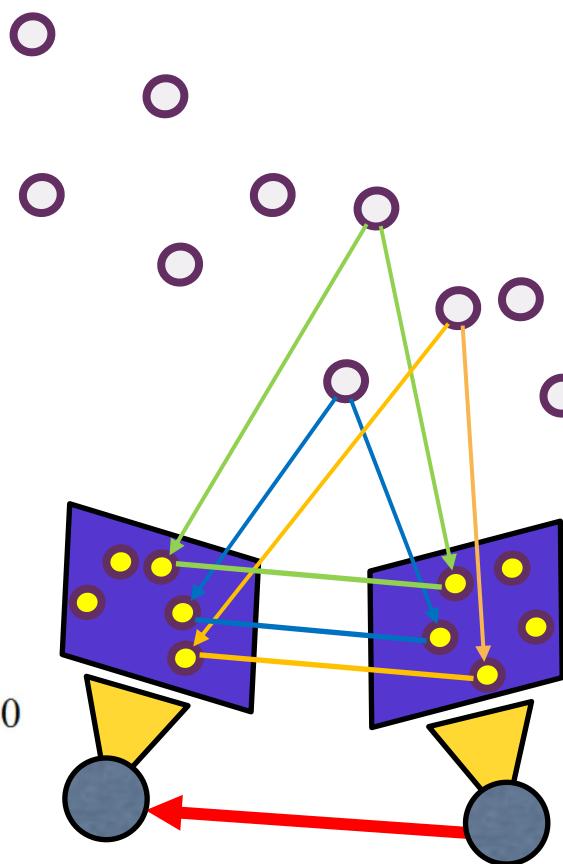
Initialization  
(Feature Matching)



# Basic Initialization and Tracking

## Initialization (Essential Matrix)

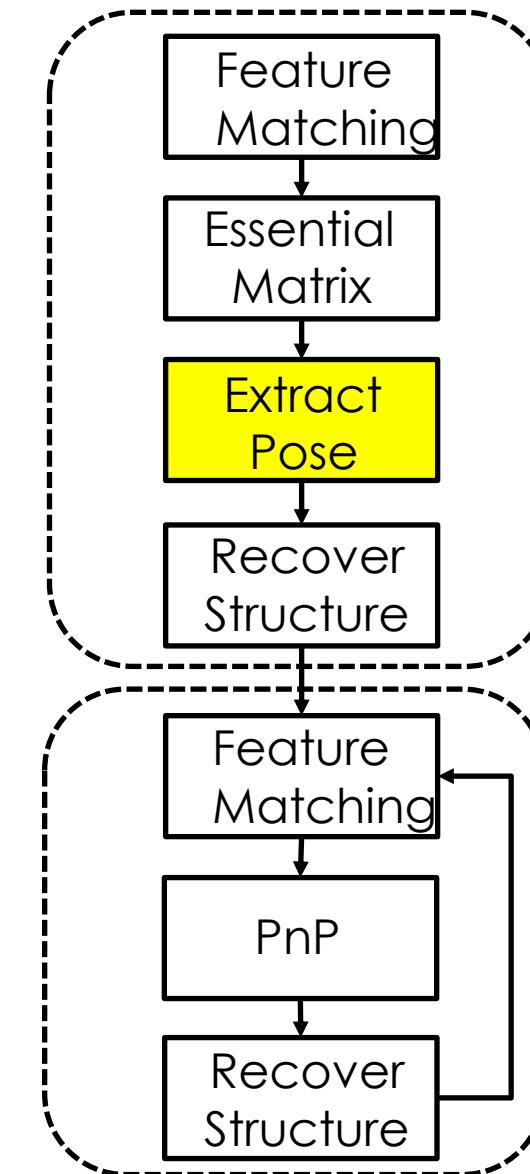
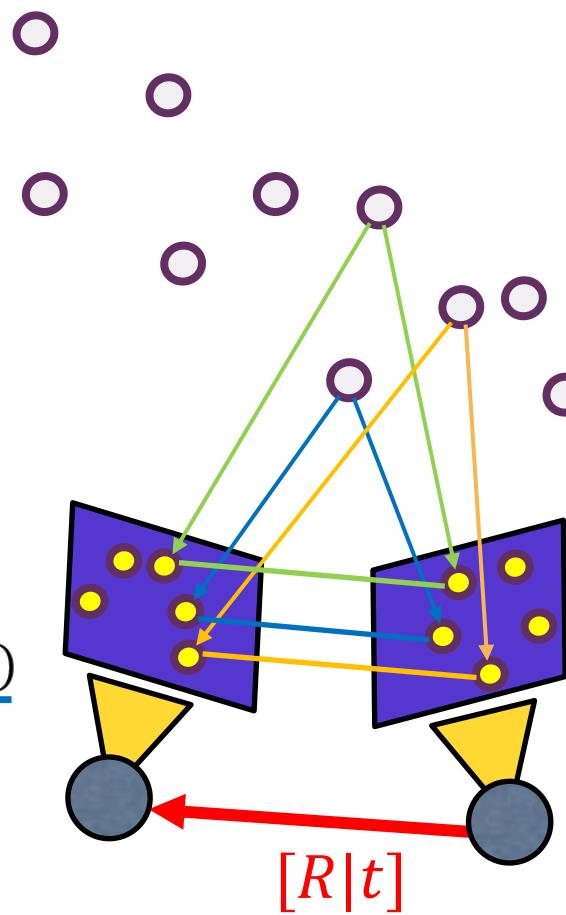
$$(x_0 \quad y_0 \quad 1) \begin{pmatrix} E_{11} & E_{12} & E_{13} \\ E_{21} & E_{22} & E_{23} \\ E_{31} & E_{32} & E_{33} \end{pmatrix} \begin{pmatrix} x_1 \\ y_1 \\ 1 \end{pmatrix} = 0$$



# Basic Initialization and Tracking

Initialization  
(Extract Pose)

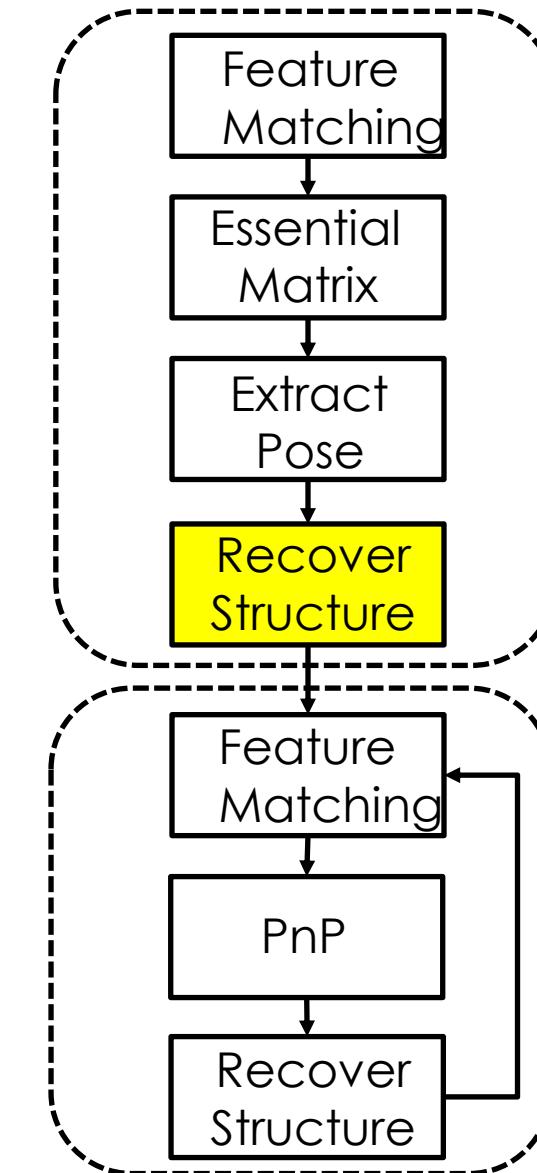
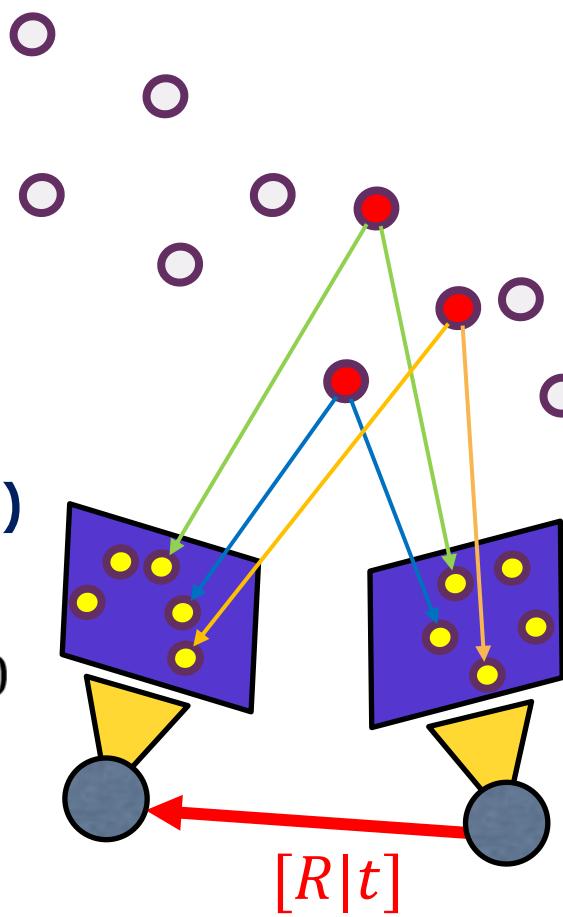
$$E = UZWV^T = \frac{(UZU^T)}{S} \frac{(UWV^T)}{R}$$



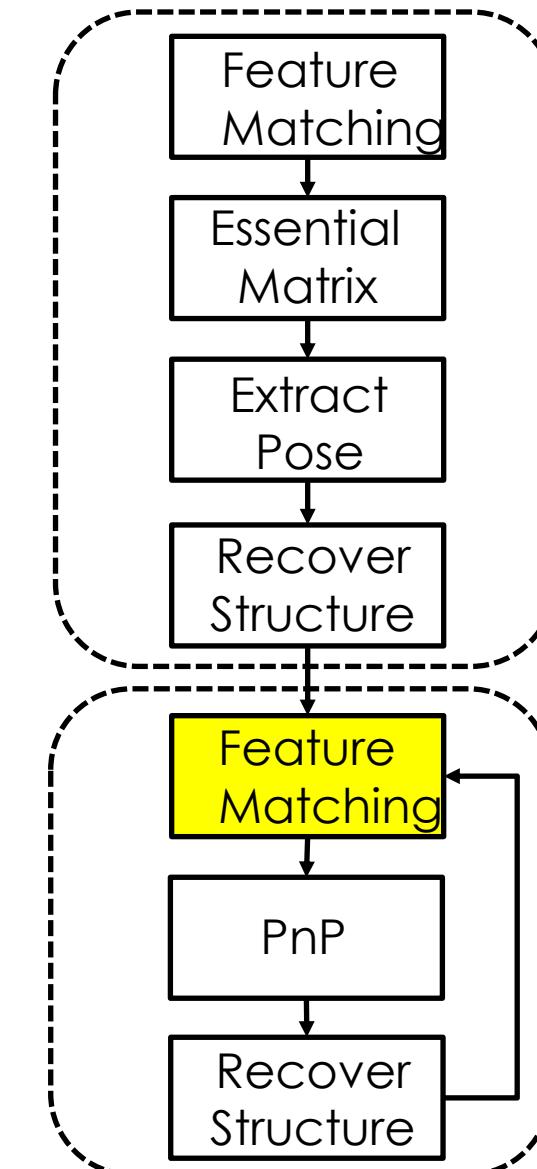
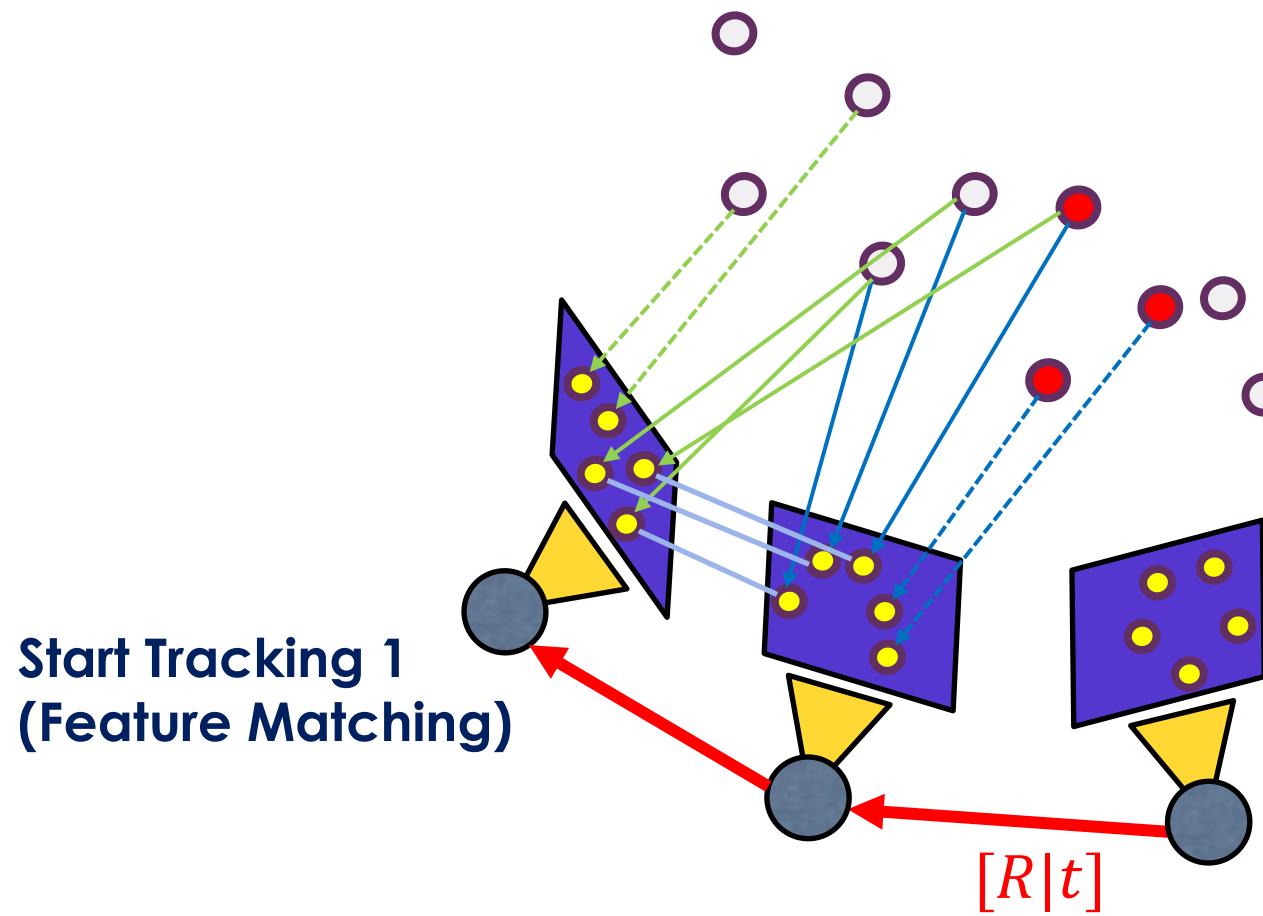
# Basic Initialization and Tracking

Initialization  
(Recover Structure)

$$\begin{pmatrix} [(P_0)_x]M_0 \\ [(P_1)_x]M_1 \end{pmatrix} P = 0$$



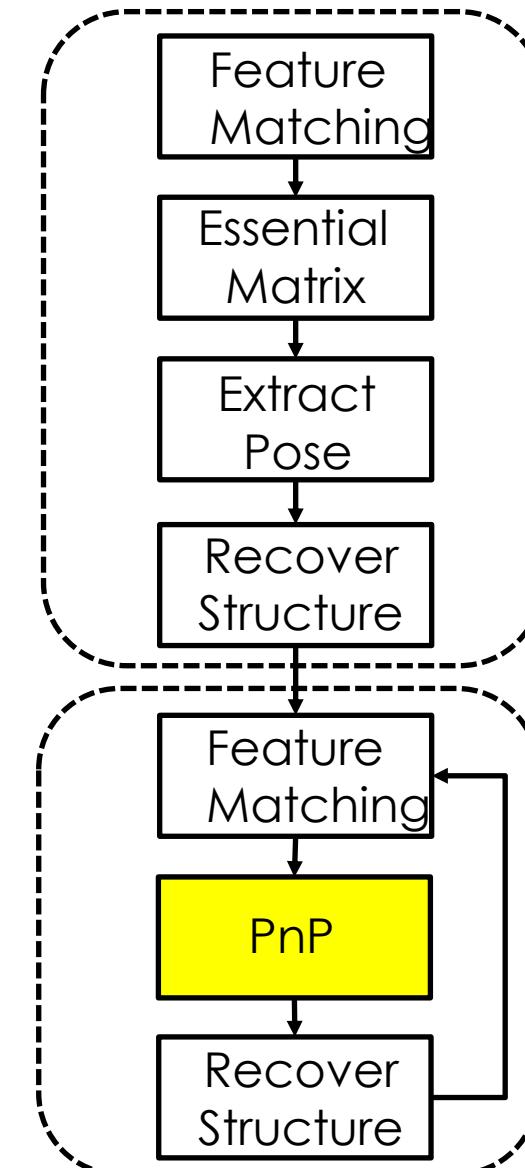
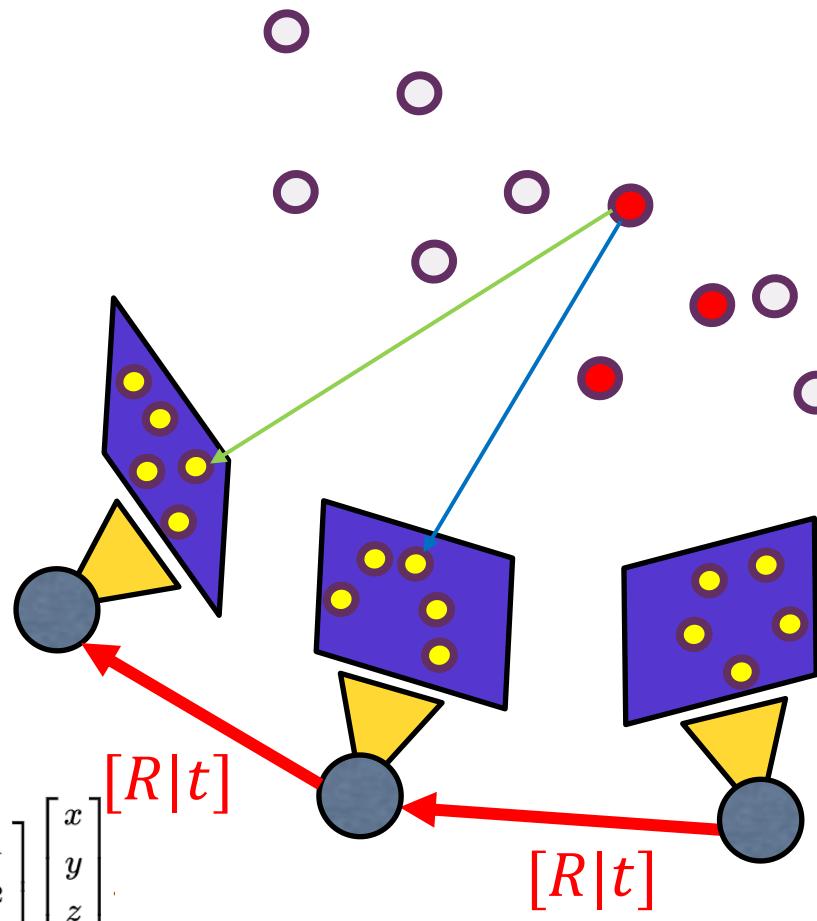
# Basic Initialization and Tracking



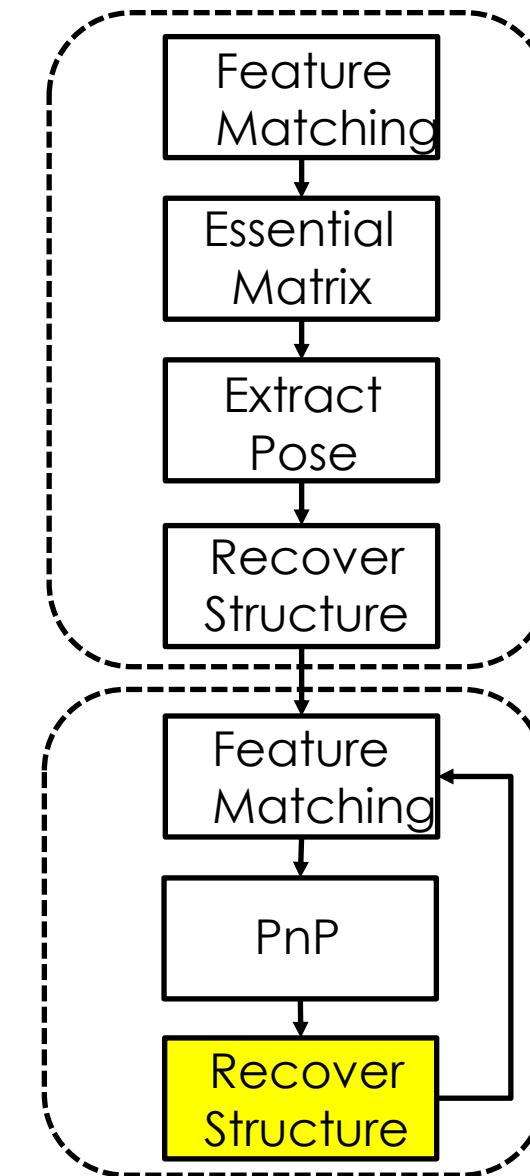
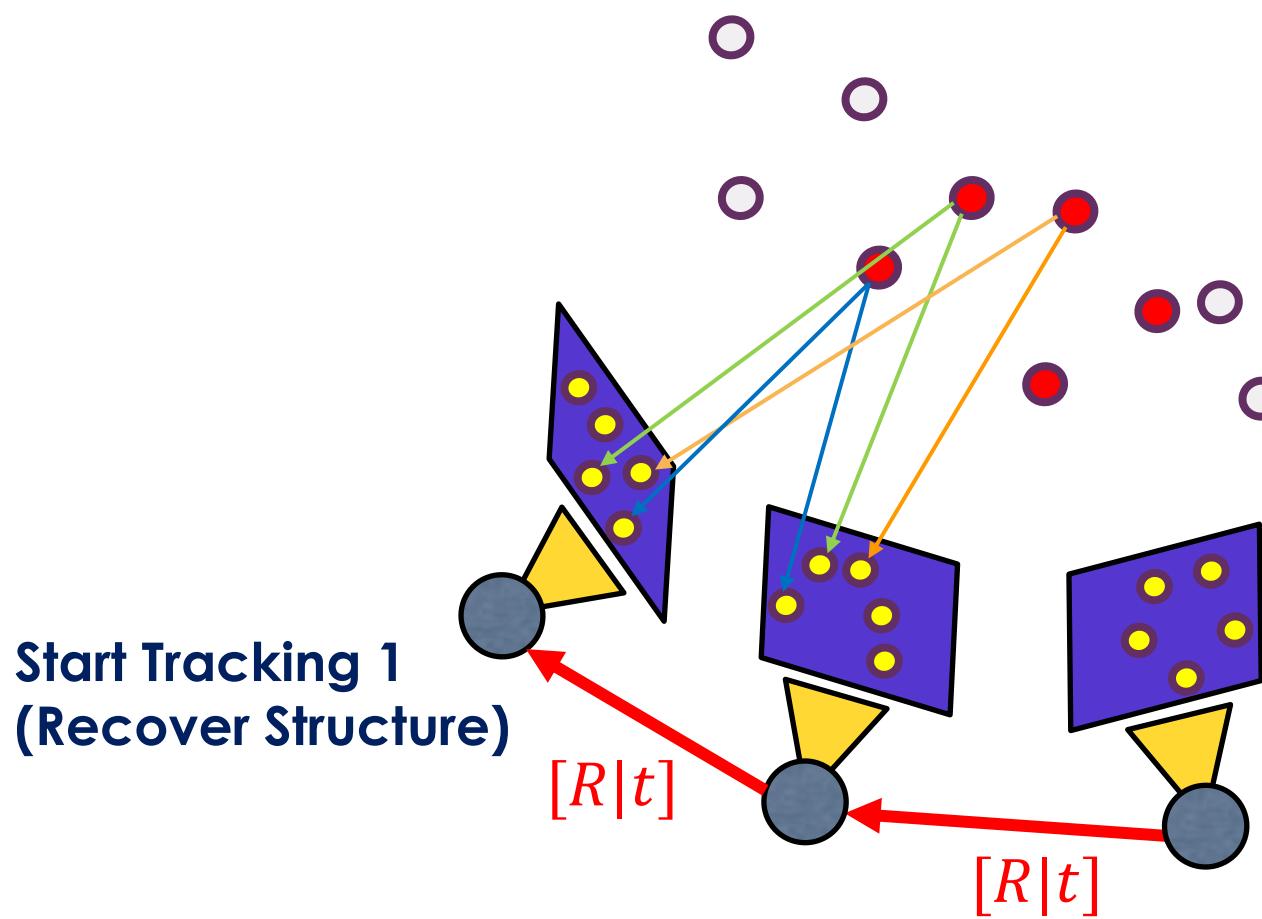
# Basic Initialization and Tracking

Start Tracking 1  
(PnP)

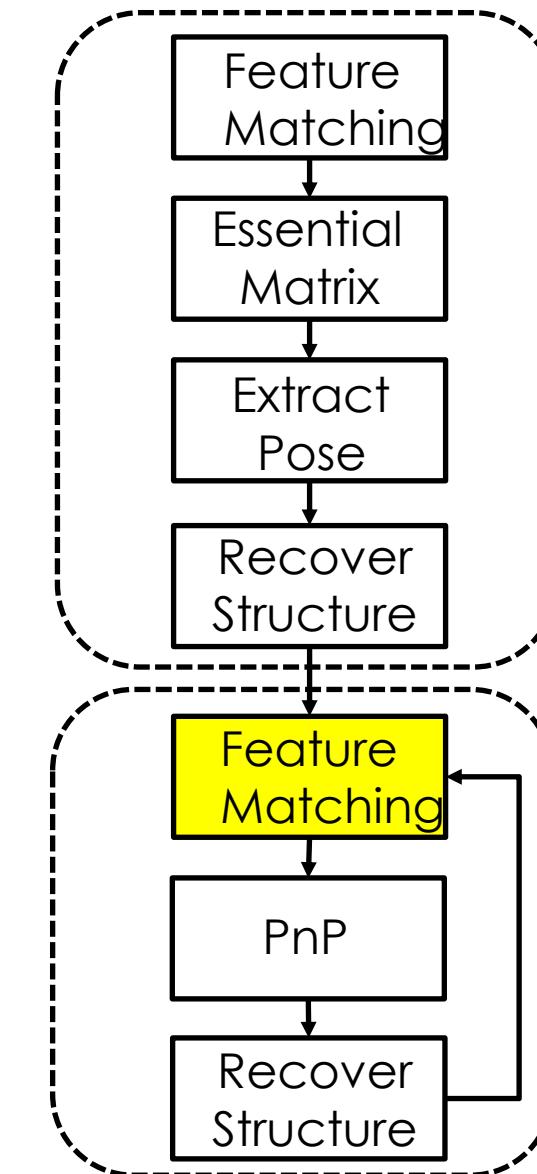
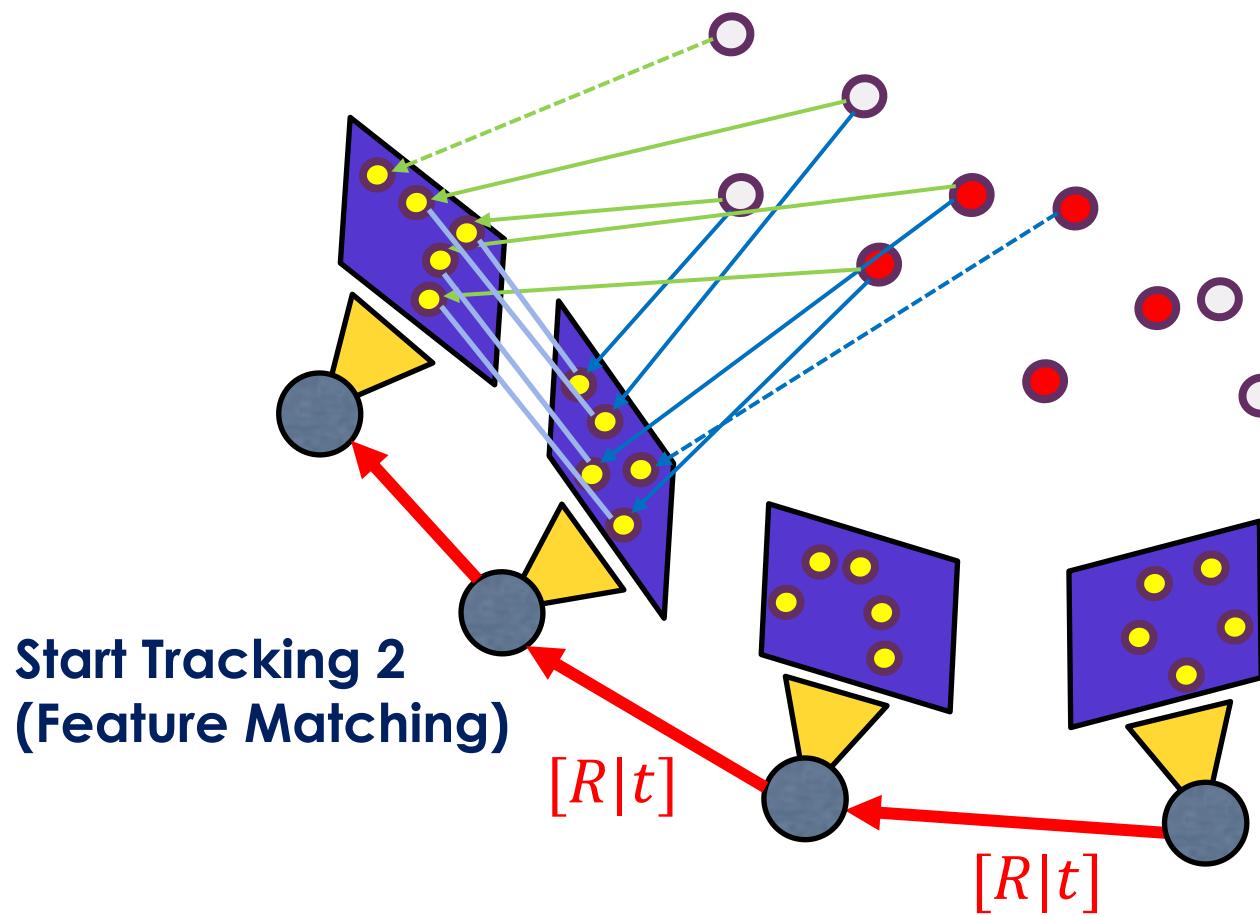
$$s \begin{bmatrix} u \\ v \\ 1 \end{bmatrix} = \begin{bmatrix} f_x & \gamma & u_0 \\ 0 & f_y & v_0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} r_{11} & r_{12} & r_{13} & t_1 \\ r_{21} & r_{22} & r_{23} & t_2 \\ r_{31} & r_{32} & r_{33} & t_3 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix}$$



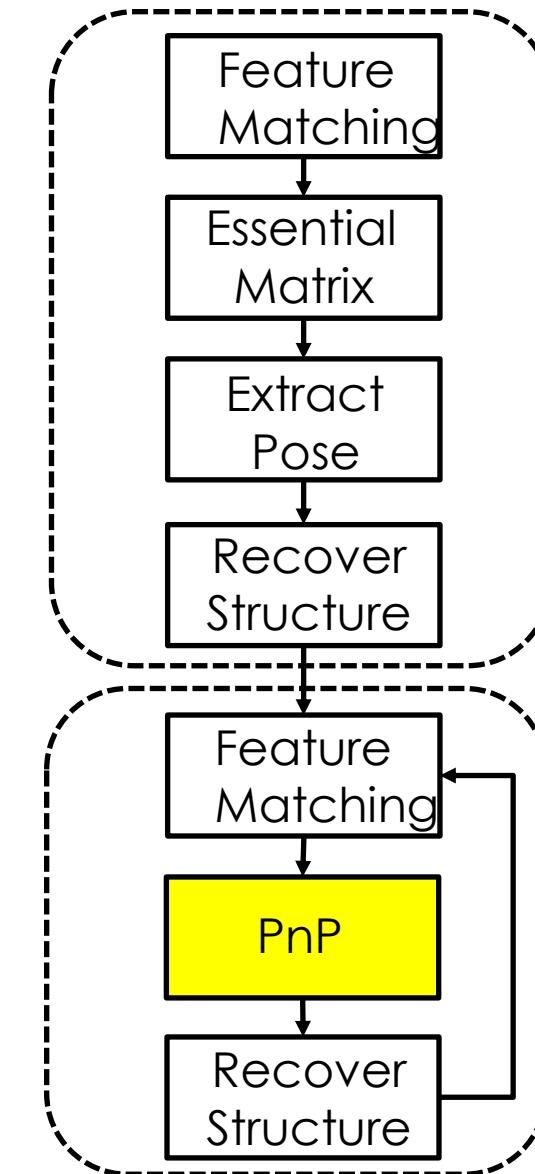
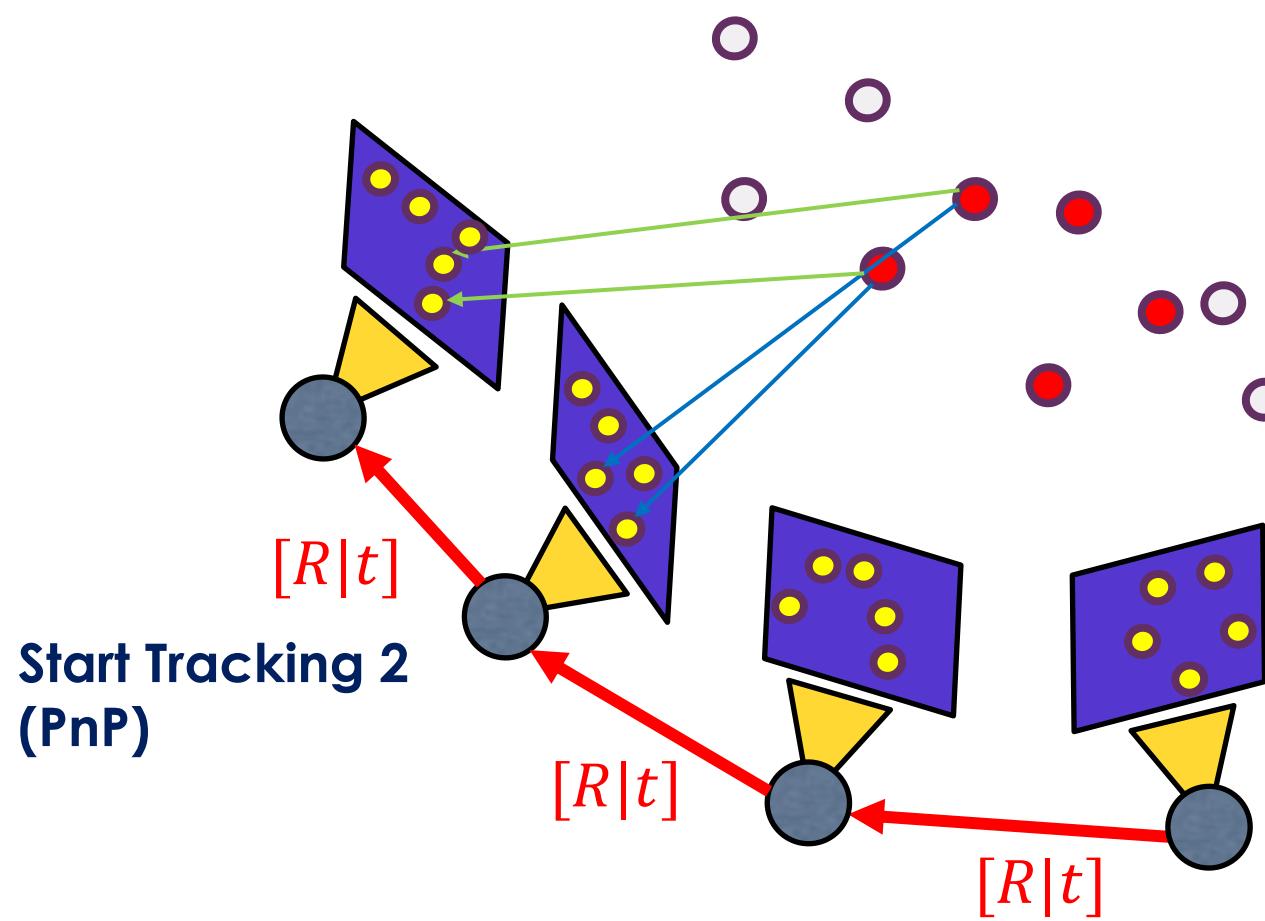
# Basic Initialization and Tracking



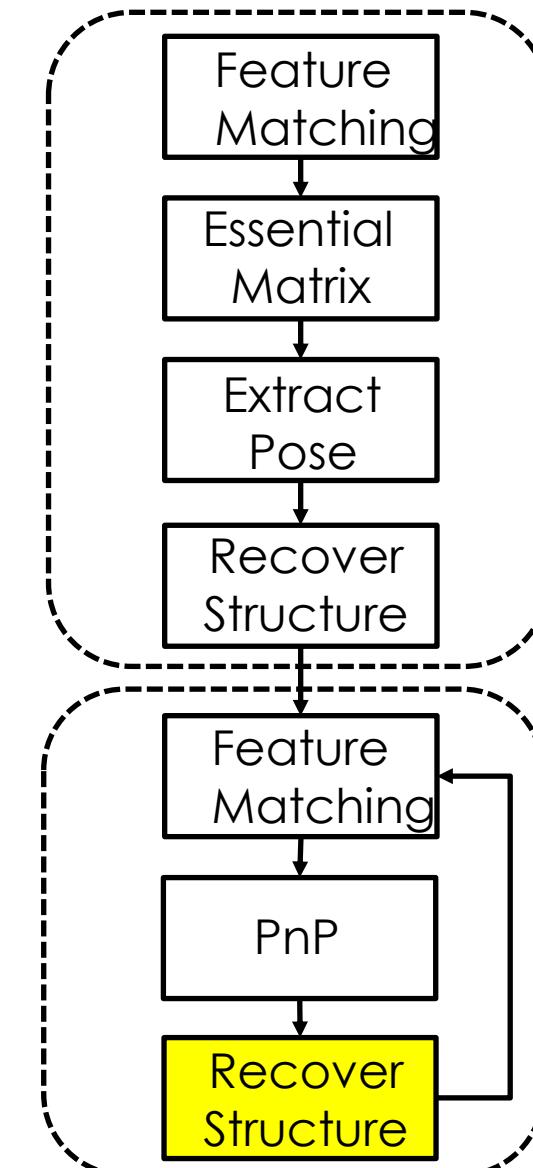
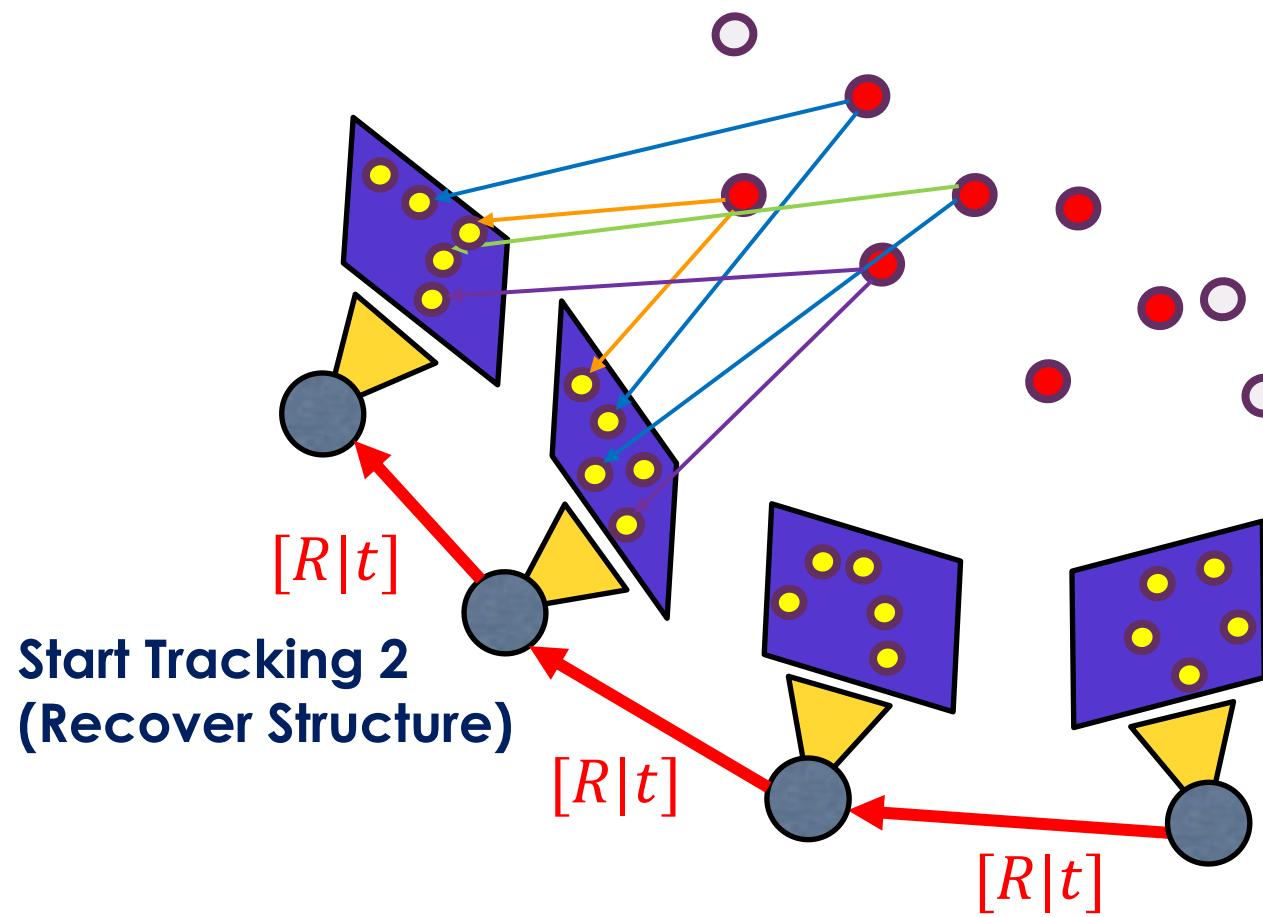
# Basic Initialization and Tracking



## Basic Initialization and Tracking

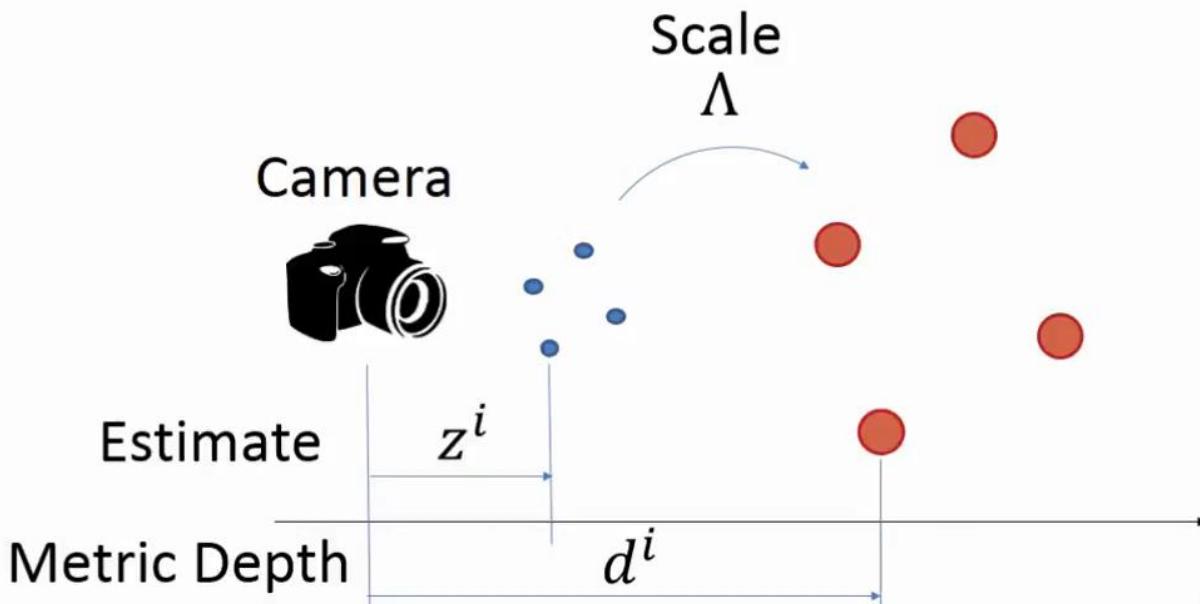


# Basic Initialization and Tracking



# Scaling Drift Problem

Scale factor defines the relationship between the estimated geometry and the metric map



# Image Matching

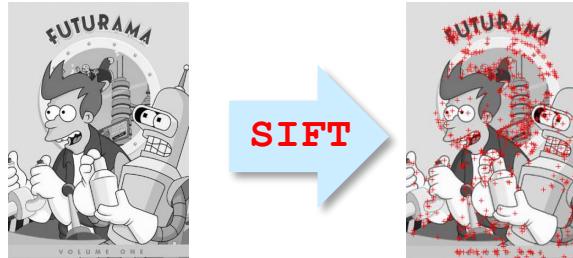
- How do we detect an object in an image?



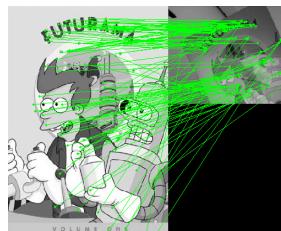
- Combines ideas from **image transformations**, **least squares**, and **robustness**

# Object Matching in Three Steps

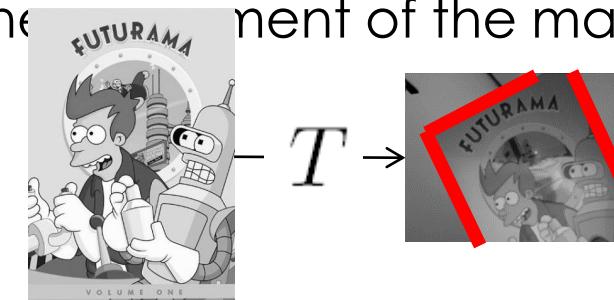
1. Detect features in the template and search images



2. Match features: find “similar-looking” features in the two images



3. Find a transformation  $T$  that explains the alignment of the matched features



## Affine Transformations

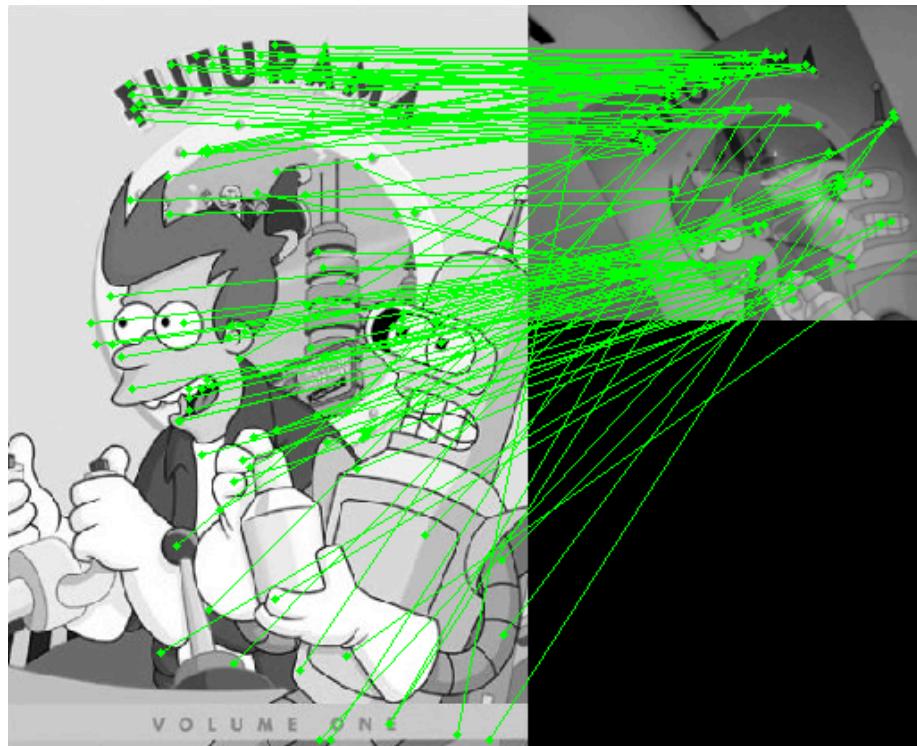
- A 2D affine transformation has the form:

$$T = \begin{bmatrix} a & b & c \\ d & e & f \\ 0 & 0 & 1 \end{bmatrix}$$

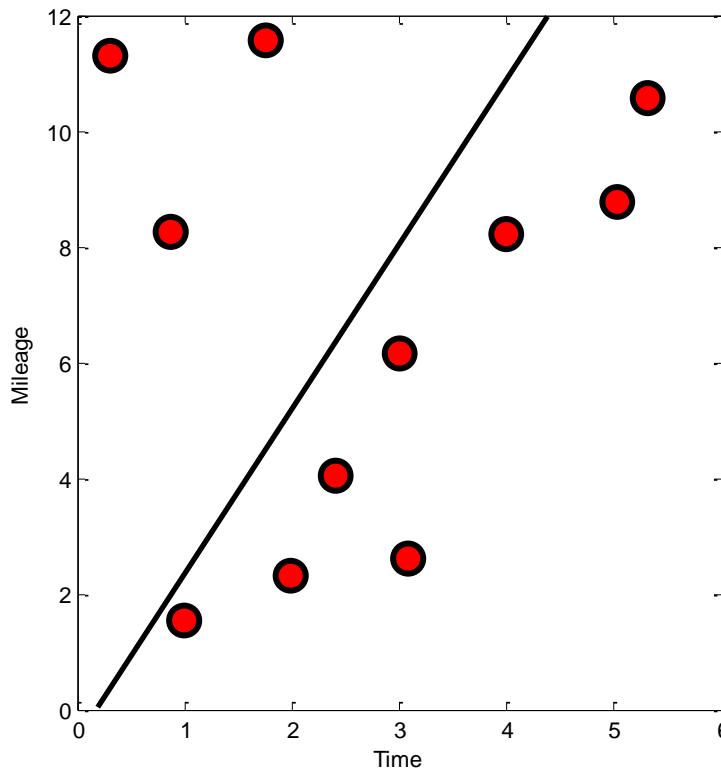
$$\begin{bmatrix} a & b & c \\ d & e & f \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix} = \begin{bmatrix} ax + by + c \\ dx + ey + f \\ 1 \end{bmatrix}$$

# Fitting affine transformations

- We will fit an affine transformation to a set of feature matches
  - Problem: there are many incorrect matches

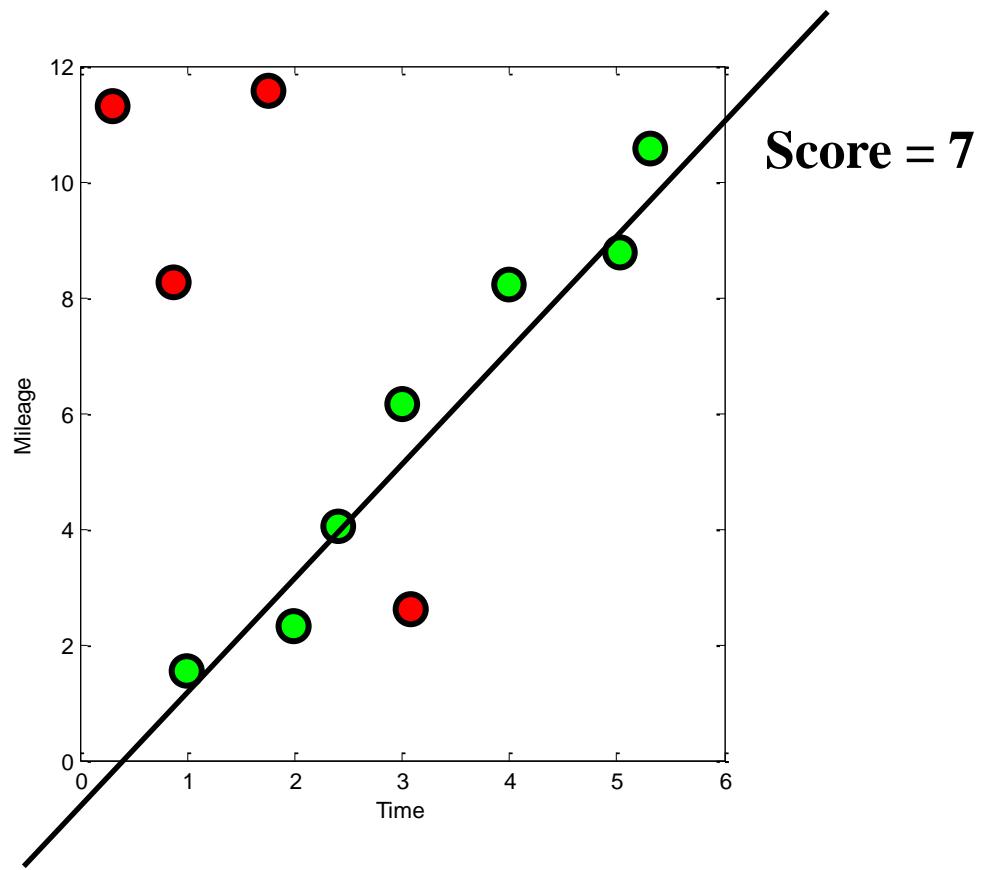


# Linear Regression



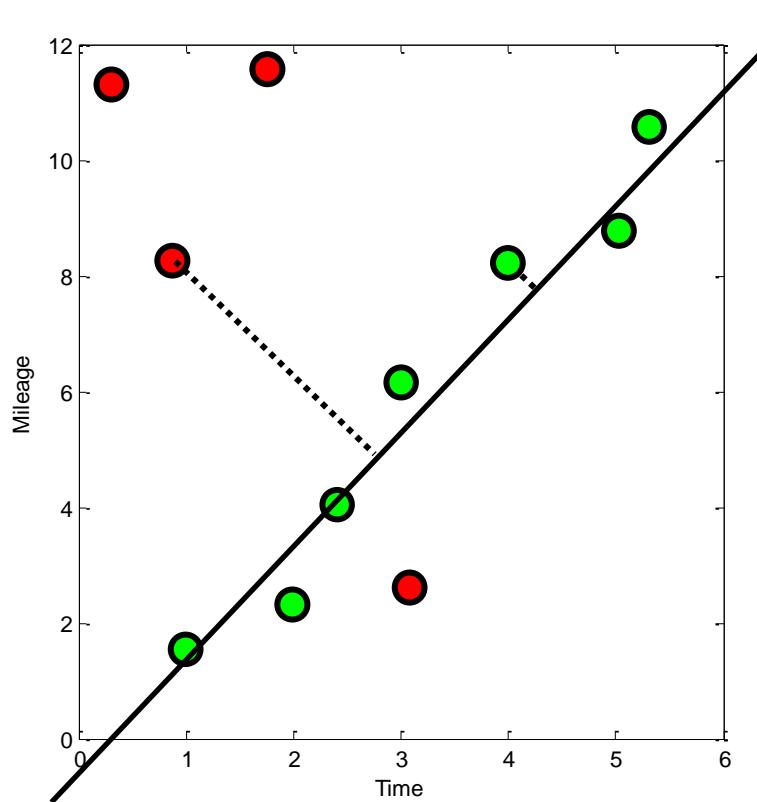
# Testing Goodness

- Idea: count the number of points that are “close” to the line



# Testing Goodness

- How can we tell if a point agrees with a line?
  - Compute the distance the point and the line, and threshold

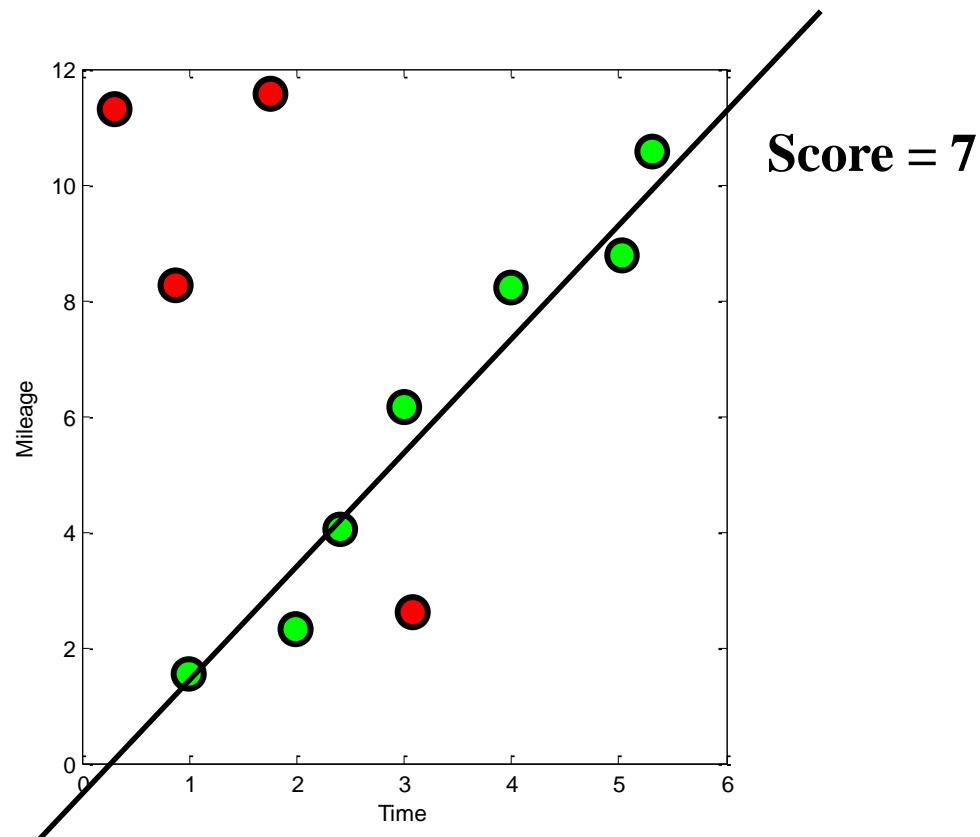


## Testing Goodness

- If the distance is small, we call this point an *inlier* to the line
  - For an inlier point and a good line, this distance will be close to (but not exactly) zero
- If the distance is large, it's an *outlier* to the line
  - For an outlier point or bad line, this distance will probably be large
- Objective function: find the line with the most inliers (or the fewest outliers)

# Optimizing for Inlier Count

- How do we find the best possible line?



# Algorithm (RANSAC)

1. Select two points at random
2. Solve for the line ( $L$ ) between these two points
3. Count the number of inliers to the line  $L$
4. If  $L$  has the highest number of inliers so far, save it
5. Repeat for  $N$  rounds, return the best  $L$

# Testing goodness

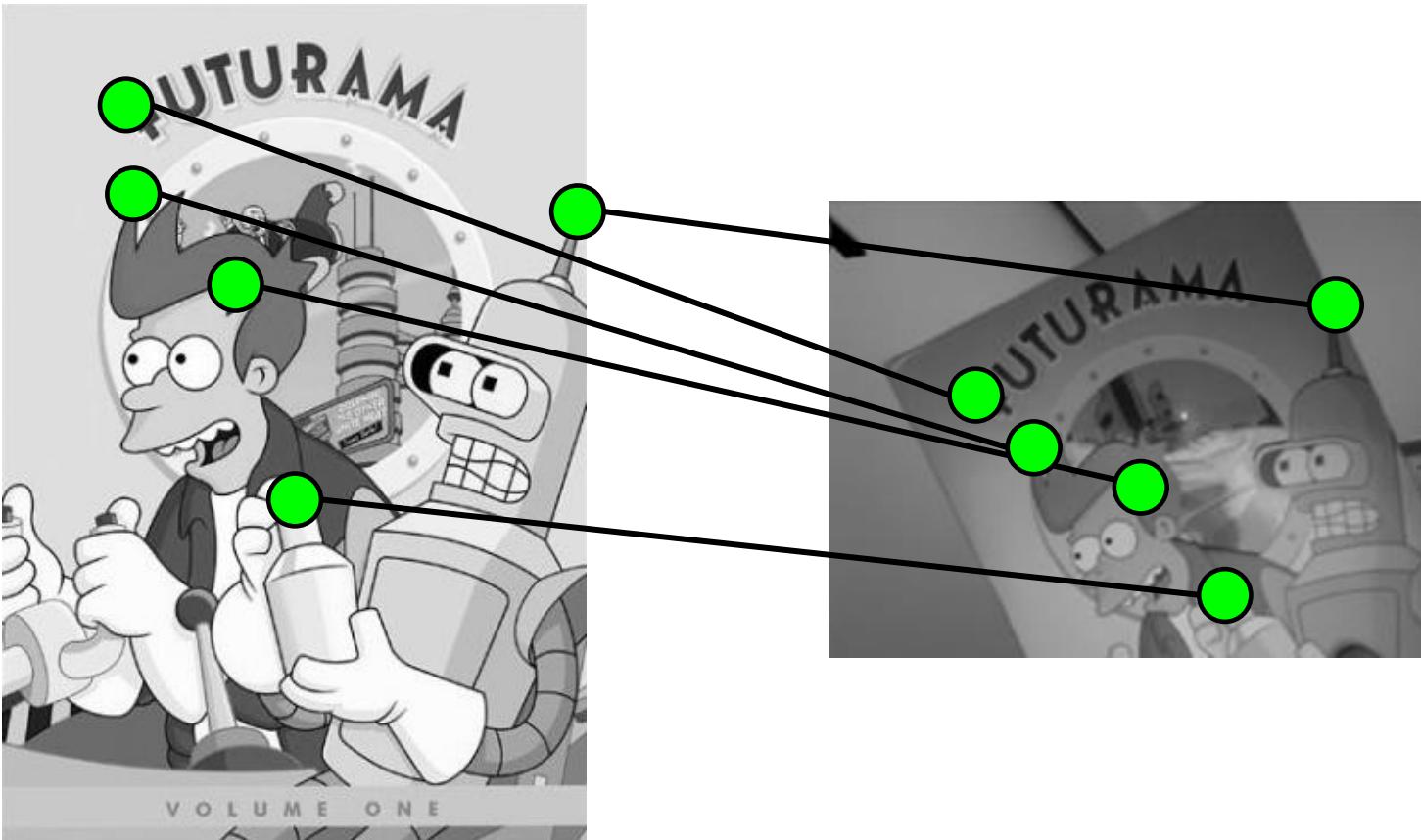
- This algorithm is called RANSAC (RANdom SAmple Consensus) – example of a randomized algorithm
- Used in an amazing number of computer vision algorithms
- Requires two parameters:
  - The agreement threshold (how close does an inlier have to be?)
  - The number of rounds (how many do we need?)

# Randomized algorithms

- Very common in computer science
  - In this case, we avoid testing an infinite set of possible lines, or all  $O(n^2)$  lines generated by pairs of points
- These algorithms find the right answer with some probability
- Often work very well in practice

## Very Similar Idea

- Given two images with a set of feature matches, how do we compute an affine transform between the two images?



# Multi-variable Fitting

- Let's consider a 2D affine transformation that maps a 2D point to another 2D point

$$T = \begin{bmatrix} a & b & c \\ d & e & f \\ 0 & 0 & 1 \end{bmatrix}$$

- We have a set of n matches

$$[x_1 \ y_1] \rightarrow [x'_1 \ y'_1]$$

$$[x_2 \ y_2] \rightarrow [x'_2 \ y'_2]$$

$$[x_3 \ y_3] \rightarrow [x'_3 \ y'_3]$$

...

$$[x_n \ y_n] \rightarrow [x'_n \ y'_n]$$

## Fitting an Affine Transformation

- Consider just one match  $[x_1 \ y_1] \rightarrow [x'_1 \ y'_1]$

$$\begin{bmatrix} a & b & c \\ d & e & f \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x_1 \\ y_1 \\ 1 \end{bmatrix} = \begin{bmatrix} x'_1 \\ y'_1 \\ 1 \end{bmatrix}$$

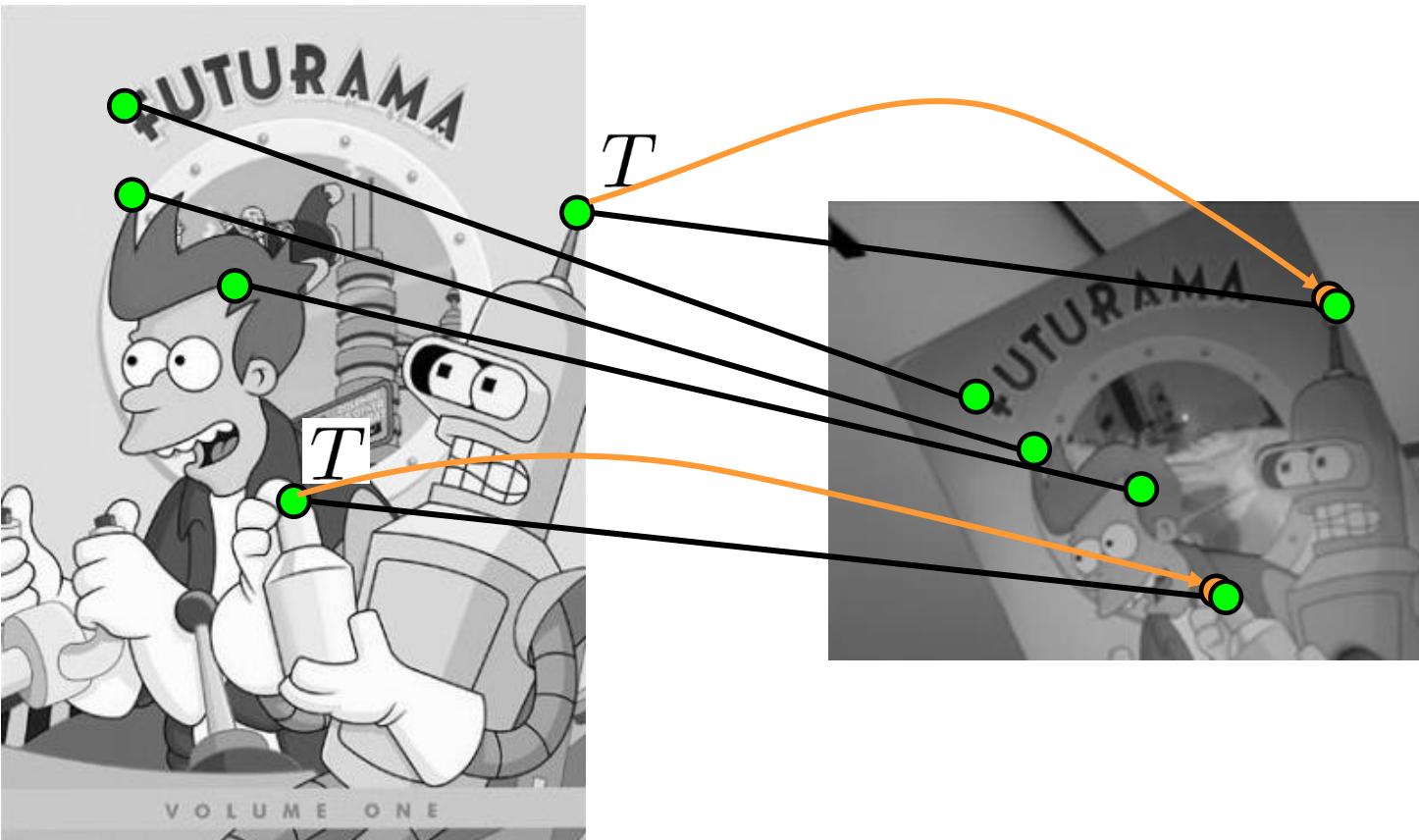
$$ax_1 + by_1 + c = x'_1$$

$$dx_1 + ey_1 + f = y'_1$$

- 2 equations, 6 unknowns  
→ we need at least 3 matches, but can fit n using least squares

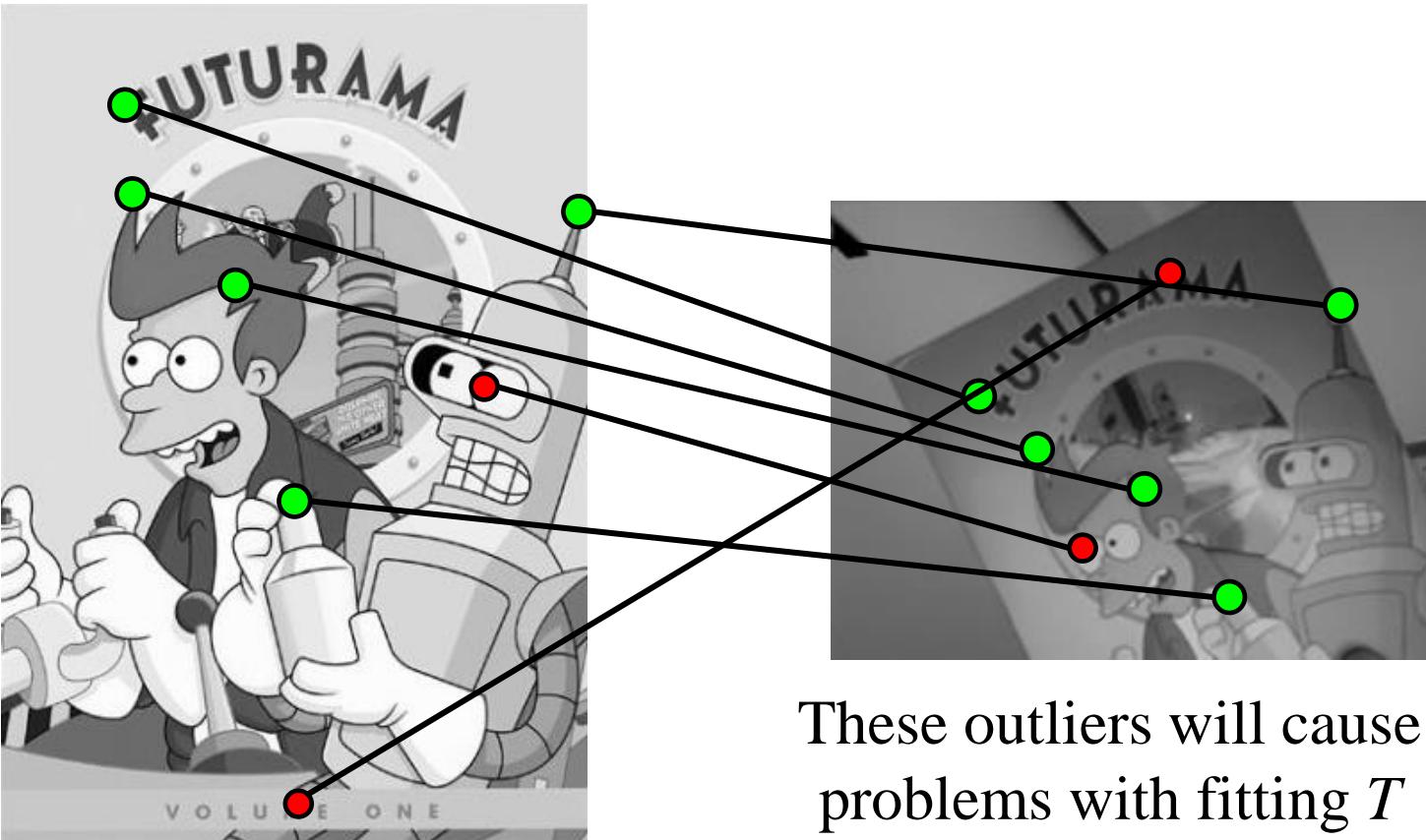
# Fitting an Affine Transformation

- Find 2D affine transformation  $T$  that maps points in image 1 as close as possible to their matches in image 2



## Back to Fitting

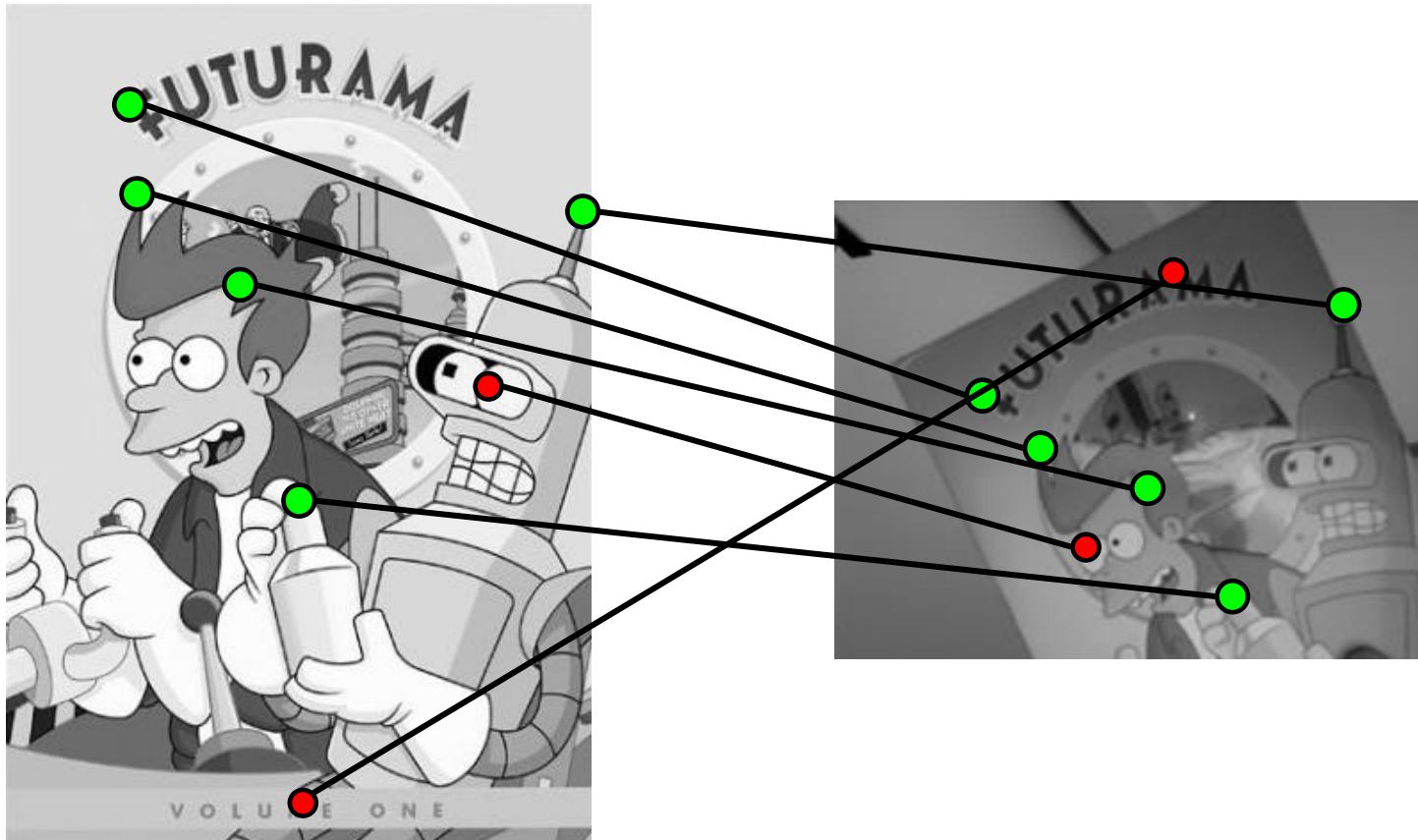
- Just like in the case of fitting a line, we have some bad data (incorrect matches)



These outliers will cause problems with fitting  $T$

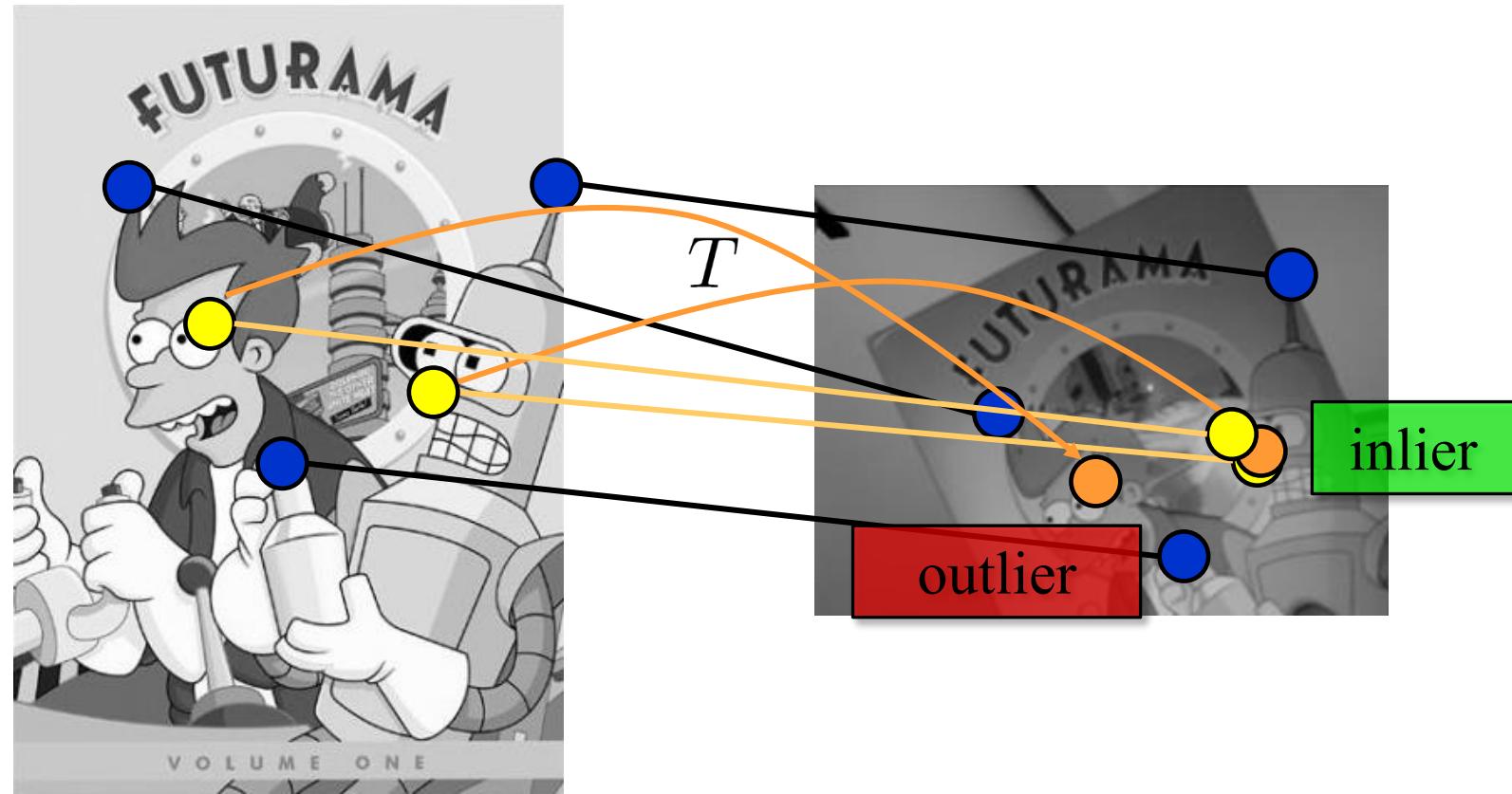
# How Do We Fix This?

- RANSAC to the rescue!



# Generate and Test an Affine Transformation

- From set of matches, select 3 at random
- Fit a transformation  $T$  to the selected matches
- Count inliers



# Transform Fitting Algorithm (RANSAC)

1. Select three matches at random
2. Solve for the affine transformation  $T$
3. Count the number of matches that are inliers to  $T$
4. If  $T$  has the highest number of inliers so far, save it
5. Repeat for  $N$  rounds, return the best  $T$

## How do we solve for T given 3 matches?

- Three matches give a linear system with six equations:

$$[x_1 \ y_1] \rightarrow [x'_1 \ y'_1] \quad \begin{aligned} ax_1 + by_1 + c &= x'_1 \\ dx_1 + ey_1 + f &= y'_1 \end{aligned}$$

$$[x_2 \ y_2] \rightarrow [x'_2 \ y'_2] \quad \begin{aligned} ax_2 + by_2 + c &= x'_2 \\ dx_2 + ey_2 + f &= y'_2 \end{aligned}$$

$$[x_3 \ y_3] \rightarrow [x'_3 \ y'_3] \quad \begin{aligned} ax_3 + by_3 + c &= x'_3 \\ dx_3 + ey_3 + f &= y'_3 \end{aligned}$$

## Two 3x3 linear systems

$$ax_1 + by_1 + c = x_1'$$

$$ax_2 + by_2 + c = x_2'$$

$$ax_3 + by_3 + c = x_3'$$

$$dx_1 + ey_1 + f = y_1'$$

$$dx_2 + ey_2 + f = y_2'$$

$$dx_3 + ey_3 + f = y_3'$$

## Solving a 3x3 system

$$ax_1 + by_1 + c = x_1'$$

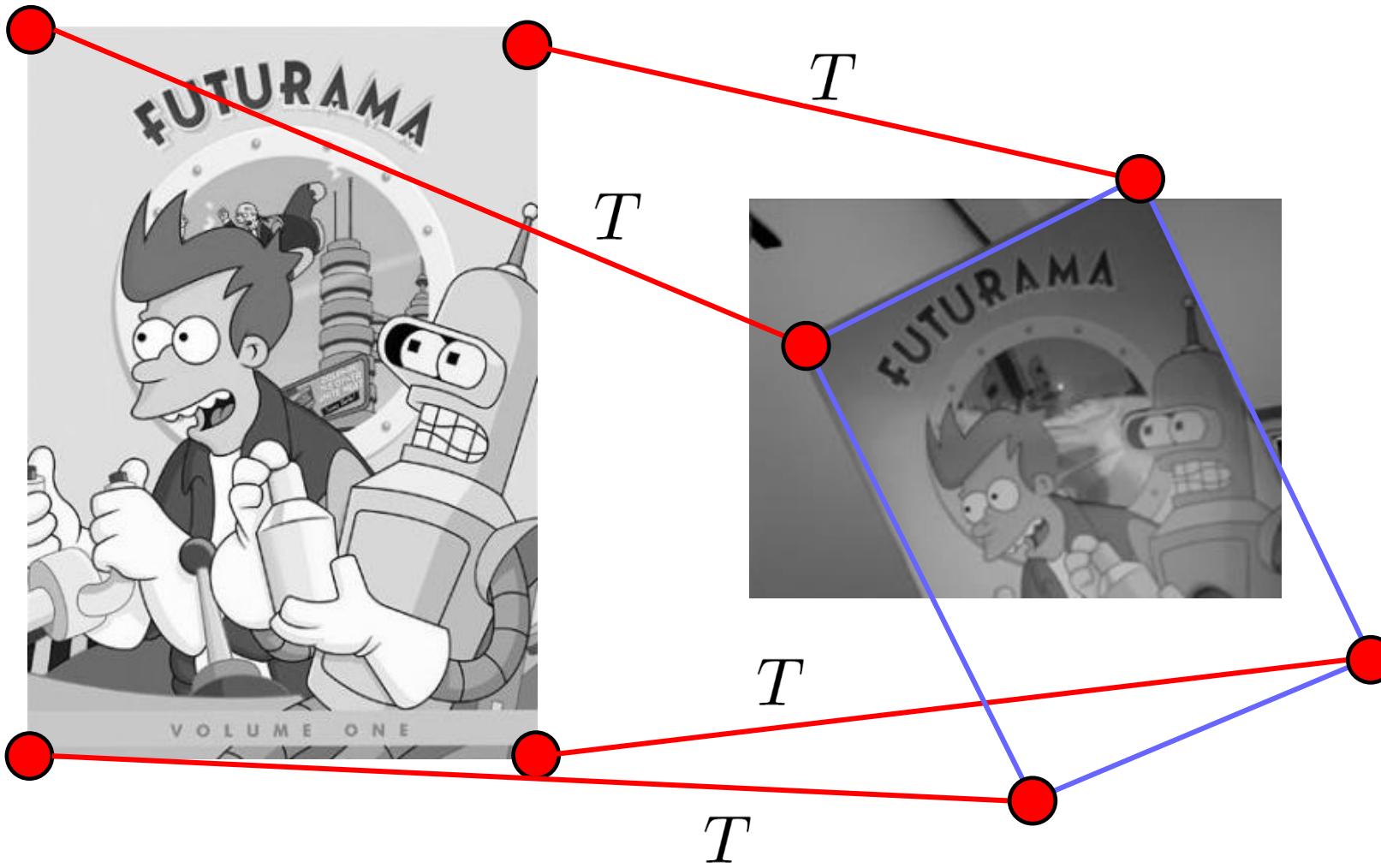
$$ax_2 + by_2 + c = x_2'$$

$$ax_3 + by_3 + c = x_3'$$

- We can write this in matrix form:

$$\begin{bmatrix} x_1 & y_1 & 1 \\ x_2 & y_2 & 1 \\ x_3 & y_3 & 1 \end{bmatrix} \begin{bmatrix} a \\ b \\ c \end{bmatrix} = \begin{bmatrix} x'_1 \\ x'_2 \\ x'_3 \end{bmatrix}$$

# Finding the Object Boundary



# RANSAC

## **RAN**dom **S**ample **C**onsensus

repeat

    select minimal sample (8 matches)

    compute solution(s) for F

    determine inliers

until  $\Gamma(\#inliers, \#samples) > 95\%$  or too many times

compute F based on all inliers

# Lie Group & Lie Algebra

# Graph Optimization for 2D Pose

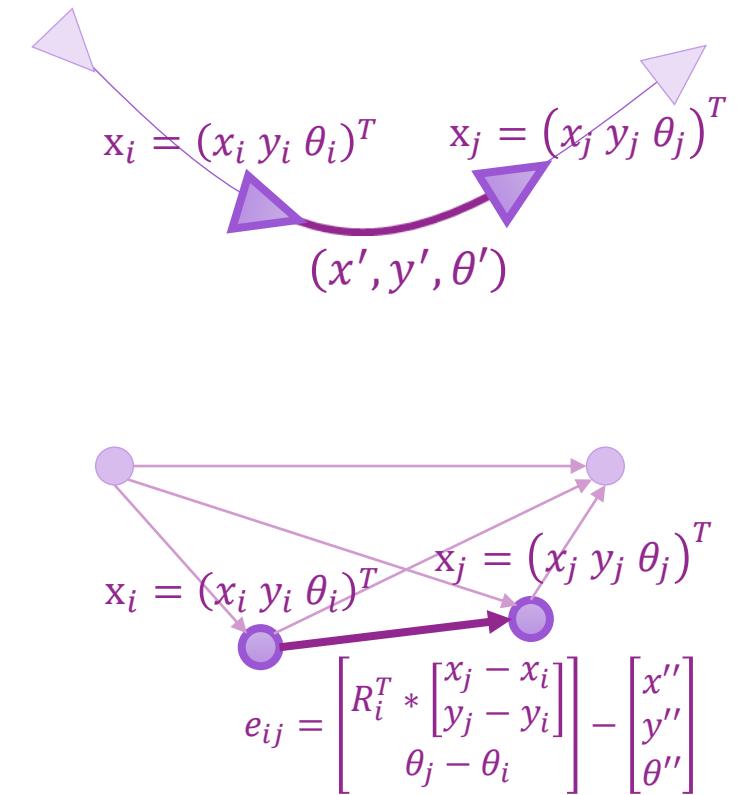
- Consider the relation between two poses:

$$\begin{bmatrix} x_j \\ y_j \\ \theta_j \end{bmatrix} = \begin{bmatrix} x_i \\ y_i \\ \theta_i \end{bmatrix} + \begin{bmatrix} R_i * \begin{bmatrix} x' \\ y' \\ \theta' \end{bmatrix} \end{bmatrix}, \text{ in which } R_i = \begin{bmatrix} \cos \theta_i & -\sin \theta_i \\ \sin \theta_i & \cos \theta_i \end{bmatrix}$$

And get  $\begin{bmatrix} x' \\ y' \\ \theta' \end{bmatrix} = \begin{bmatrix} R_i^T * \begin{bmatrix} x_j - x_i \\ y_j - y_i \\ \theta_j - \theta_i \end{bmatrix} \end{bmatrix}$

- After measuring the transform  $(x'', y'', \theta'')$  between two nodes, we can write down the error term:

$$e_{ij} = \begin{bmatrix} x' \\ y' \\ \theta' \end{bmatrix} - \begin{bmatrix} x'' \\ y'' \\ \theta'' \end{bmatrix} = \begin{bmatrix} R_i^T * \begin{bmatrix} x_j - x_i \\ y_j - y_i \\ \theta_j - \theta_i \end{bmatrix} \end{bmatrix} - \begin{bmatrix} x'' \\ y'' \\ \theta'' \end{bmatrix}$$

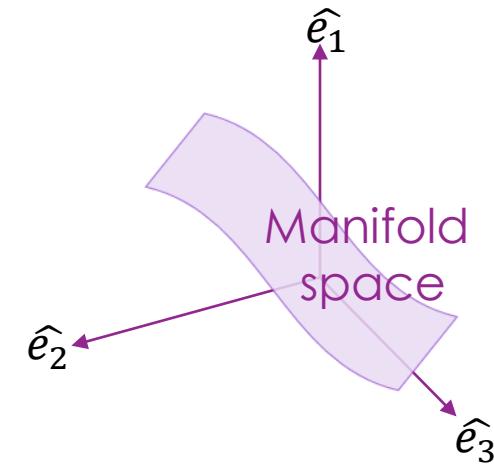


# Optimization of Transformation

- For optimization problem, the most important thing we care about is to compute the **derivative of the control parameters**.
- In 2D case, the rotation can be represented by one parameter  $\theta$ . However, the derivative of rotation becomes more complicated in 3D space.
- The key difficulties is that the **rotation matrix does not satisfy the “Closure” property on addition operation**. In other words, we can only optimize the rotation matrix along the manifold of the space instead of the full space.

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

$$\frac{\partial f(R)}{\partial R} = \lim_{\Delta R \rightarrow 0} \frac{f(R + \Delta R) - f(R)}{\Delta R}$$



# Rotation Representations

- Rotation matrix
- Euler Angles
- Axis-angle
- Quaternion
- and many more...

**Rotations**  
A project by  BERKELEY.EDU  
<http://rotations.berkeley.edu>

$$\hat{\omega} \times \vec{r} = \hat{\omega} \times (\vec{r}_\perp + \vec{r}_\parallel) = \hat{\omega} \times \vec{r}_\perp + \hat{\omega} \times \vec{r}_\parallel = \hat{\omega} \times \vec{r}_\perp$$

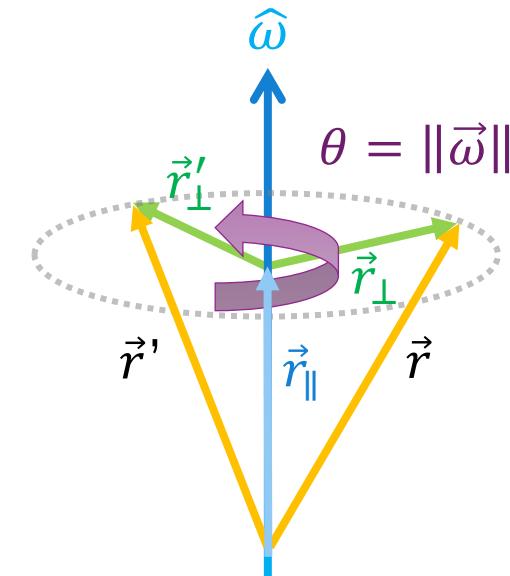
## Axis-Angle

- Specify rotation axis  $\hat{\omega}$ , and rotation angle  $\|\vec{\omega}\|$

$$\begin{aligned}\vec{r}'_\perp &= \cos \theta \vec{r}_\perp + \sin \theta (\hat{\omega} \times \vec{r}_\perp) \\ &= \cos \theta \vec{r}_\perp + \sin \theta (\hat{\omega} \times \vec{r})\end{aligned}$$

$$\vec{r}_\perp = \vec{r} - \vec{r}_\parallel = \vec{r} - (\vec{r} \cdot \hat{\omega})\hat{\omega}$$

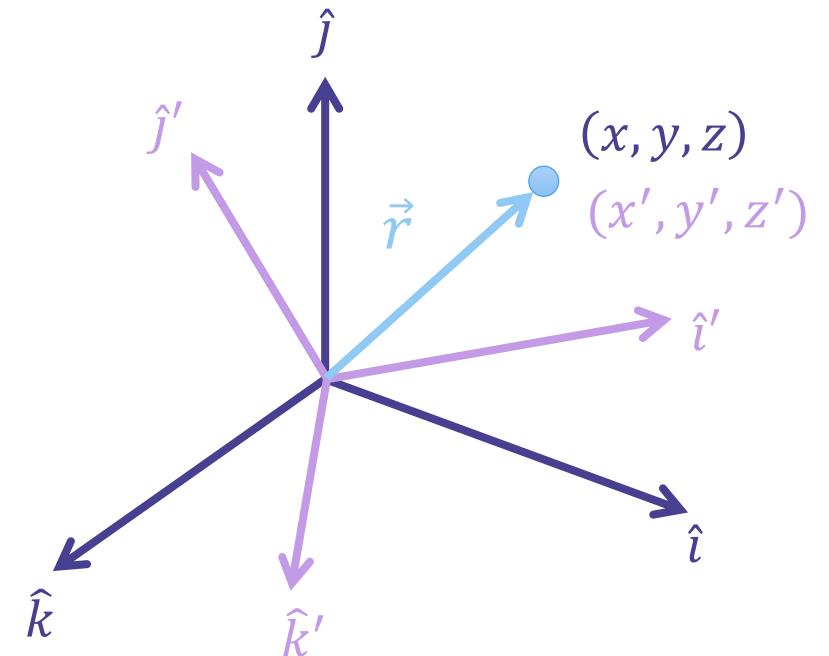
$$\begin{aligned}\vec{r}' &= \vec{r}'_\perp + \vec{r}_\parallel \\ &= \cos \theta (\vec{r} - (\vec{r} \cdot \hat{\omega})\hat{\omega}) + \sin \theta (\hat{\omega} \times \vec{r}) + (\vec{r} \cdot \hat{\omega})\hat{\omega} \\ &= \cos \theta \vec{r} + \sin \theta (\hat{\omega} \times \vec{r}) + (1 - \cos \theta)((\vec{r} \cdot \hat{\omega})\hat{\omega})\end{aligned}$$



(Rodrigues Rotation)

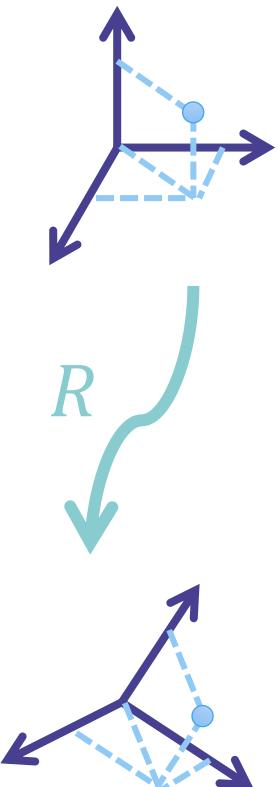
# Change Axes in Cartesian Coordinate

- **Geometric information = coordinates + unit basis**
  - Coordinates are meaningless without unit basis
- $\vec{r}$ : displacement vector
- $\vec{r} = x\hat{i} + y\hat{j} + z\hat{k} = x'\hat{i}' + y'\hat{j}' + z'\hat{k}'$
- Two types of transformations
  - Coordinate-system transformations
    - Transform basis vector
    - Vector is **the same**, but components change
  - Transform vector in *the same* coordinate
    - Vector is **different** from original one



$\vec{r}$  is fixed!  
But its components change!!

# Rotation Matrix



$$\mathbf{a} = [\mathbf{e}_1 \quad \mathbf{e}_2 \quad \mathbf{e}_3] \begin{bmatrix} a_1 \\ a_2 \\ a_3 \end{bmatrix} = a_1 \mathbf{e}_1 + a_2 \mathbf{e}_2 + a_3 \mathbf{e}_3$$

orthogonal matrix

$$[\mathbf{e}_1 \quad \mathbf{e}_2 \quad \mathbf{e}_3] \begin{bmatrix} a_1 \\ a_2 \\ a_3 \end{bmatrix} = [\mathbf{e}_1' \quad \mathbf{e}_2' \quad \mathbf{e}_3'] \begin{bmatrix} a_1' \\ a_2' \\ a_3' \end{bmatrix}$$

$$\mathbf{a} = \begin{bmatrix} a_1 \\ a_2 \\ a_3 \end{bmatrix} = \boxed{\begin{bmatrix} \mathbf{e}_1^T \mathbf{e}_1' & \mathbf{e}_1^T \mathbf{e}_2' & \mathbf{e}_1^T \mathbf{e}_3' \\ \mathbf{e}_2^T \mathbf{e}_1' & \mathbf{e}_2^T \mathbf{e}_2' & \mathbf{e}_2^T \mathbf{e}_3' \\ \mathbf{e}_3^T \mathbf{e}_1' & \mathbf{e}_3^T \mathbf{e}_2' & \mathbf{e}_3^T \mathbf{e}_3' \end{bmatrix}} \begin{bmatrix} a_1' \\ a_2' \\ a_3' \end{bmatrix} = \mathbf{R}\mathbf{a}'$$

$\mathbf{R}$ : 3D rotations centered at the origin

$$\mathbf{a}' = \mathbf{R}^{-1}\mathbf{a} = \mathbf{R}^T\mathbf{a}$$

**SO: Special Orthogonal Group**  $SO(3) = \{\mathbf{R} \in \mathbb{R}^{3 \times 3} \mid \mathbf{R}\mathbf{R}^T = \mathbf{I}, \det(\mathbf{R}) = 1\}$

# Transformation Matrix

$$\mathbf{a}' = \mathbf{R}\mathbf{a} + \mathbf{t}$$

$$\mathbf{a}_1 = \mathbf{R}_{12}\mathbf{a}_2 + \mathbf{t}_{12}$$

$$\mathbf{b} = \mathbf{R}_1\mathbf{a} + \mathbf{t}_1$$

$$\mathbf{c} = \mathbf{R}_2\mathbf{b} + \mathbf{t}_2$$

$$\mathbf{c} = \mathbf{R}_2(\mathbf{R}_1\mathbf{a} + \mathbf{t}_1) + \mathbf{t}_2$$

*Too tedious!*

$$\begin{bmatrix} \mathbf{a}' \\ 1 \end{bmatrix} = \begin{bmatrix} \mathbf{R} & \mathbf{t} \\ \mathbf{0}^T & 1 \end{bmatrix} \begin{bmatrix} \mathbf{a} \\ 1 \end{bmatrix} \stackrel{\text{def}}{=} \mathbf{T} \begin{bmatrix} \mathbf{a} \\ 1 \end{bmatrix}$$

*homogeneous coordinate*

$\mathbf{T}$ : 3D rotations with translations  
(preserve distance and orientation)

$$\mathbf{T}^{-1} = \begin{bmatrix} \mathbf{R}^T & -\mathbf{R}^T \mathbf{t} \\ \mathbf{0}^T & 1 \end{bmatrix}$$

**SE: Special Euclidean group**  $SE(3) = \left\{ \mathbf{T} = \begin{bmatrix} \mathbf{R} & \mathbf{t} \\ \mathbf{0}^T & 1 \end{bmatrix} \in \mathbb{R}^{4 \times 4} \mid \mathbf{R} \in SO(3), \mathbf{t} \in \mathbb{R}^3 \right\}$

# Group

- A family of transformations forms a **group**
- A set **A** together with a binary operation  $\circ$  defined on its elements is called a group **G**, if it satisfies the axioms of **closure, identity, inverse and associativity**

## Closure

$$\forall a_1, a_2 \in A, \quad a_1 \circ a_2 \in A$$

## Inverse

$$\forall a \exists a^{-1} \in A: a \circ a^{-1} = a^{-1} \circ a = e$$

## Identity

$$\exists e \in A: a \circ e = e \circ a = a$$

## Associativity

$$\forall a_1, a_2, a_3 \in A, \quad a_1 \circ (a_2 \circ a_3) = (a_1 \circ a_2) \circ a_3$$

# Interpolating Rotation Matrices

*90° CW around z-axis*

$$0.5 \begin{bmatrix} 0 & 1 & 0 \\ -1 & 0 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

*90° CCW around z-axis*

$$+ 0.5 \begin{bmatrix} 0 & -1 & 0 \\ 1 & 0 & 0 \\ 0 & 0 & 1 \end{bmatrix} = \begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$



***Oops!! This is NOT a rotation matrix!!***

Rotation matrix is a group with multiplication **NOT** addition

$$R_1 + R_2 \notin SO(3)$$

$$R_1 R_2 \in SO(3)$$

# Lie Group

- **Lie group** is a group with the **smoothness** property.
  - $\text{SO}(n)$  and  $\text{SE}(n)$  are both Lie groups
- To solve the optimization problem on manifold space, we need to figure out the concept of the “**Lie Group**” and the mapping to “**Lie Algebra**”.

# Property of Lie Group SO(3)

- An arbitrary rotation matrix satisfy the orthogonal property:  $\mathbf{R}\mathbf{R}^T = \mathbf{I}$
- Assume  $\mathbf{R}$  is the rotation of camera that changes along time, we can represent the rotation as a function of time:  $\mathbf{R}(t)$
- Compute the derivative of the equation:  $\mathbf{R}(t)\mathbf{R}(t)^T = \mathbf{I}$

$$\begin{aligned}\dot{\mathbf{R}}(t)\mathbf{R}(t)^T + \mathbf{R}(t)\dot{\mathbf{R}}(t)^T &= 0 \\ \dot{\mathbf{R}}(t)\mathbf{R}(t)^T &= -(\dot{\mathbf{R}}(t)\mathbf{R}(t)^T)^T\end{aligned}$$

- We can observe that  $\dot{\mathbf{R}}(t)\mathbf{R}(t)^T$  is an antisymmetric matrix constructed according to an unique vector. Define the operators “ $\wedge$ ” and “ $\vee$ ” are the mapping between the antisymmetric matrix and the vector:

$$\mathbf{a}^\wedge = \mathbf{A} = \begin{bmatrix} 0 & -a_3 & a_2 \\ a_3 & 0 & -a_1 \\ -a_2 & a_1 & 0 \end{bmatrix}, \quad \mathbf{A}^\vee = \mathbf{a} = \begin{bmatrix} a_1 \\ a_2 \\ a_3 \end{bmatrix}$$

$$\dot{\mathbf{R}}(t)\mathbf{R}(t)^T = -(\dot{\mathbf{R}}(t)\mathbf{R}(t)^T)^T$$

## Property of Lie Group SO(3)

- Thus we can find a 3D vector  $\Phi(t)$  corresponding to  $\dot{\mathbf{R}}(t)\mathbf{R}(t)^T$ :

$$\dot{\mathbf{R}}(t)\mathbf{R}(t)^T = \Phi(t)^\wedge$$

- The derivative of the rotation matrix can be obtained by multiplying  $\mathbf{R}(t)$  on both sides :

$$\dot{\mathbf{R}}(t) = \Phi(t)^\wedge \mathbf{R}(t) = \begin{bmatrix} 0 & -\phi_3 & \phi_2 \\ \phi_3 & 0 & -\phi_1 \\ -\phi_2 & \phi_1 & 0 \end{bmatrix} \mathbf{R}(t)$$

- Assume  $t_0 = 0, \mathbf{R}(0) = \mathbf{I}$ , we can apply **Taylor expansion** near  $t_0$ . The vector  $\Phi$  is on the “tangent space” of the rotation matrix which reflect the derivative property of the rotation:

$$\mathbf{R}(t) \approx \mathbf{R}(t_0) + \dot{\mathbf{R}}(t_0)(t - t_0) = \mathbf{I} + \Phi(t_0)^\wedge(t)$$

# Lie Algebra

- Assume the vector  $\Phi$  is constant in small time interval around  $t_0$ :

$$\Phi(t_0) = \Phi_0 \quad \dot{\mathbf{R}}(t) = \Phi(t)^{\wedge} \mathbf{R}(t) = \Phi_0^{\wedge} \mathbf{R}(t)$$

- Solve the differential equation of  $\mathbf{R}$  with the initial condition  $\mathbf{R}(0) = \mathbf{I}$ , and we can find the rotation matrix can be computed by an exponential mapping of the asymmetric matrix:

$$\mathbf{R}(t) = \exp(\Phi_0^{\wedge} t)$$

- We then define the **Lie Algebra** as the set of  $\Phi$  or  $\Phi^{\wedge}$

$$\Phi = \Phi^{\wedge} = \begin{bmatrix} 0 & -\phi_3 & \phi_2 \\ \phi_3 & 0 & -\phi_1 \\ -\phi_2 & \phi_1 & 0 \end{bmatrix} \in \mathbb{R}^{3 \times 3}$$

$$\mathfrak{so}(3) = \{\Phi \in \mathbb{R}^3, \Phi = \Phi^{\wedge} \in \mathbb{R}^{3 \times 3}\}$$

# Lie Algebra

- Every Lie Group has a corresponding Lie Algebra representing its **tangent space**
- A set  $\mathbb{V}$  together with a field  $\mathbb{F}$  and a binary operation  $[,]$  (Lie Bracket) defined on its elements is called a Lee Algebra  $\mathfrak{g}$ , if it satisfies the axioms of **closure, bilinearity, alternativity, and Jacobi identity**

## Closure

$$\forall \mathbf{X}, \mathbf{Y} \in \mathbb{V}, \quad [\mathbf{X}, \mathbf{Y}] \in \mathbb{V}$$

## Bilinearity

$$\forall \mathbf{X}, \mathbf{Y}, \mathbf{Z} \in \mathbb{V} \text{ and } a, b \in \mathbb{F}$$

$$[a\mathbf{X} + b\mathbf{Y}, \mathbf{Z}] = a[\mathbf{X}, \mathbf{Z}] + b[\mathbf{Y}, \mathbf{Z}],$$

## Alternativity

$$\forall \mathbf{X} \in \mathbb{V}, \quad [\mathbf{X}, \mathbf{X}] = 0$$

$$[\mathbf{Z}, a\mathbf{X} + b\mathbf{Y}] = a[\mathbf{Z}, \mathbf{X}] + b[\mathbf{Z}, \mathbf{Y}]$$

## Jacobi identity

$$\forall \mathbf{X}, \mathbf{Y}, \mathbf{Z} \in \mathbb{V}, \quad [\mathbf{X}, [\mathbf{Y}, \mathbf{Z}]] + [\mathbf{Z}, [\mathbf{X}, \mathbf{Y}]] + [\mathbf{Y}, [\mathbf{Z}, \mathbf{X}]] = 0$$

# Exponential Mapping of Lie Algebra

$$\phi = \theta a$$

$$\|a\| = 1$$

$$\exp(A) = \sum_{n=0}^{\infty} \frac{1}{n!} A^n$$

$$a^\wedge a^\wedge = \begin{bmatrix} -a_2^2 - a_3^2 & a_1 a_2 & a_1 a_3 \\ a_1 a_2 & -a_1^2 - a_3^2 & a_2 a_3 \\ a_1 a_3 & a_2 a_3 & -a_1^2 - a_2^2 \end{bmatrix} = \begin{bmatrix} a_1^2 & a_1 a_2 & a_1 a_3 \\ a_1 a_2 & a_1^2 & a_2 a_3 \\ a_1 a_3 & a_2 a_3 & a_3^2 \end{bmatrix} - \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} = aa^T - I$$

$$a^\wedge b = -b^\wedge a$$

$$\begin{bmatrix} 0 & -a_3 & a_2 \\ a_3 & 0 & -a_1 \\ -a_2 & a_1 & 0 \end{bmatrix} \begin{bmatrix} b_1 \\ b_2 \\ b_3 \end{bmatrix} = \begin{bmatrix} -a_3 b_2 + a_2 b_3 \\ a_3 b_1 - a_1 b_3 \\ -a_2 b_1 + a_1 b_2 \end{bmatrix}$$

$$\begin{bmatrix} 0 & -b_3 & b_2 \\ b_3 & 0 & -b_1 \\ -b_2 & a_1 & 0 \end{bmatrix} \begin{bmatrix} a_1 \\ a_2 \\ a_3 \end{bmatrix} = \begin{bmatrix} -a_2 b_3 + a_3 b_2 \\ a_1 b_3 - a_3 b_1 \\ -a_1 b_2 + a_2 b_1 \end{bmatrix}$$

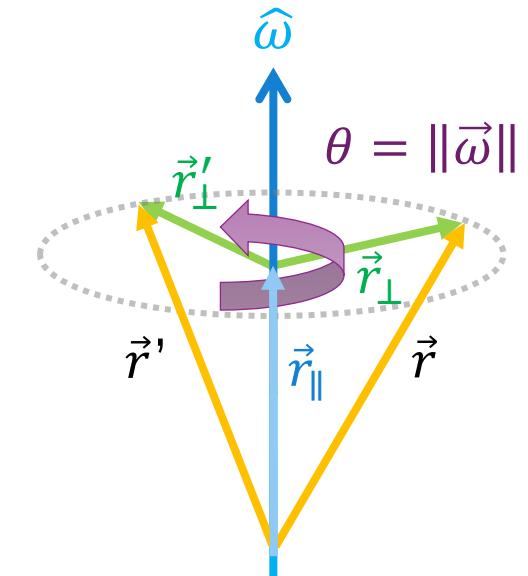
$$a^\wedge a = -a^\wedge a = 0$$

$$a^\wedge a^\wedge a^\wedge = a^\wedge (aa^T - I) = -a^\wedge$$

# Exponential Mapping of Lie Algebra

- The exponential of the matrix:

$$\begin{aligned}\exp(\phi^\wedge) &= \exp(\theta a^\wedge) = \sum_{n=0}^{\infty} \frac{1}{n!} (\theta a^\wedge)^n \\&= I + \theta a^\wedge + \frac{1}{2!} \theta^2 a^\wedge a^\wedge + \frac{1}{3!} \theta^3 a^\wedge a^\wedge a^\wedge + \frac{1}{4!} \theta^4 (a^\wedge)^4 + \dots \\&= aa^T - a^\wedge a^\wedge + \theta a^\wedge + \frac{1}{2!} \theta^2 a^\wedge a^\wedge - \frac{1}{3!} \theta^3 a^\wedge - \frac{1}{4!} \theta^4 (a^\wedge)^2 + \dots \\&= aa^T + \left( \theta - \frac{1}{3!} + \frac{1}{5!} - \dots \right) a^\wedge - \left( 1 - \frac{1}{2!} \theta^2 + \frac{1}{4!} \theta^4 - \dots \right) a^\wedge a^\wedge \\&= aa^T + \sin \theta a^\wedge - \cos \theta (aa^T - I) \\&= \cos \theta I + (1 - \cos \theta)aa^T + \sin \theta a^\wedge \quad (\text{Rodrigues Rotation})\end{aligned}$$



$$\vec{r}' = \cos \theta \vec{r} + \sin \theta (\hat{\omega} \times \vec{r}) + (1 - \cos \theta)((\vec{r} \cdot \hat{\omega})\hat{\omega})$$

# Lie Group vs. Lie Algebra

Lie Group  
 $SO(3)$

$$R \in \mathbb{R}^{3 \times 3}$$

$$RR^T = I$$

$$\det(R) = 1$$

Mapping

Exponential Mapping

$$\exp(\phi^\wedge) = \cos \theta I + (1 - \cos \theta)aa^T + \sin \theta a^\wedge$$

Logarithmic Mapping

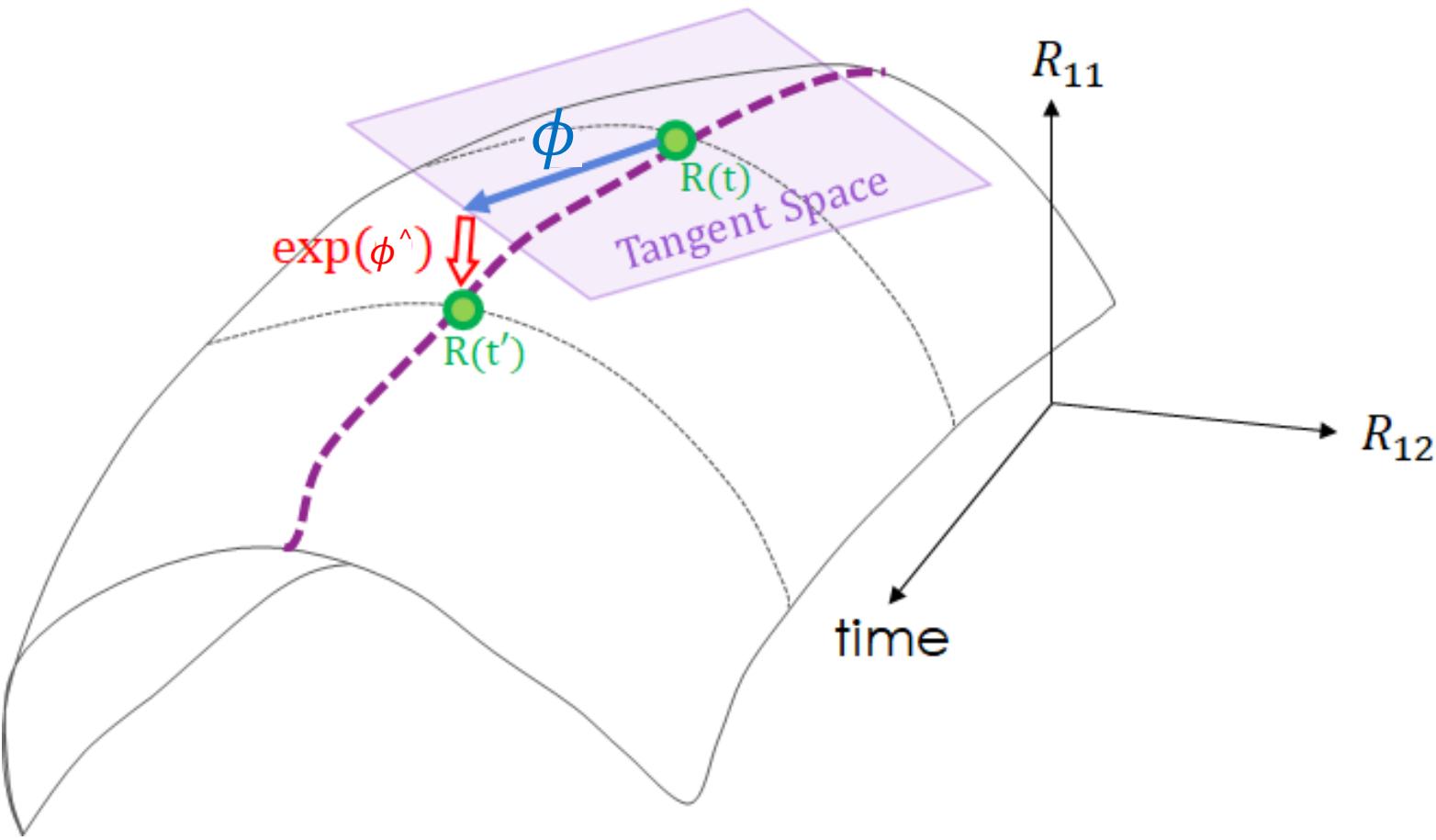
$$\phi = (\ln R)^\vee = \left( \sum_{n=0}^{\infty} \frac{(-1)^n}{n+1} (R - I)^{n+1} \right)^\vee$$
$$\theta = \arccos \frac{\text{tr}(R) - 1}{2} \quad Ra = a$$

Lie Algebra  
 $\mathfrak{so}(3)$

$$\phi \in \mathbb{R}^3$$

$$\phi^\wedge = \begin{bmatrix} 0 & -\phi_3 & \phi_2 \\ \phi_3 & 0 & -\phi_1 \\ -\phi_2 & \phi_1 & 0 \end{bmatrix}$$

# 3D Rotation in Lie Group/Algebra



# BCH Equation and Approximation

- Can we apply addition operation on the Lie Algebra ?

$$\exp(\phi_1^\wedge) \exp(\phi_2^\wedge) = \exp((\phi_1 + \phi_2)^\wedge) ?? \quad \ln(\exp(A) \exp(B)) = A + B ??$$

- Baker-Campbell-Hausdorff (BCH) Equation

$$\ln(\exp(A) \exp(B)) = A + \sum_{n=0}^{\infty} (-1)^n \left( \frac{B_n}{n!} \right) [A, [A, \dots [A, B] \dots]] = A + B + \frac{1}{2}[A, B] + \frac{1}{12}[A, [A, B]] + \dots$$

- Linear Approximation

$$\ln(\exp(\phi_1^\wedge) \exp(\phi_2^\wedge)) \approx \begin{cases} J_l(\phi_2)^{-1}\phi_1 + \phi_2 & \text{Left Multiply Approximation} \\ J_r(\phi_1)^{-1}\phi_2 + \phi_1 & \text{Right Multiply Approximation} \end{cases}$$

$$J_l = \frac{\sin \theta}{\theta} I + \left( 1 - \frac{\sin \theta}{\theta} \right) aa^T + \frac{1 - \cos \theta}{\theta} a^\wedge$$

# Derivative of Lie Algebra

$$\cancel{\frac{\partial(Rp)}{\partial R}}$$

$$\begin{aligned}
 & \rightarrow \frac{\partial(\exp(\phi^\wedge)p)}{\partial\phi} = \lim_{\delta\phi\rightarrow 0} \frac{\exp((\phi + \delta\phi)^\wedge)p - \exp(\phi^\wedge)p}{\delta\phi} \\
 &= \lim_{\delta\phi\rightarrow 0} \frac{\exp(J_l\delta\phi)^\wedge \exp(\phi^\wedge)p - \exp(\phi^\wedge)p}{\delta\phi} \\
 &= \lim_{\delta\phi\rightarrow 0} \frac{(I + (J_l\delta\phi)^\wedge) \exp(\phi^\wedge)p - \exp(\phi^\wedge)p}{\delta\phi} \\
 &= \lim_{\delta\phi\rightarrow 0} \frac{(J_l\delta\phi)^\wedge \exp(\phi^\wedge)p}{\delta\phi} \\
 &= \lim_{\delta\phi\rightarrow 0} \frac{-(\exp(\phi^\wedge)p)^\wedge J_l \delta\phi}{\delta\phi} \\
 &= -(Rp)^\wedge J_l
 \end{aligned}$$

$$\phi + \delta\phi = \delta\phi + \phi \rightarrow J_l(\phi_2)^{-1}\phi_1 + \phi_2$$

$$\ln(\exp(\phi_1^\wedge)\exp(\phi_2^\wedge)) \approx \begin{cases} J_l(\phi_2)^{-1}\phi_1 + \phi_2 \\ J_r(\phi_1)^{-1}\phi_2 + \phi_1 \end{cases}$$

1st order Taylor Expansion

$$a^\wedge b = -b^\wedge a$$

$$R_{init} = \exp(\phi_{init}^\wedge)$$

$$\phi_{new} = \phi_{init} - \alpha \left( \frac{\partial u}{\partial \phi} \right)^T = \phi_{init} + \alpha J_l^T (Rp)^\wedge \left. \frac{\partial u}{\partial (Rp)} \right|_{R=R_{init}}^T$$

$$\frac{\partial u}{\partial \phi} = \frac{\partial u}{\partial (Rp)} \frac{\partial (Rp)}{\partial \phi} = -\frac{\partial u}{\partial (Rp)} (Rp)^\wedge J_l$$

# Perturbation Model

Derivative of Lie Algebra:

$$\frac{\partial(\exp(\phi^\wedge)p)}{\partial\phi} = \lim_{\delta\phi\rightarrow 0} \frac{\exp((\phi + \delta\phi)^\wedge)p - \exp(\phi^\wedge)p}{\delta\phi}$$

Left Perturbation:

$$\begin{aligned}\frac{\partial(Rp)}{\partial\psi} &= \lim_{\psi\rightarrow 0} \frac{\exp(\psi^\wedge)\exp(\phi^\wedge)p - \exp(\phi^\wedge)p}{\psi} \\ &= \lim_{\psi\rightarrow 0} \frac{(I + \psi^\wedge)\exp(\phi^\wedge)p - \exp(\phi^\wedge)p}{\psi} \\ &= \lim_{\psi\rightarrow 0} \frac{\psi^\wedge Rp}{\psi} = \lim_{\psi\rightarrow 0} \frac{-(Rp)^\wedge\psi}{\psi} = -(Rp)^\wedge\end{aligned}$$

1st order Taylor Expansion

$$a^\wedge b = -b^\wedge a$$

$$R_{new} = \exp(\psi^\wedge) R_{init}$$

# Q&A

# Feature Point Extraction

# Popular Feature Extractors

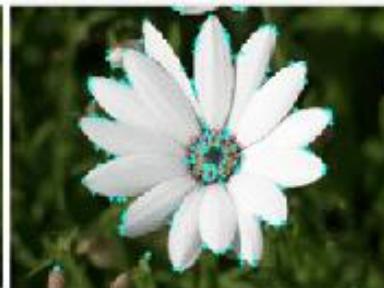
SURF



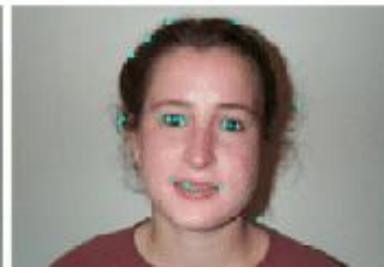
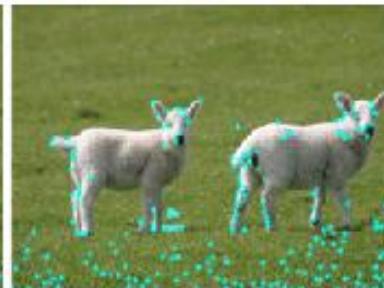
SIFT



Harris



CEFF

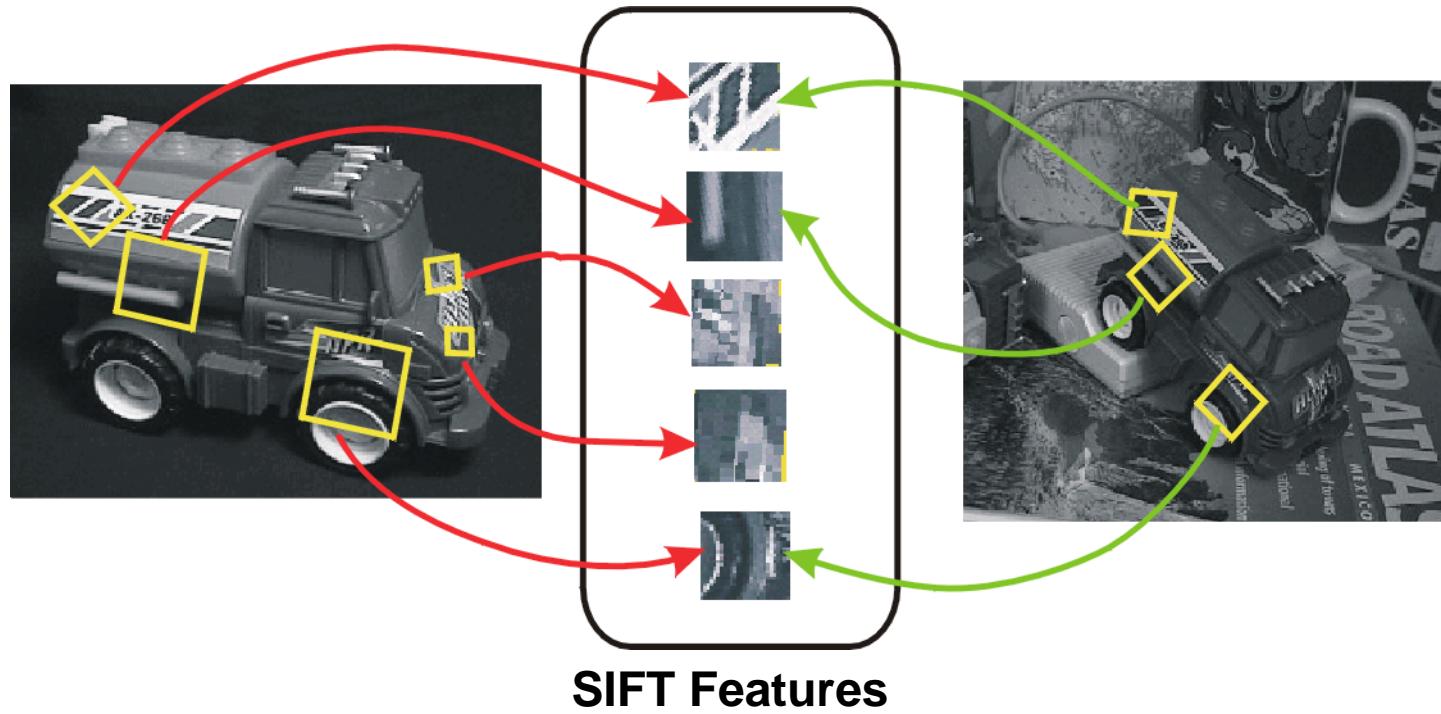


[Ref] Nawaz, Mehmood, et al. Clustering based one-to-one hypergraph matching with a large number of feature points. *Signal Processing: Image Communication*, 2019, 74: 289-298.



## Idea of SIFT

- Image content is transformed into local feature coordinates that are invariant to translation, rotation, scale, and other imaging parameters

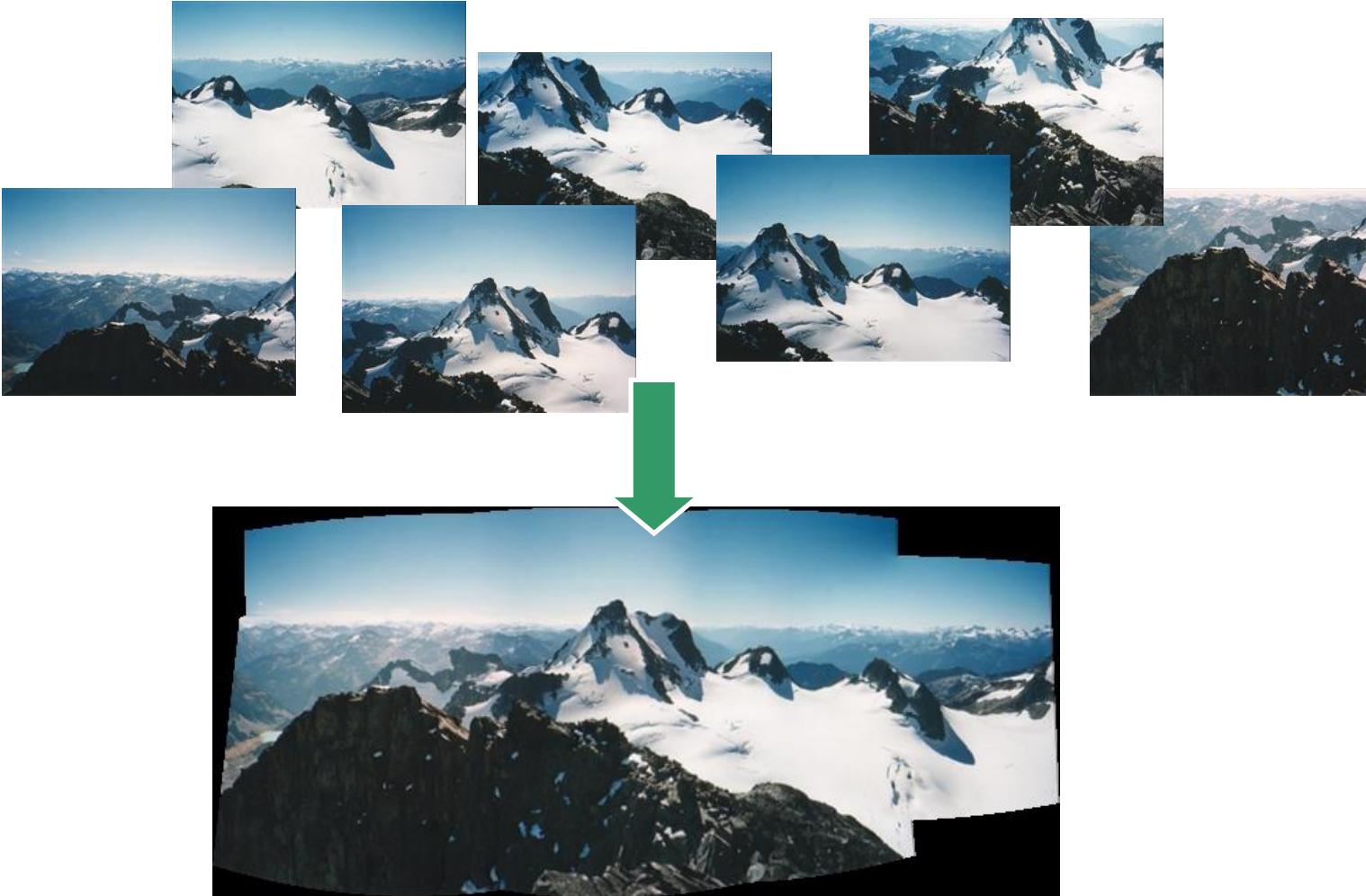


[Ref] Lowe, David G. Distinctive image features from scale-invariant keypoints.  
*International journal of computer vision*, 2004, 60.2: 91-110.

# Application: Object Recognition (Matching)



# Application: Image Stitching



# Application: Photosynth



(a)

## Photo Tourism

Exploring photo collections in 3D

**Microsoft**



(b)



(c)

# Claimed Advantages of SIFT

- **Locality**
  - features are local, so robust to occlusion and clutter (no prior segmentation)
- **Distinctiveness**
  - individual features can be matched to a large database of objects
- **Quantity**
  - many features can be generated for even small objects
- **Efficiency**
  - close to real-time performance
- **Extensibility**
  - can easily be extended to wide range of other feature types, with each adding robustness

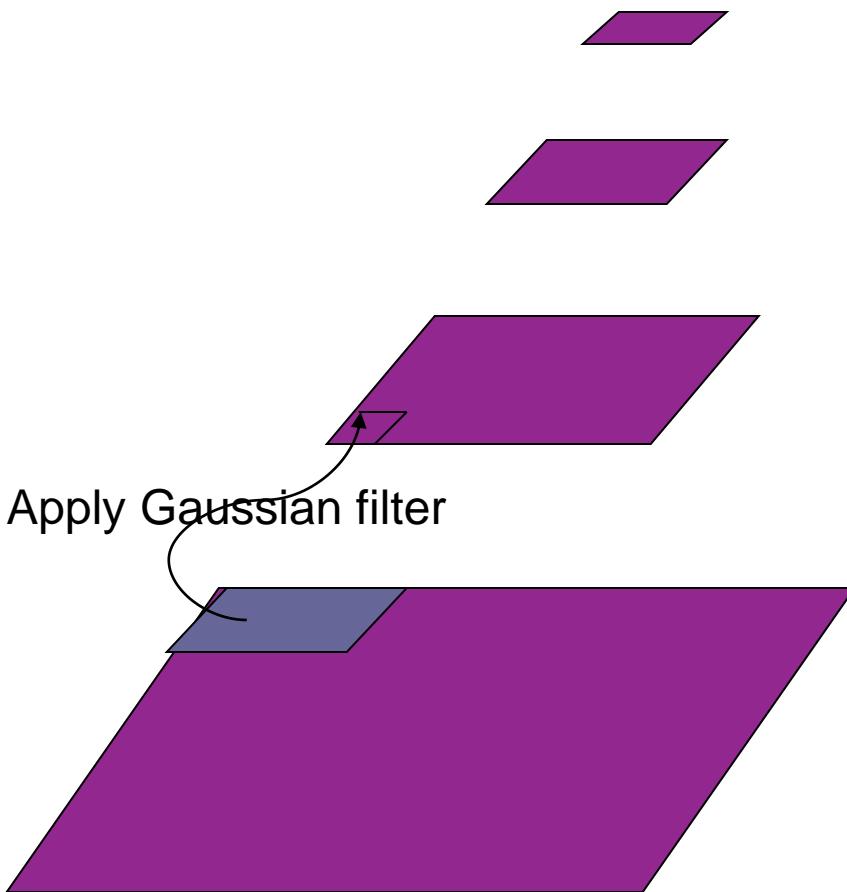
## 4 Steps of SIFT

- **Scale-space extrema detection**
  - Search over multiple scales and image locations
- **Keypoint localization**
  - Fit a model to determine location and scale
  - Select keypoints based on a measure of stability
- **Orientation assignment**
  - Compute best orientation(s) for each keypoint region
- **Keypoint descriptor**
  - Use local image gradients at selected scale and rotation to describe each keypoint region

# 1. Scale-space Extrema Detection

- Goal:
  - Identify locations and scales that can be repeatably assigned under different views of the same scene or object.
- Method:
  - Search for stable features across multiple scales using a **continuous function of scale**.
- Prior work has shown that under a variety of assumptions, the best function is a **Gaussian function**.
- The scale space of an image is a function  $L(x,y,\sigma)$  that is produced from the convolution of a Gaussian kernel (at different scales) with the input image.

# Gaussian Pyramid

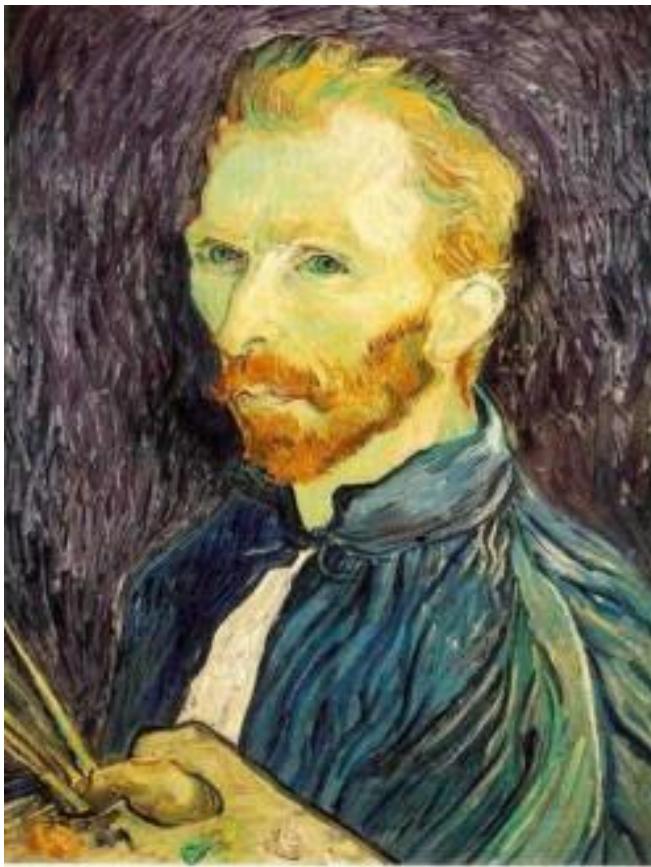


And so on.

At 2<sup>nd</sup> level, each pixel is the result of applying a Gaussian mask to the first level and then subsampling to reduce the size.

Bottom level is the original image.

# Example



Gaussian 1/2



G 1/4



G 1/8

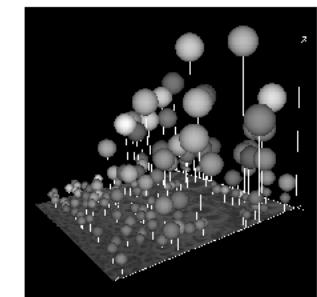
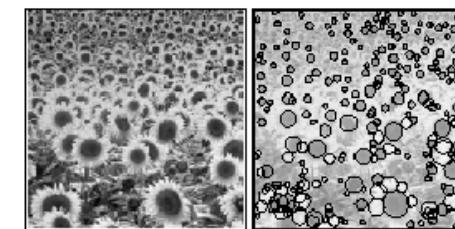
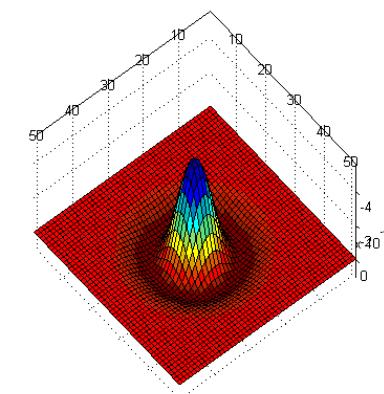
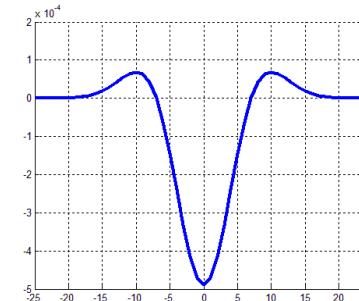
# Lowe's Scale-space Interest Points

- **Laplacian of Gaussian kernel**
  - Scale normalized
  - Proposed by Lindeberg
- Scale-space detection
  - Find local maxima across scale/space
  - A good “blob” detector

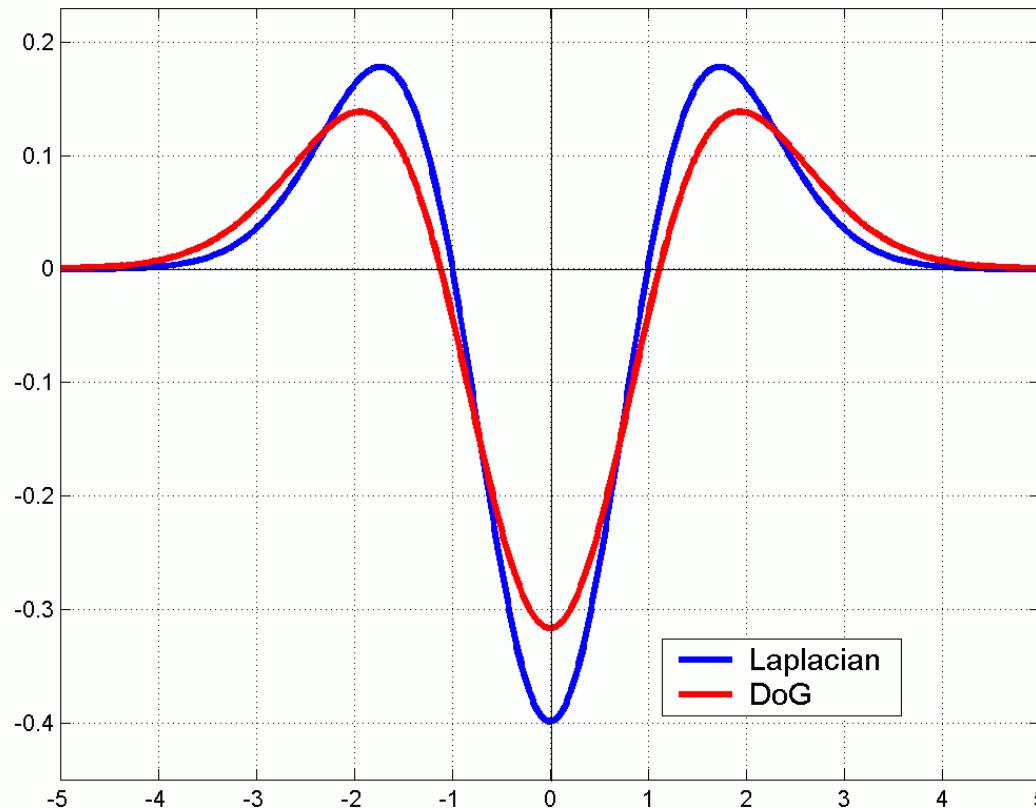
$$G(x, y, \sigma) = \frac{1}{\sqrt{2\pi}\sigma} e^{-\frac{1}{2} \frac{x^2+y^2}{\sigma^2}}$$

$$\Delta[G_\sigma(x, y) * f(x, y)] = [\Delta G_\sigma(x, y)] * f(x, y)$$

$$LoG = \Delta G_\sigma(x, y) = \frac{\partial^2 G_\sigma(x, y)}{\partial x^2} + \frac{\partial^2 G_\sigma(x, y)}{\partial y^2} = \frac{x^2 + y^2 - 2\sigma^2}{\sigma^4} e^{-(x^2+y^2)/2\sigma^2}$$



# Lowe's Scale-space Interest Points: Difference of Gaussians



$$G_\sigma(x, y) = \frac{1}{2\pi\sigma^2} \exp\left(-\frac{x^2 + y^2}{2\sigma^2}\right)$$

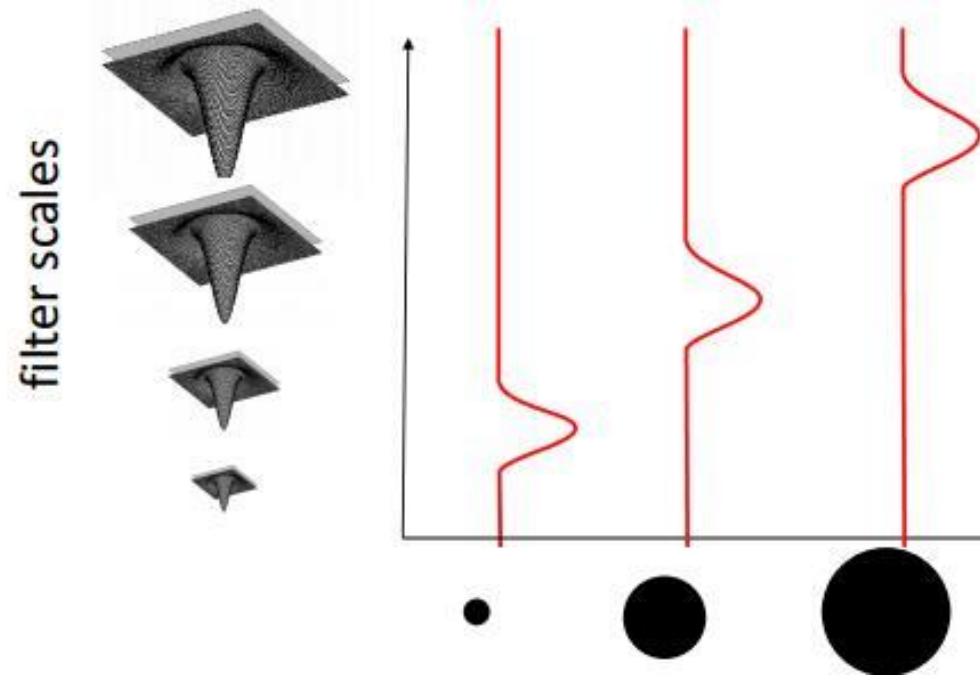
$$DoG = G_{\sigma_1} - G_{\sigma_2} = \frac{1}{\sqrt{2\pi}} \left[ \frac{1}{\sigma_1} e^{-(x^2+y^2)/2\sigma_1^2} - \frac{1}{\sigma_2} e^{-(x^2+y^2)/2\sigma_2^2} \right]$$

$$\frac{\partial G}{\partial \sigma} = \sigma \nabla^2 G$$

$$\frac{\partial G}{\partial \sigma} \approx \frac{G(x, y, k\sigma) - G(x, y, \sigma)}{k\sigma - \sigma}$$

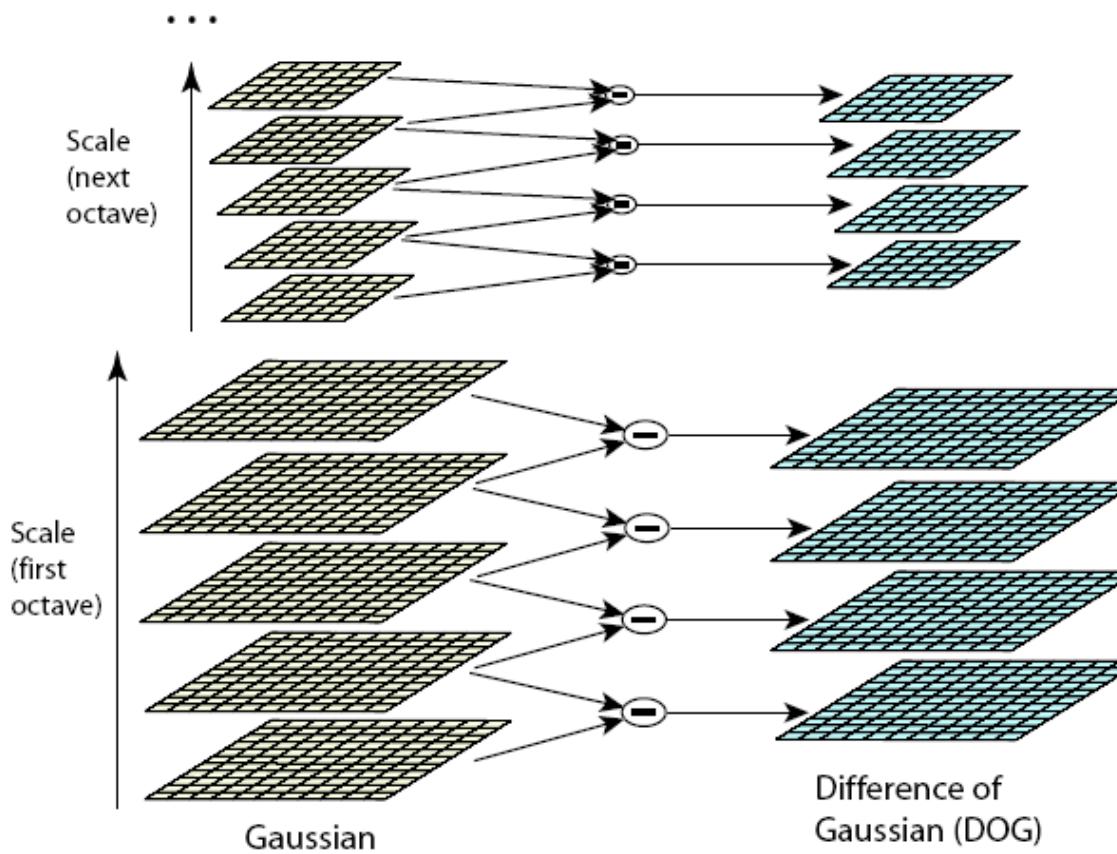
$$G(x, y, k\sigma) - G(x, y, \sigma) \approx (k - 1)\sigma^2 \nabla^2 G$$

# Lowe's Scale-space Interest Points: Difference of Gaussians



# Lowe's Pyramid Scheme

$$\begin{aligned}\sigma_{s+1} &= 2^{(s+1)/s} \sigma_0 \\ \vdots \\ \sigma_i &= 2^{i/s} \sigma_0 \\ \vdots \\ \sigma_2 &= 2^{2/s} \sigma_0 \\ \sigma_1 &= 2^{1/s} \sigma_0 \\ \sigma_0\end{aligned}$$

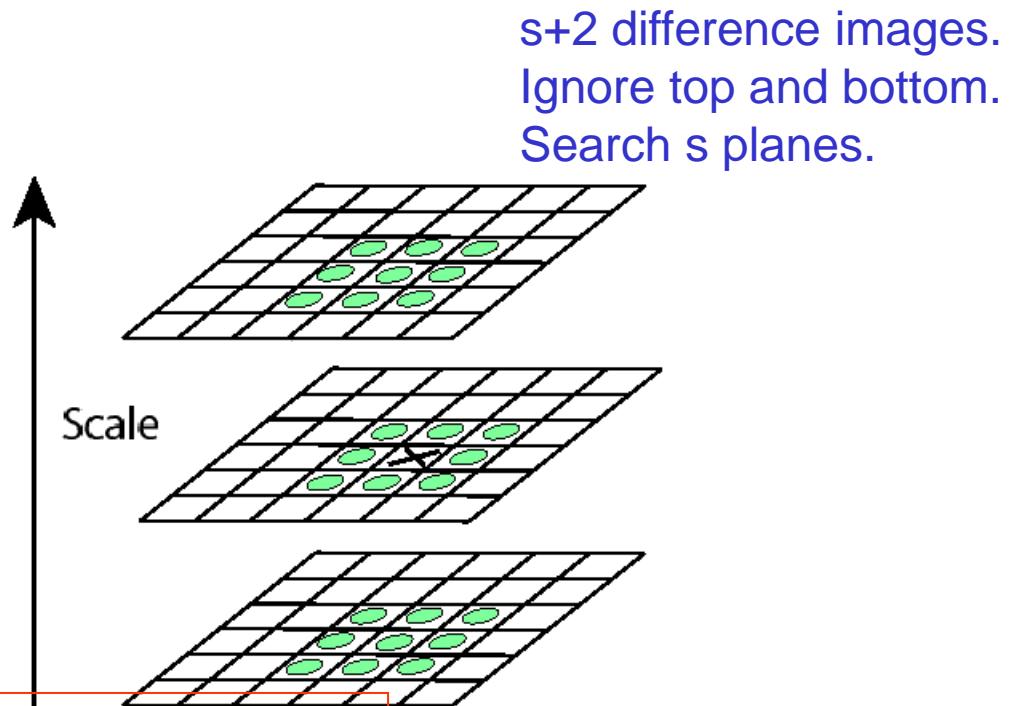


The parameter  $s$  determines the number of images per octave

## 2. Key point localization

- Detect maxima and minima of difference-of-Gaussian in scale space
- Each point is compared to its 8 neighbors in the current image and 9 neighbors each in the scales above and below

For each max or min found,  
output is the **location** and  
the **scale**.



## 2. Keypoint Localization

- There are still a lot of points, some of them are not good enough
  - The locations of keypoints may be not accurate

Taylor series expansion

$$D(\mathbf{x}) = D + \frac{\partial D^T}{\partial \mathbf{x}} \mathbf{x} + \frac{1}{2} \mathbf{x}^T \frac{\partial^2 D}{\partial \mathbf{x}^2} \mathbf{x}$$

$$\hat{\mathbf{x}} = -\frac{\partial^2 D}{\partial \mathbf{x}^2}^{-1} \frac{\partial D}{\partial \mathbf{x}}$$

$$D(\hat{\mathbf{x}}) = D + \frac{1}{2} \frac{\partial D^T}{\partial \mathbf{x}} \hat{\mathbf{x}}$$

# Eliminating the Edge Response

- Reject flats by a gradient threshold:

- $|D(\hat{x})| < 0.03$

- Reject edges by a ratio threshold:

$$\mathbf{H} = \begin{bmatrix} D_{xx} & D_{xy} \\ D_{xy} & D_{yy} \end{bmatrix}$$

$$\text{Tr}(\mathbf{H}) = D_{xx} + D_{yy} = \alpha + \beta,$$

$$\text{Det}(\mathbf{H}) = D_{xx}D_{yy} - (D_{xy})^2 = \alpha\beta.$$

Let  $\alpha$  be the eigenvalue with larger magnitude and  $\beta$  the smaller.

$$\frac{\text{Tr}(\mathbf{H})^2}{\text{Det}(\mathbf{H})} = \frac{(\alpha + \beta)^2}{\alpha\beta} = \frac{(r\beta + \beta)^2}{r\beta^2} = \frac{(r+1)^2}{r},$$

Let  $r = \alpha/\beta$ .  
So  $\alpha = r\beta$

- $r < 10$

$(r+1)^2/r$  is at a min when the 2 eigenvalues are equal.

# Eliminating the Edge Response

233x189  
input image



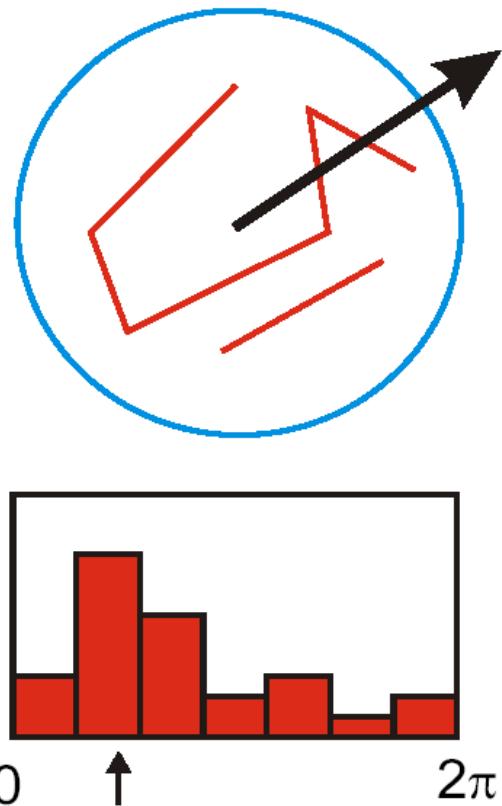
832  
initial keypoints

729  
keypoints after  
gradient threshold



536  
keypoints after  
ratio threshold

### 3. Orientation assignment



- Create histogram of local gradient directions at selected scale
- Assign canonical orientation at peak of smoothed histogram
- Each key specifies stable 2D coordinates  
( $x, y, \text{scale}, \text{orientation}$ )

If 2 major orientations, use both.

## Orientation Assignment

- Assign an orientation to each keypoint, the keypoint descriptor can be represented relative to this orientation and therefore **achieve invariance to image rotation**
- Compute **magnitude** and **orientation** on the Gaussian smoothed images

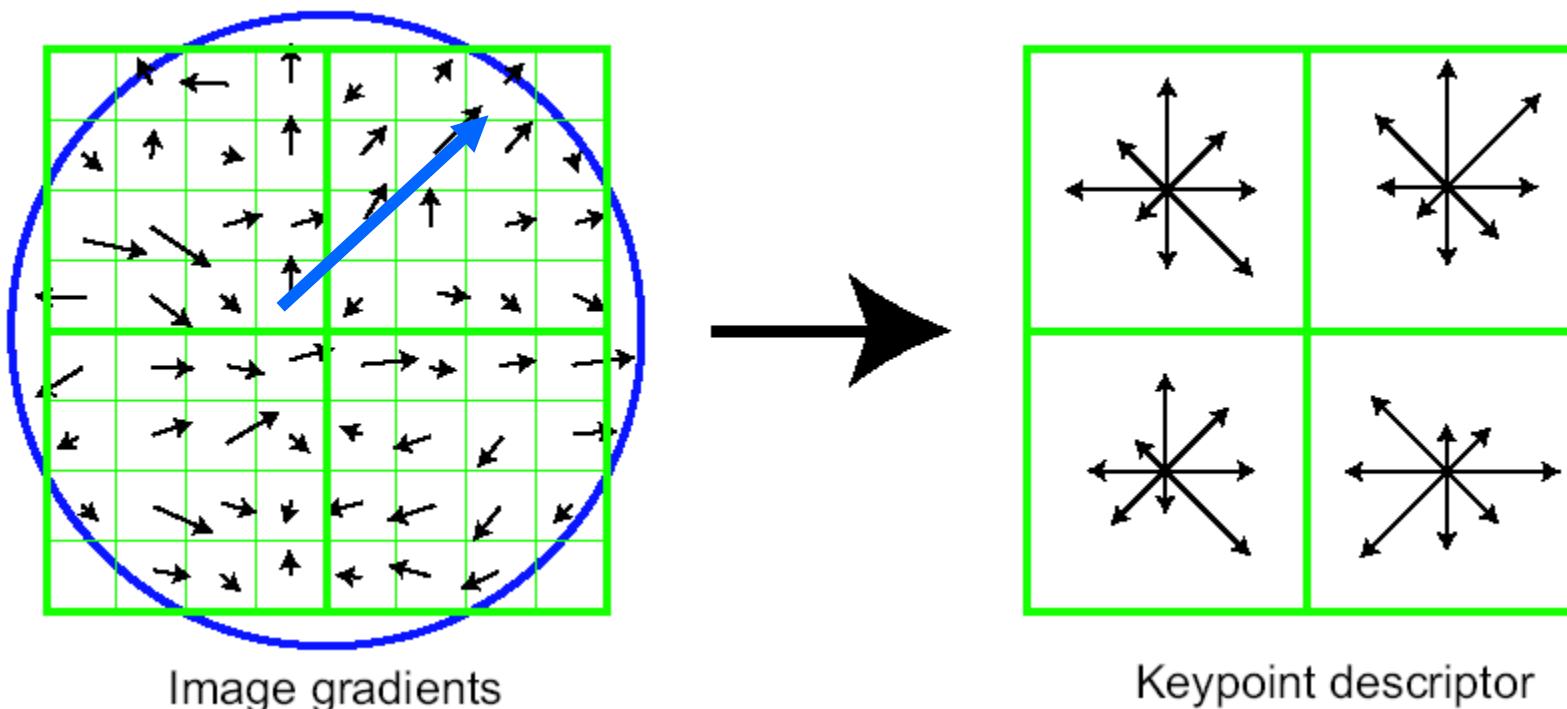
$$m(x, y) = \sqrt{(L(x+1, y) - L(x-1, y))^2 + (L(x, y+1) - L(x, y-1))^2}$$

$$\theta(x, y) = \tan^{-1}((L(x, y+1) - L(x, y-1)) / (L(x+1, y) - L(x-1, y)))$$

## 4. Keypoint Descriptors

- At this point, each keypoint has
  - location
  - scale
  - orientation
- Next is to compute a descriptor for the local image region about each keypoint that is
  - highly distinctive
  - invariant as possible to variations such as changes in viewpoint and illumination

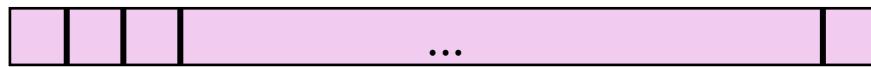
# Lowe's Keypoint Descriptor (shown with 2 X 2 descriptors over 8 X 8)



In experiments, 4x4 arrays of 8 bin histogram is used,  
a total of 128 features for one keypoint

# Lowe's Keypoint Descriptor

- Use the **normalized region** about the keypoint
- Compute gradient magnitude and orientation at each point in the region
- Weight them by a Gaussian window overlaid on the circle
- Create an **orientation histogram** over the  $4 \times 4$  subregions of the window
- $4 \times 4$  descriptors over  $16 \times 16$  sample array were used in practice.  $4 \times 4$  times 8 directions gives a **vector of 128 values**.



# Application on Object Recognition

- The SIFT features of training images are extracted and stored
- For a query image
  1. Extract SIFT feature
  2. Efficient nearest neighbor indexing
  3. 3 keypoints, Geometry verification (RANSAC)



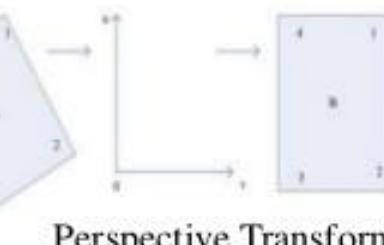
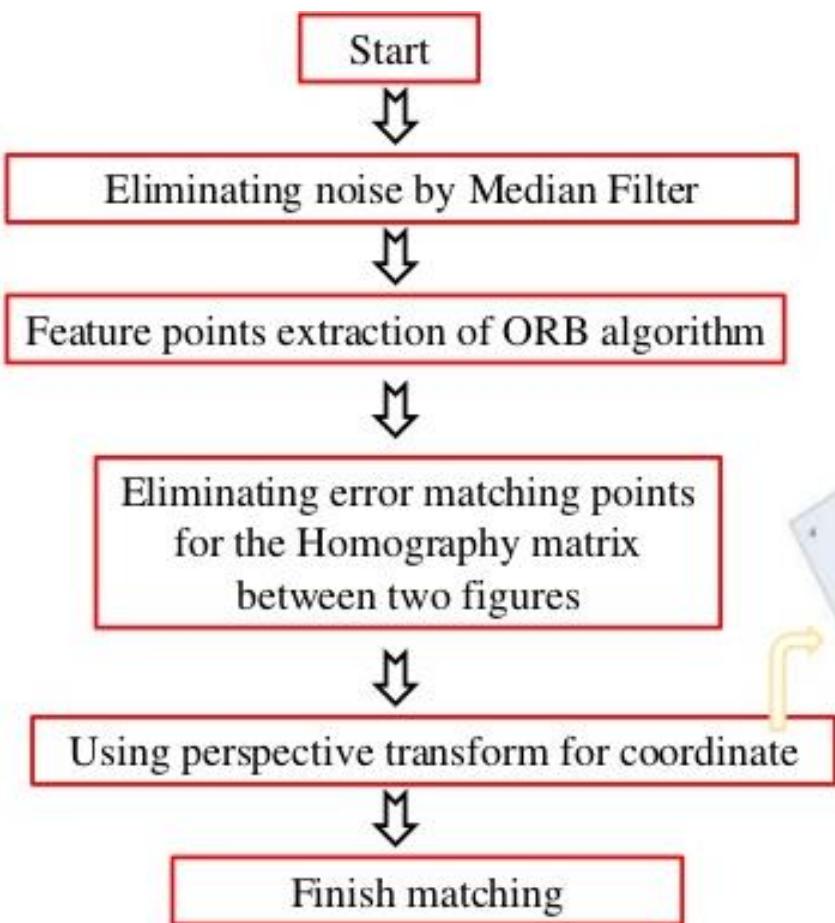
## Extensions

- PCA-SIFT
  1. Work on patches with size 41\*41 pixels
  2. Compute vertical and horizontal gradient for all pixels (2\*39\*39 dimensions)
  3. Use PCA to project it to 20 dimensions

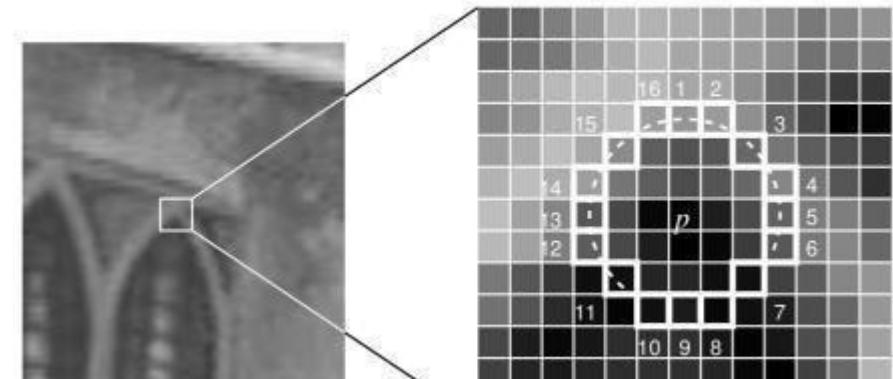
## SURF

- Approximate SIFT
- Works almost equally well
- Very fast

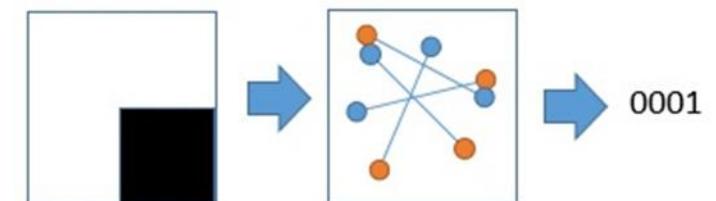
# ORB Feature Matching



Perspective Transform



FAST Feature Detection



$$m_{pq} = \sum_{x,y} x^p y^q I(x,y)$$

$$C = \left( \frac{m_{10}}{m_{00}}, \frac{m_{01}}{m_{00}} \right)$$

$$\theta = \text{atan2}(m_{01}, m_{10})$$

BRIEF Descriptor

# Bag of Words

