# Visual SLAM

Presentation heavily based on coursera robotics perception course

# Terminology

- **Visual Odometry :** The process of incrementally estimating your position and orientation with respect to an initial reference frame by tracking visual features
- **Visual SLAM :** used interchangeably but Visual SLAM produces also map of features while visual odometry focuses on the camera trajectory

# Warning : no filters here … in some way

- 2 methodologies have become predominant in V-SLAM
  - 1) Filtering methods fuse the information from all the images with a probability distribution
  - 2) Nonfiltering methods (also called keyframe methods) retain the optimization of global bundle adjustment to selected keyframes
- We focus on method 2 for V-SLAM here !

  (a la ORB SLAM which is a state of the art method)
- But in some way, anyways, we will slide over a window, track and control uncertainty and minimize least squares all over the place

# Formulation of the VO problem

- An agent is moving through an environment and taking images with a rigidly attached camera system at discrete time instants k
- 2 cameras positions at adjacent time instants k-1 and k are related by a rigid body transformation (4x4 matrix)
- The set of camera poses C[0:N] contains the transformations of the camera with respect to the initial coordinate frame at k=0

# Formulation of the VO problem

Two camera positions at adjacent time instants $k-1$ and $k$ are related by the rigid body transformation $T_{k,k-1} \in \mathbb{R}^{4\times 4}$ of the following form:

$$T_{k,k-1} = \begin{bmatrix} R_{k,k-1} & t_{k,k-1} \\ 0 & 1 \end{bmatrix}, \qquad (1)$$

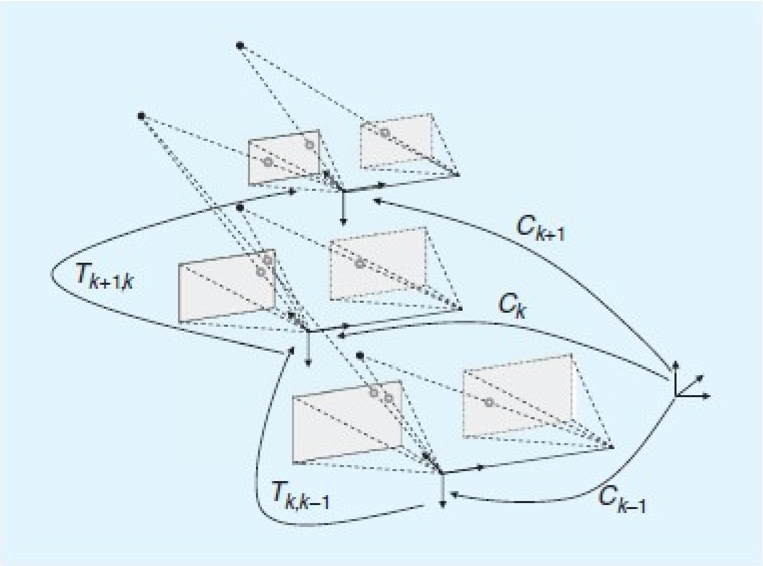


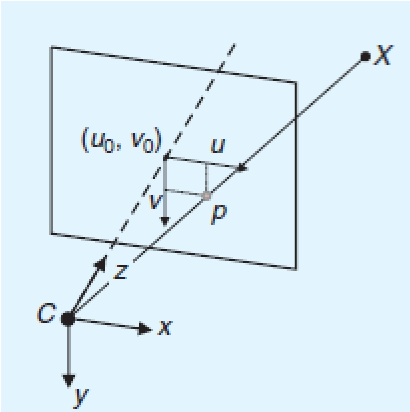

**Figure 1.** An illustration of the visual odometry problem. The relative poses $T_{k,k-1}$ of adjacent camera positions (or positions of a camera system) are computed from visual features and concatenated to get the absolute poses $C_k$ with respect to the initial coordinate frame at $k = 0$.

# Preliminaries : Perspective camera model

***Perspective Camera Model***

The most used model for perspective camera assumes a pin-hole projection system: the image is formed by the intersection of the light rays from the objects through the center of the lens (projection center), with the focal plane [Figure 3(a)]. Let $X = [x, y, z]^\top$ be a scene point in the camera reference frame and $p = [u, v]^\top$ its projection on the image plane measured in pixels. The mapping from the 3-D world to the 2-D image is given by the perspective projection equation:

$$\lambda \begin{bmatrix} u \\ v \\ 1 \end{bmatrix} = KX = \begin{bmatrix} \alpha_u & 0 & u_0 \\ 0 & \alpha_v & v_0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \end{bmatrix}, \qquad (2)$$



(a)

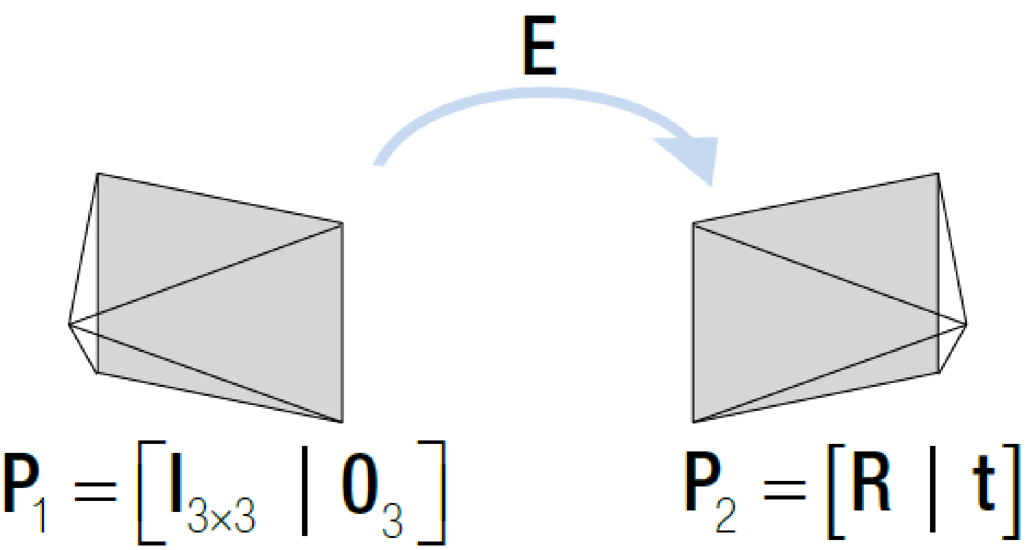
# Preliminaries : 2D transformations and Homogeneous coordinates

- Rotation : orthogonal matrix
- Translation
- Similarity : scaled rotation + translation
- Affine transform : parallel lines remain paralel
- Homography or projective transform: the most generic 2D linear transform (straight lines remain straight but not necessarily parallel)
- Reminder : we will use homogeneous coordinates (so that translation can be handled via matrix multiplications)

# Preliminaries : Rigid body transformation

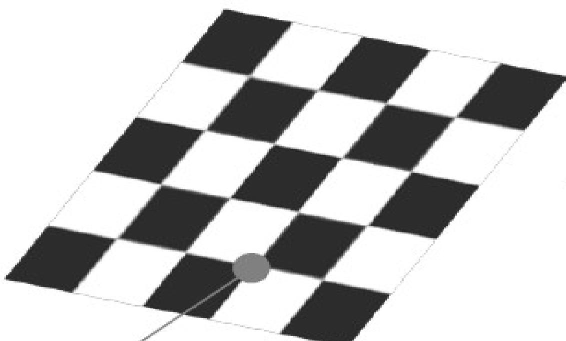
$$\underline{\mathbf{E} = [\mathbf{t}]_\times \mathbf{R}}$$ How to decompose the essential matrix to rotation and translation?

Essential matrix (rank 2)

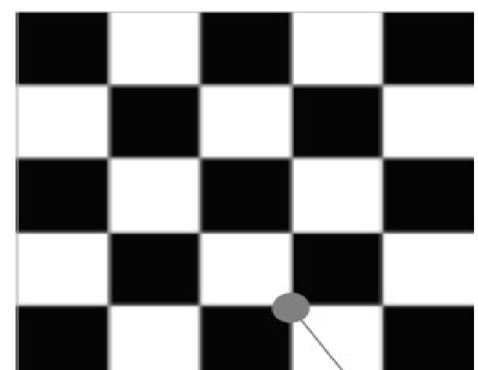


# Preliminaries :
# fundamental example

## Homography Linear Estimation



$$\mathbf{x}_2 = \mathbf{H}\mathbf{x}_1$$



$$\mathbf{x}_2 = \begin{bmatrix} u_2 \\ v_2 \\ 1 \end{bmatrix}$$

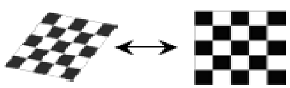
$$\mathbf{H} = \begin{bmatrix} h_{11} & h_{12} & h_{13} \\ h_{21} & h_{22} & h_{23} \\ h_{31} & h_{32} & h_{33} \end{bmatrix}$$

$$\mathbf{x}_1 = \begin{bmatrix} u_1 \\ v_1 \\ 1 \end{bmatrix}$$

How to estimate $\mathbf{H}$ ?

# Preliminaries : fundamental example

## Homography Linear Estimation



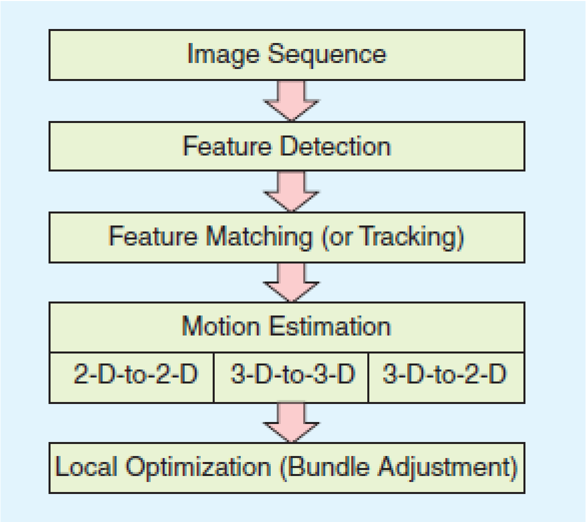
$$\mathbf{x}_2 = \mathbf{H}\mathbf{x}_1 \longrightarrow [\mathbf{x}_2]_\times \mathbf{H}\mathbf{x}_1 = \mathbf{0} \longrightarrow$$

$$\begin{bmatrix} u_2 \\ v_2 \\ 1 \end{bmatrix}_\times \begin{bmatrix} \mathbf{h}_1 \\ \mathbf{h}_2 \\ \mathbf{h}_3 \end{bmatrix} \mathbf{x}_1 = \begin{bmatrix} u_2 \\ v_2 \\ 1 \end{bmatrix}_\times \begin{bmatrix} \mathbf{h}_1\mathbf{x}_1 \\ \mathbf{h}_2\mathbf{x}_1 \\ \mathbf{h}_3\mathbf{x}_1 \end{bmatrix} = \begin{bmatrix} u_2 \\ v_2 \\ 1 \end{bmatrix}_\times = \begin{bmatrix} u_2 \\ v_2 \\ 1 \end{bmatrix}_\times = \begin{bmatrix} u_2 \\ v_2 \\ 1 \end{bmatrix}_\times \begin{bmatrix} \mathbf{x}_1^\mathsf{T}\mathbf{h}_3^\mathsf{T} \\ \mathbf{x}_2^\mathsf{T}\mathbf{h}_3^\mathsf{T} \\ \mathbf{x}_3^\mathsf{T}\mathbf{h}_3^\mathsf{T} \end{bmatrix}$$

$$= \begin{bmatrix} 0 & -1 & v_2 \\ 1 & 0 & -u_2 \\ -v_2 & u_2 & 0 \end{bmatrix} \begin{bmatrix} \mathbf{x}_1^\mathsf{T} & \mathbf{0}_{1\times 3} & \mathbf{0}_{1\times 3} \\ \mathbf{0}_{1\times 3} & \mathbf{x}_1^\mathsf{T} & \mathbf{0}_{1\times 3} \\ \mathbf{0}_{1\times 3} & \mathbf{0}_{1\times 3} & \mathbf{x}_1^\mathsf{T} \end{bmatrix} \begin{bmatrix} \mathbf{h}_1^\mathsf{T} \\ \mathbf{h}_2^\mathsf{T} \\ \mathbf{h}_3^\mathsf{T} \end{bmatrix}$$

$$= \underbrace{\begin{bmatrix} \mathbf{0}_{1\times 3} & -\mathbf{x}_1^\mathsf{T} & v_2\mathbf{x}_1^\mathsf{T} \\ \mathbf{x}_1^\mathsf{T} & \mathbf{0}_{1\times 3} & -u_2\mathbf{x}_1^\mathsf{T} \\ -v_2\mathbf{x}_1^\mathsf{T} & u_2\mathbf{x}_1^\mathsf{T} & \mathbf{0}_{1\times 3} \end{bmatrix}}_{3 \times 9} \begin{bmatrix} \mathbf{h}_1^\mathsf{T} \\ \mathbf{h}_2^\mathsf{T} \\ \mathbf{h}_3^\mathsf{T} \end{bmatrix} = \mathbf{0} \longrightarrow \mathbf{A}\mathbf{x} = \mathbf{0}$$

# VO or V-SLAM pipeline (simplified)



# VO pipeline (some details)

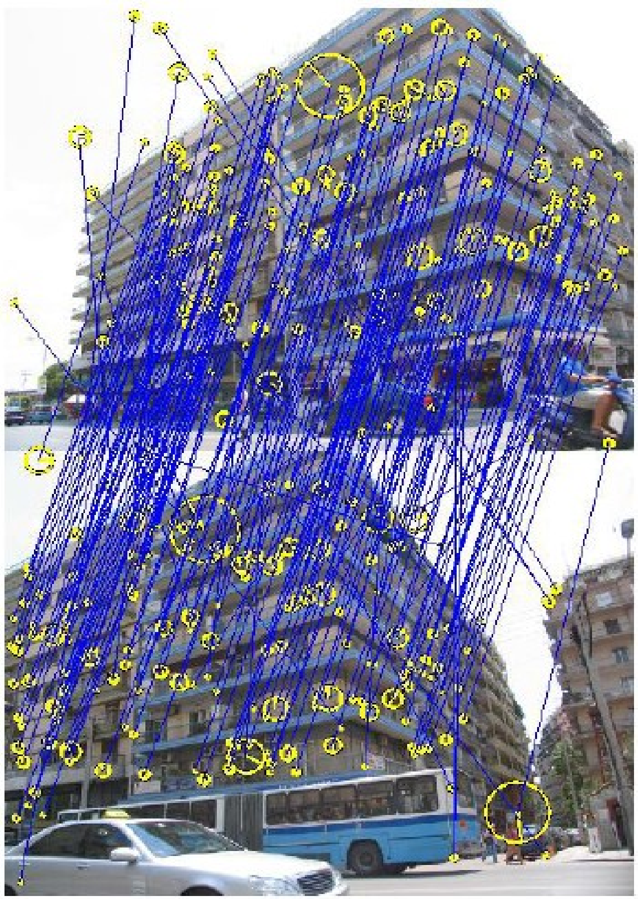
## Structure from Motion Pipeline

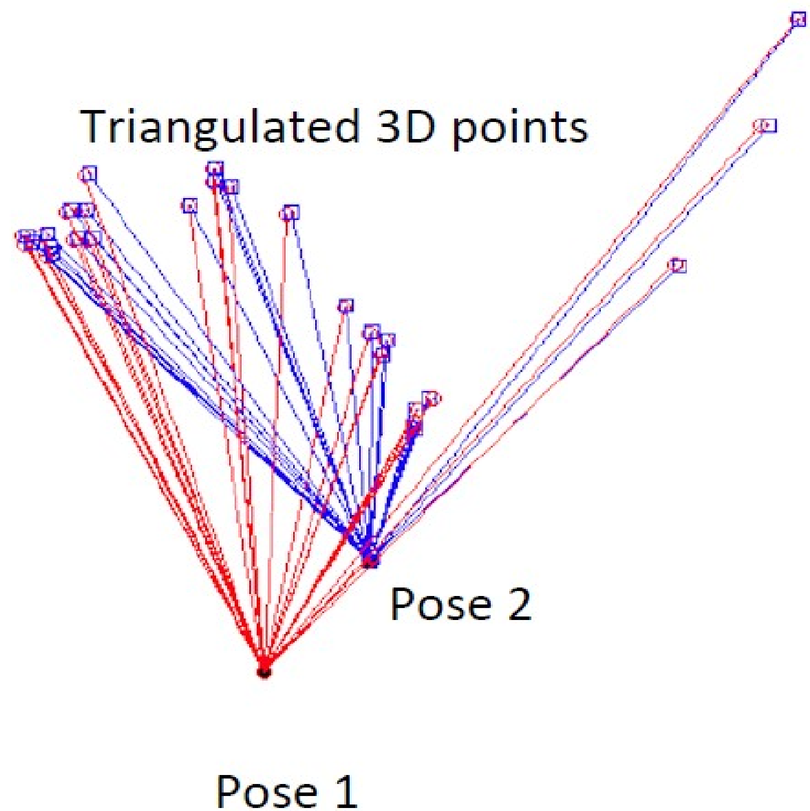
**Algorithm 1** Structure from Motion

```
for all possible pair of images do
    [x1 x2] = GetInliersRANSAC(x1, x2);                ▷ Reject outlier correspondences.
end for
F = EstimateFundamentalMatrix(x1, x2);                 ▷ Use the first two images.
E = EssentialMatrixFromFundamentalMatrix(F, K);
[Cset Rset] = ExtractCameraPose(E);
for i = 1 :  4 do
    Xset{i} = LinearTriangulation(K, zeros(3,1), eye(3), Cset{i}, Rset{i}, x1, x2);
end for
[C R] = DisambiguateCameraPose(Cset, Rset, Xset);      ▷ Check the cheirality condition.
X = NonlinearTriangulation(K, zeros(3,1), eye(3), C, R, x1, x2));
Cset← {C}, Rset← {R}
for i = 3 : I do                ▷ Register camera and add 3D points for the rest of images
    [Cnew Rnew] = PnPRANSAC(X, x, K);                  ▷ Register the i^th image.
    [Cnew Rnew] = NonlinearPnP(X, x, K, Cnew, Rnew);
    Cset ← Cset ∪ Cnew
    Rset ← Rset ∪ Rnew
    Xnew = LinearTriangulation(K, C0, R0, Cnew, Rnew, x1, x2);
    Xnew = NonlinearTriangulation(K, C0, R0, Cnew, Rnew, x1, x2);   ▷ Add 3D points.
    X ← X ∪ Xnew
    V = BuildVisibilityMatrix(traj);                   ▷ Get visibility matrix.
    [Cset Rset X] = BundleAdjustment(Cset, Rset, X, K, traj, V);    ▷ Bundle adjustment.
end for
```

# What do we need features for?

- For finding points so that we solve localization and reconstruction

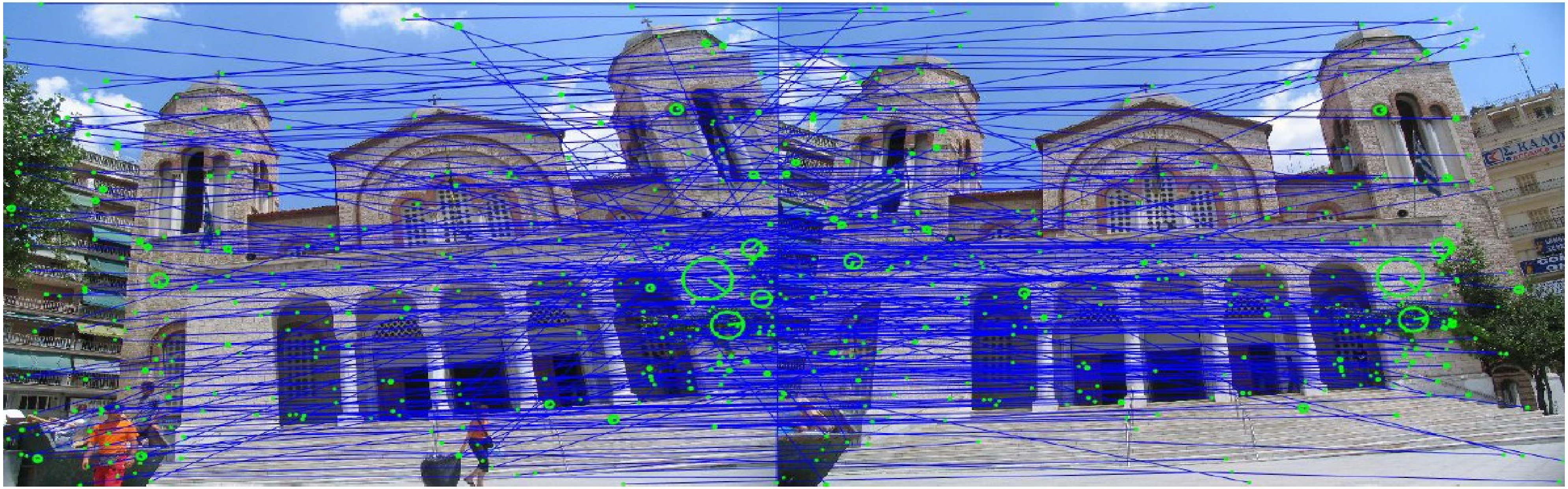




# Features wanted

## What else do we want from features?

- We should be able to match features using a descriptor of the neighborhood.
- This descriptor should not change significantly under viewpoint changes like scale and rotation.
- We call this property descriptor invariance.



Invariant detection and description

# 2D to 2D : motion estimation

- Objective : from image k and image k-1, estimate Rk (rotation) and Tk (translation) for our camera
  - Use only pixels coordinates
  - Assume camera is calibrated

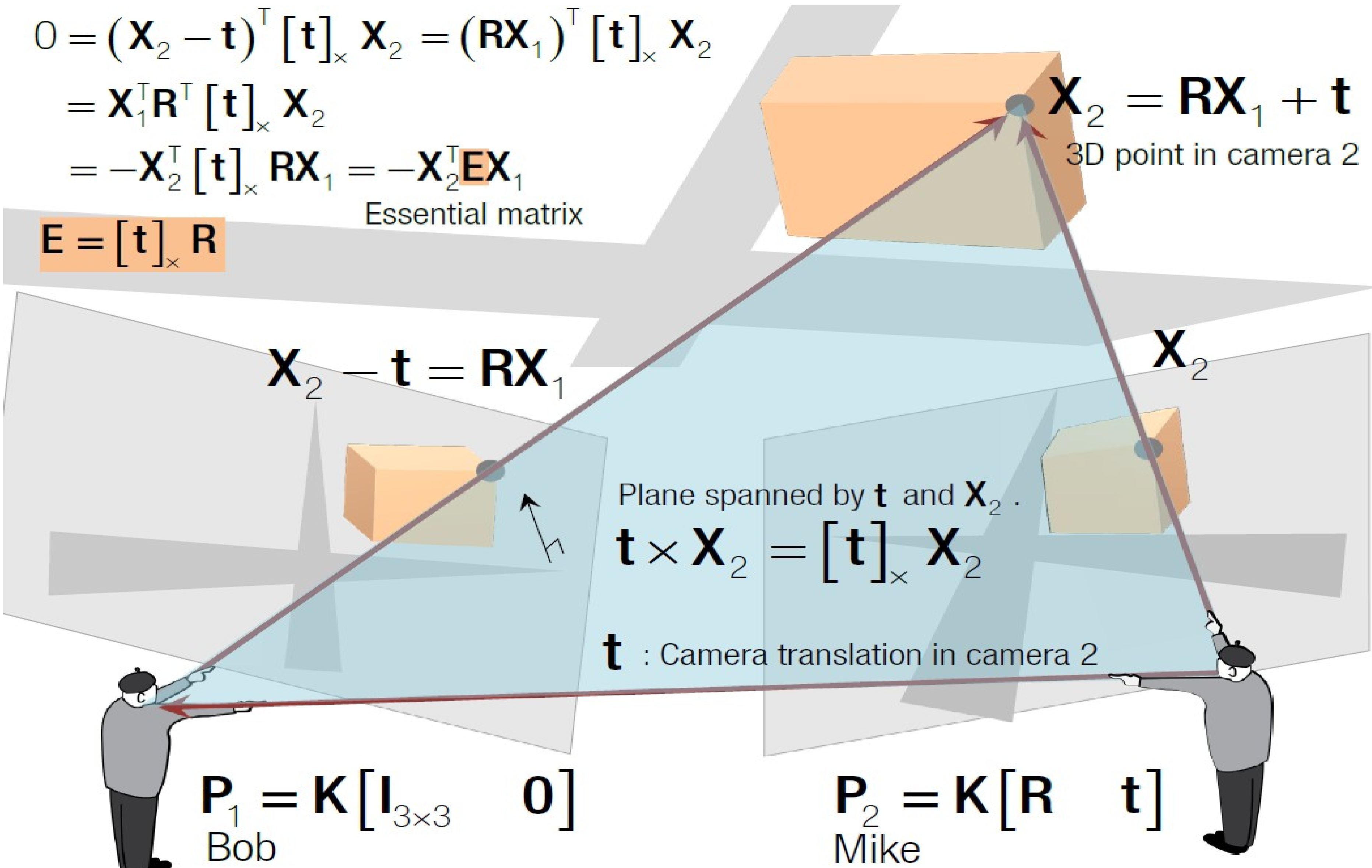
    (K intrinsics known, easier to handle)

# 2D to 2D : motion estimation

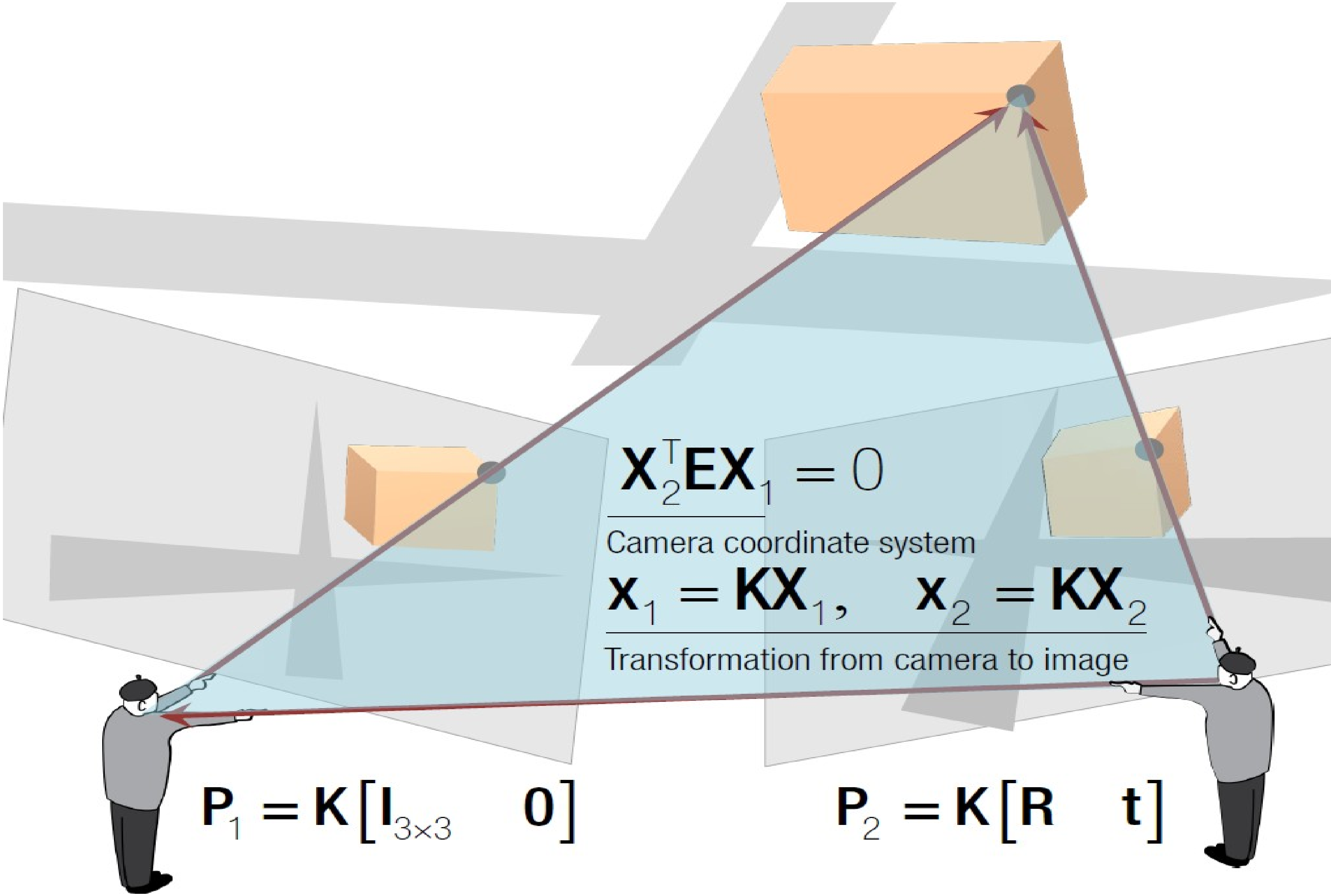
$$
\begin{aligned}
0 &= (\mathbf{X}_2 - \mathbf{t})^{\mathsf{T}} [\mathbf{t}]_\times \mathbf{X}_2 = (\mathbf{R}\mathbf{X}_1)^{\mathsf{T}} [\mathbf{t}]_\times \mathbf{X}_2 \\
&= \mathbf{X}_1^{\mathsf{T}} \mathbf{R}^{\mathsf{T}} [\mathbf{t}]_\times \mathbf{X}_2 \\
&= -\mathbf{X}_2^{\mathsf{T}} [\mathbf{t}]_\times \mathbf{R}\mathbf{X}_1 = -\mathbf{X}_2^{\mathsf{T}} \boxed{\mathbf{E}} \mathbf{X}_1
\end{aligned}
$$

Essential matrix

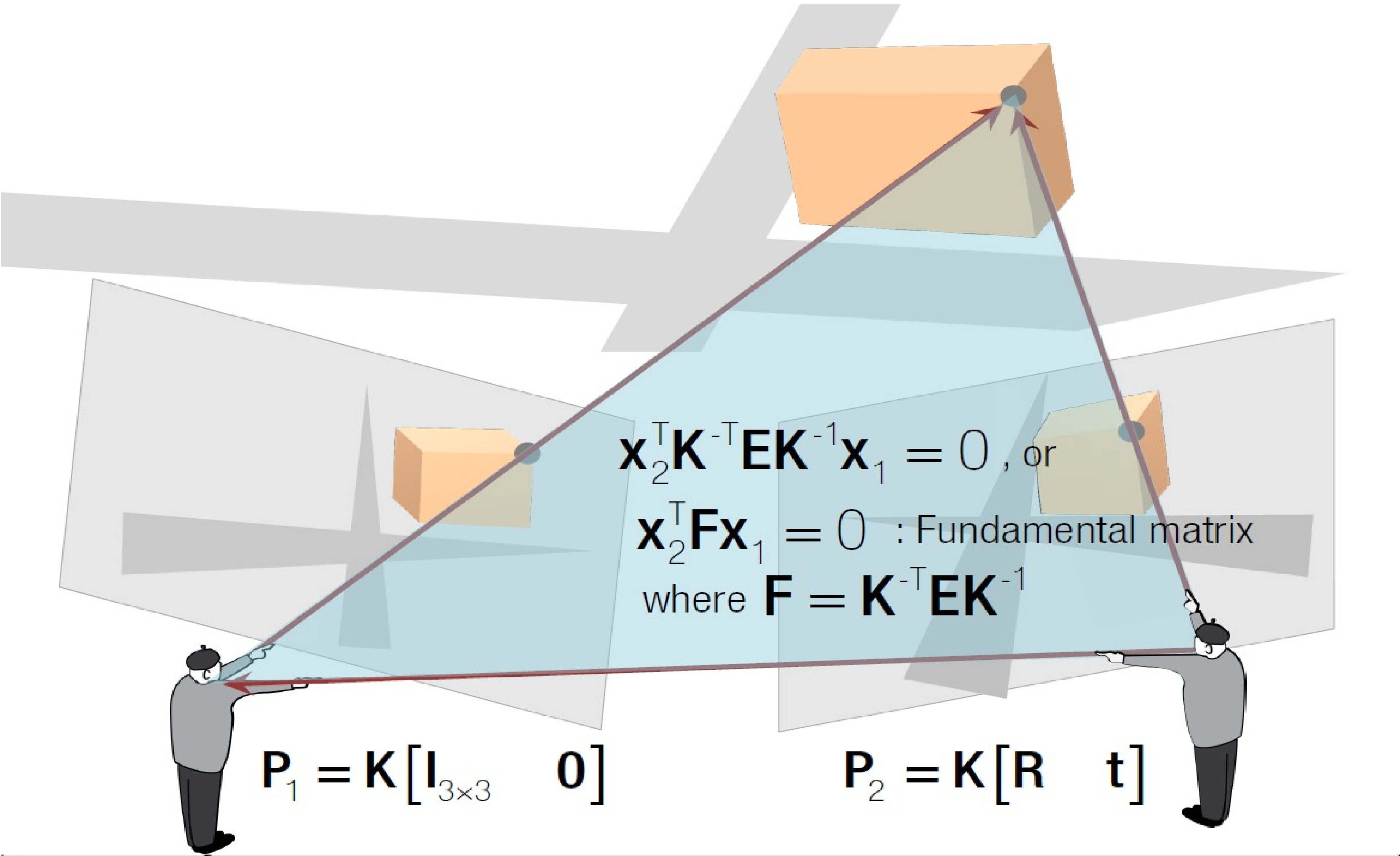
$$\boxed{\mathbf{E} = [\mathbf{t}]_\times \mathbf{R}}$$



# 2D to 2D : motion estimation



# 2D to 2D : motion estimation



# 2D to 2D : motion estimation

## 1. Pairwise Image Feature Matching

$\mathbf{x}_1$ $\mathbf{x}_2$

$$\mathbf{x}_2^{\mathsf{T}}\mathbf{F}\mathbf{x}_1 = 0$$

Epipolar constraint

$$\begin{bmatrix} u_1^1u_1^2 & u_1^1v_1^2 & u_1^1 & v_1^1u_1^2 & v_1^1v_1^2 & v_1^1 & u_1^2 & v_1^2 & 1 \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ u_8^1u_8^2 & u_8^1v_8^2 & u_8^1 & v_8^1u_8^2 & v_8^1v_8^2 & v_8^1 & u_8^2 & v_8^2 & 1 \end{bmatrix} \begin{bmatrix} f_{11} \\ f_{21} \\ f_{31} \\ f_{12} \\ \vdots \\ f_{23} \\ f_{31} \\ f_{32} \\ f_{33} \end{bmatrix} = 0$$

$$\mathbf{A}\;\mathbf{x} = \mathbf{0}$$

# Outliers rejection

## 1.2 Match Outlier Rejection via RANSAC

**Goal** Given $N$ correspondences between two images ($N \geq 8$), $\mathbf{x}_1 \leftrightarrow \mathbf{x}_2$, implement the following function that estimates inlier correspondences using fundamental matrix based RANSAC:

```
[y1 y2 idx] = GetInliersRANSAC(x1, x2)
```

(INPUT) `x1` and `x2`: $N\times 2$ matrices whose row represents a correspondence.

(OUTPUT) `y1` and `y2`: $N_i\times 2$ matrices whose row represents an inlier correspondence where $N_i$ is the number of inliers.

(OUTPUT) `idx`: $N\times 1$ vector that indicates ID of inlier `y1`.

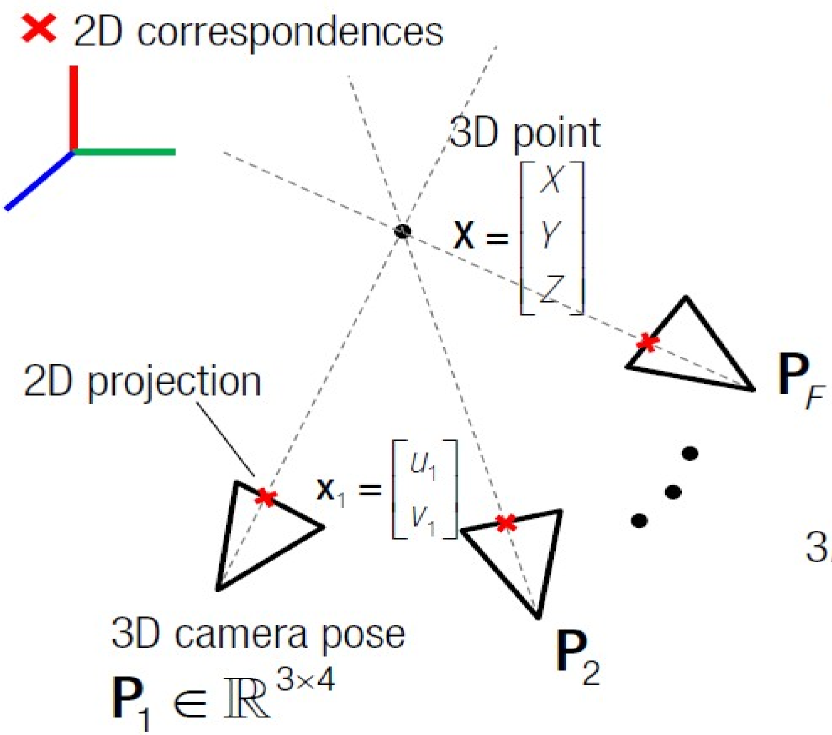
A pseudo code the RANSAC is shown in Algorithm 2.

**Algorithm 2** GetInliersRANSAC

$n \leftarrow 0$
**for** $i = 1 : M$ **do**
    Choose 8 correspondences, $\hat{\mathbf{x}}_1$ and $\hat{\mathbf{x}}_2$, randomly
    `F = EstimateFundamentalMatrix(`$\hat{\mathbf{x}}_1$`, `$\hat{\mathbf{x}}_2$`)`
    $\mathcal{S} \leftarrow \emptyset$
    **for** $j = 1 : N$ **do**
        **if** $|\mathbf{x}_{2j}^{\top}\mathbf{F}\mathbf{x}_{1j}| < \epsilon$ **then**
            $\mathcal{S} \leftarrow \mathcal{S} \cup \{j\}$
        **end if**
    **end for**
    **if** $n < |\mathcal{S}|$ **then**
        $n \leftarrow |\mathcal{S}|$
        $\mathcal{S}_{in} \leftarrow \mathcal{S}$
    **end if**
**end for**

# 2D to 3D : building a map

## Point Triangulation




$$\lambda \begin{bmatrix} \mathbf{x}_1 \\ 1 \end{bmatrix} = \mathbf{P}_1 \begin{bmatrix} \mathbf{X} \\ 1 \end{bmatrix} \longrightarrow \begin{bmatrix} \mathbf{x}_1 \\ 1 \end{bmatrix}_\times \mathbf{P}_1 \begin{bmatrix} \mathbf{X} \\ 1 \end{bmatrix} = \mathbf{0}$$

$$\begin{bmatrix} \mathbf{x}_2 \\ 1 \end{bmatrix}_\times \mathbf{P}_2 \begin{bmatrix} \mathbf{X} \\ 1 \end{bmatrix} = \mathbf{0}$$

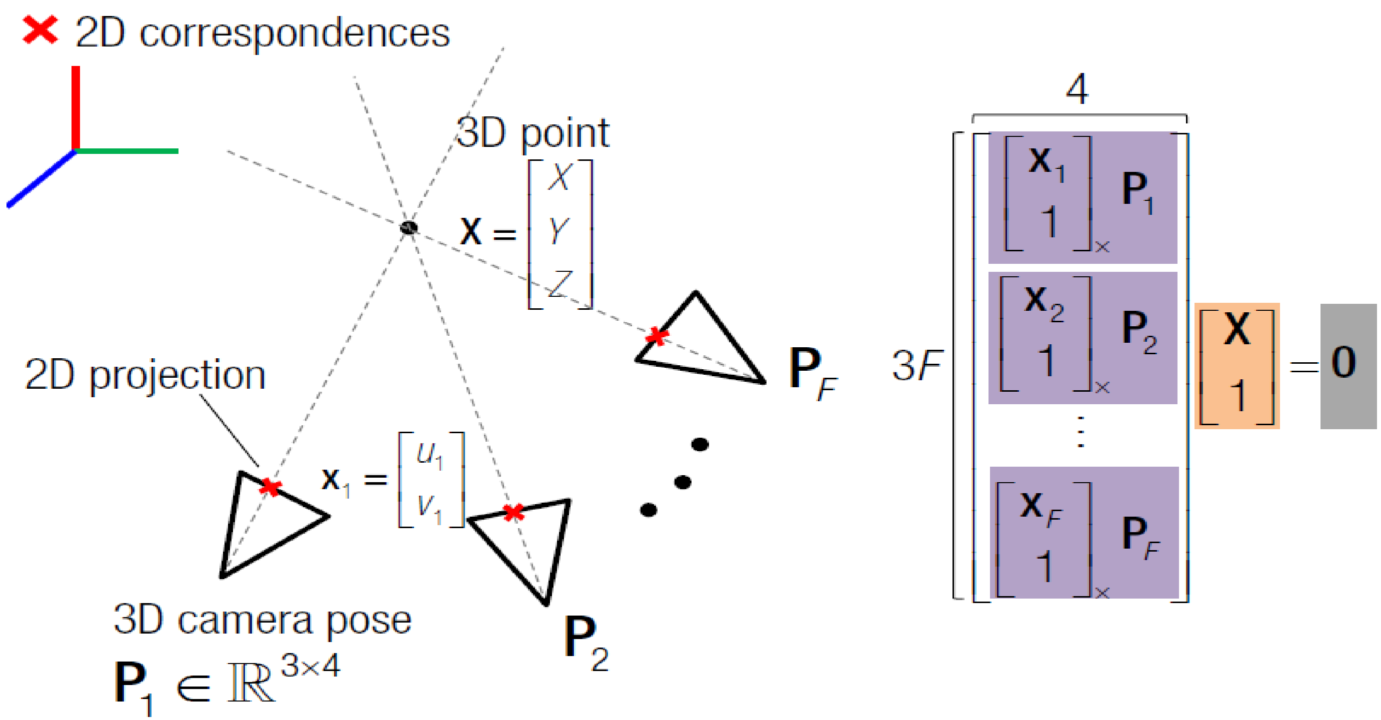
$$3F \overbrace{\begin{bmatrix} \begin{bmatrix} \mathbf{x}_1 \\ 1 \end{bmatrix}_\times \mathbf{P}_1 \\ \begin{bmatrix} \mathbf{x}_2 \\ 1 \end{bmatrix}_\times \mathbf{P}_2 \\ \vdots \\ \begin{bmatrix} \mathbf{x}_F \\ 1 \end{bmatrix}_\times \mathbf{P}_F \end{bmatrix}}^{4} \begin{bmatrix} \mathbf{X} \\ 1 \end{bmatrix} = \mathbf{0}$$

$$\mathrm{rank}\left(\begin{bmatrix} \mathbf{x} \\ 1 \end{bmatrix}_\times \mathbf{P}\right) = 2$$

Least squares if $F \geq 2$

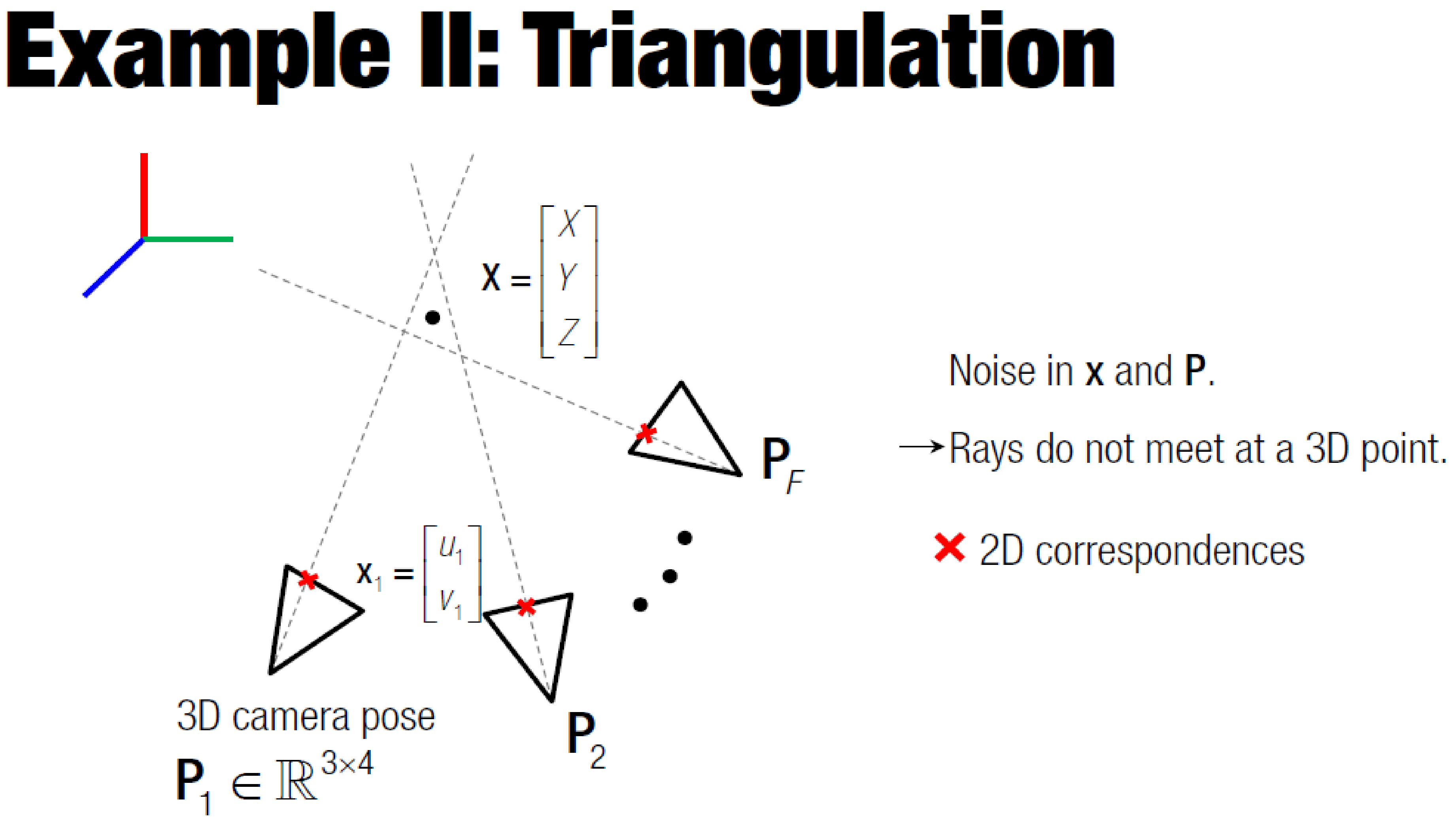
# 2D to 3D : building a map

## Example II: Triangulation




$\min_{\mathbf{x}} \|\mathbf{Ax} - \mathbf{b}\|^2$ Minimizes an algebraic error, i.e., there is no geometrical meaning.

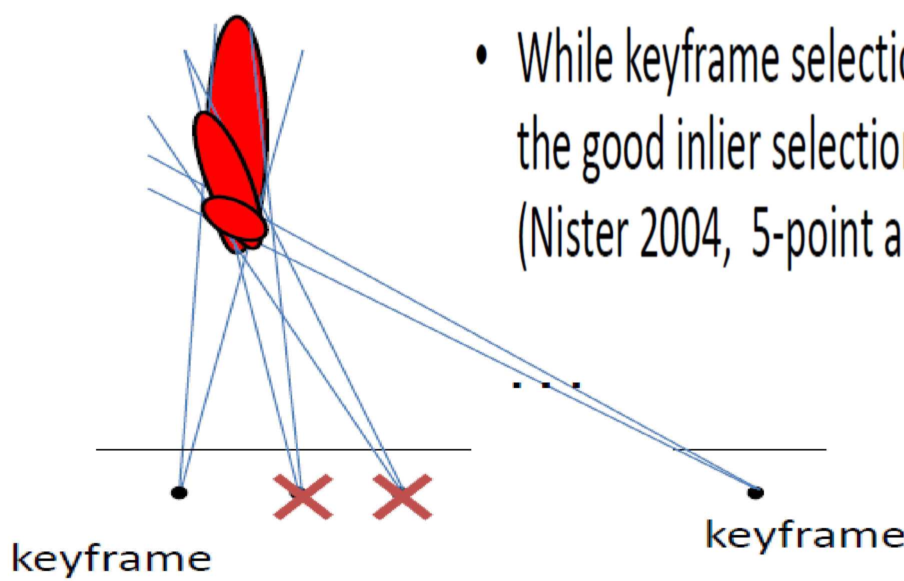
# Keyframe selection via error estimate



# Keyframe selection via error estimate

## Triangulation and Keyframe Selection

- Pose (translation) update depends on triangulated points whose error depends on baseline and distance.
- Wait until error in 3D triangulation decreases and then update pose: keyframe



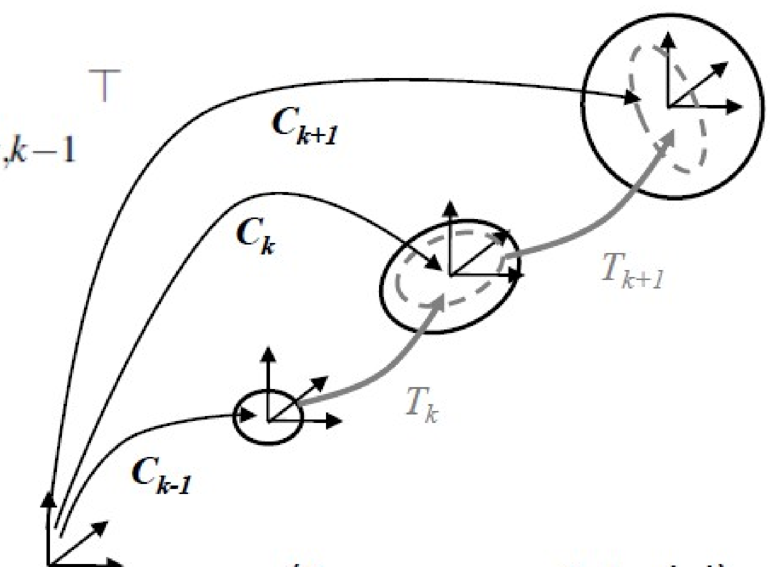

- While keyframe selection reduces drift, a large factor is the good inlier selection in point correspondences (Nister 2004, 5-point algorithm)

# Error Propagation

State Covariance is an estimate of its uncertainty. If uncertainty is Gaussian it can be visualized as an ellipsoid.

Its update depends on the previous uncertainty $\Sigma_{k-1}$, the measurement uncertainty $\Sigma_{k,k-1}$ , and the Jacobian with respect to state $J$.

$$\begin{aligned}\Sigma_k &= J\begin{bmatrix}\Sigma_{k-1} & 0 \\ 0 & \Sigma_{k,k-1}\end{bmatrix}J^\top \\ &= J_{\vec{C}_{k-1}}\Sigma_{k-1}J_{\vec{C}_{k-1}}{}^\top + J_{\vec{T}_{k,k-1}}\Sigma_{k,k-1}J_{\vec{T}_{k,k-1}}{}^\top\end{aligned}$$




(Scaramuzza Tutorial)

# 3D to 2D : motion & localization from a 3D map

## Camera 3D Registration

### Perspective-n-Point Algorithm

3D point cloud via triangulation

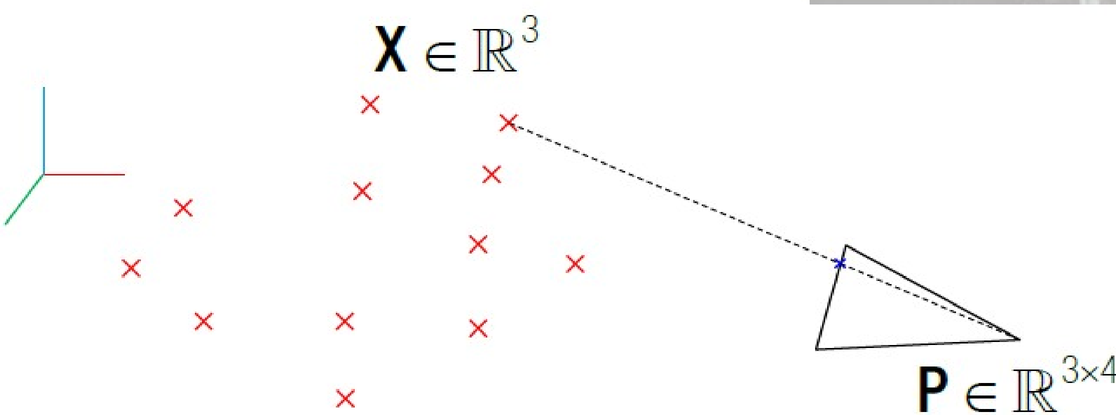
Where?

# 3D to 2D : motion & localization from a 3D map

## Perspective-n-Point


2D image

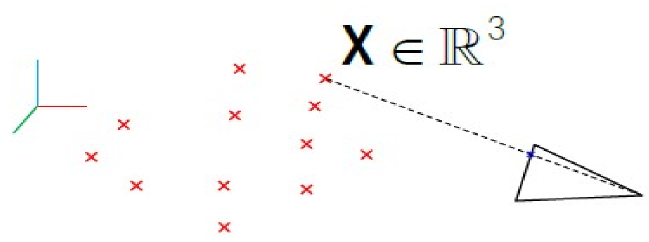
3D point cloud




$$\lambda \begin{bmatrix} \mathbf{x}_1 \\ 1 \end{bmatrix} = \mathbf{P} \begin{bmatrix} \mathbf{X}_1 \\ 1 \end{bmatrix} \longrightarrow \begin{bmatrix} \mathbf{x}_1 \\ 1 \end{bmatrix}_\times \mathbf{P} \begin{bmatrix} \mathbf{X}_1 \\ 1 \end{bmatrix} = \mathbf{0}$$

# 3D to 2D : motion & localization from a 3D map

## Perspective-n-Point





2D image

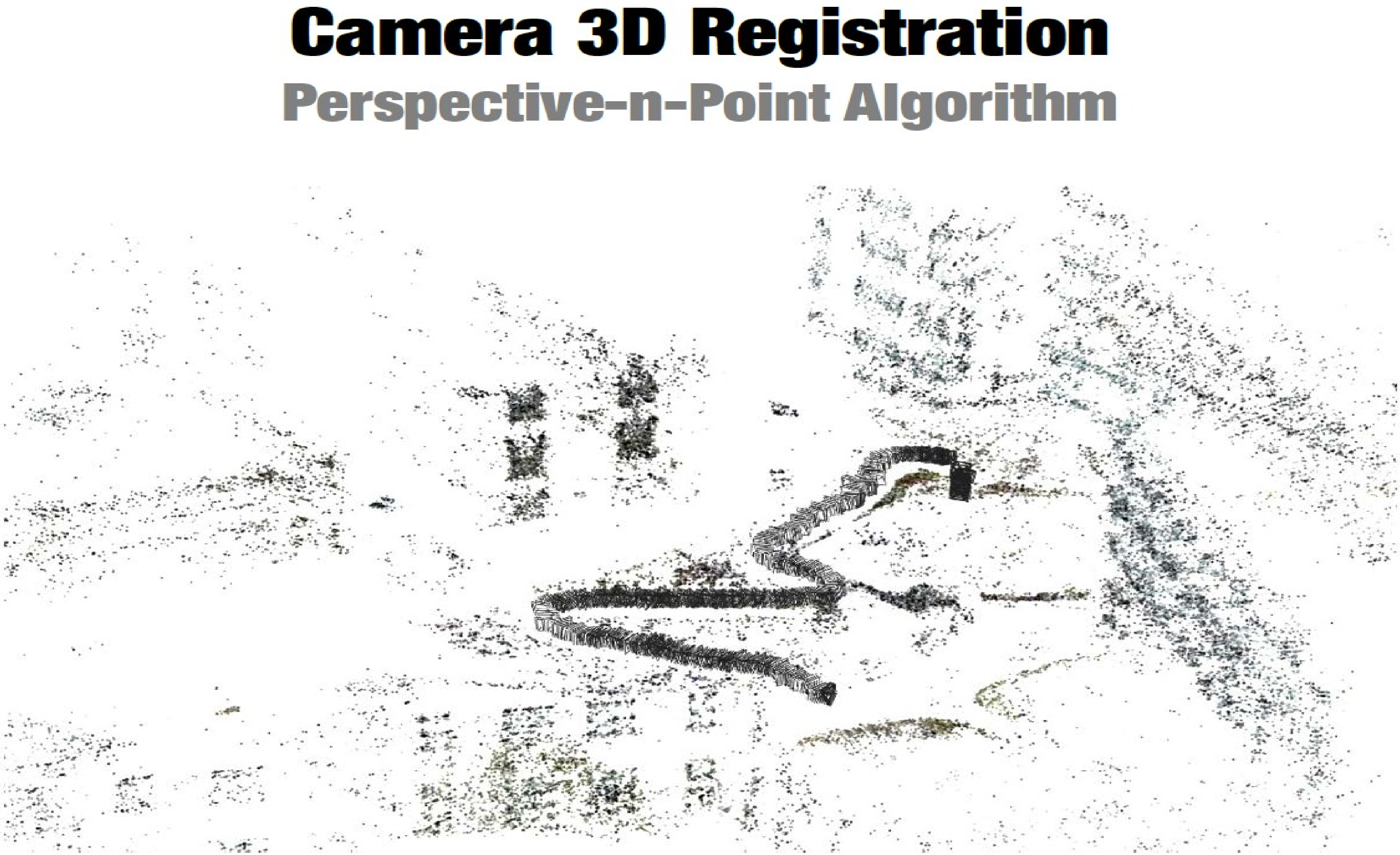
$$\begin{bmatrix} u \\ v \\ 1 \end{bmatrix}_\times \begin{bmatrix} \mathbf{P}_1 \\ \mathbf{P}_2 \\ \mathbf{P}_3 \end{bmatrix} \tilde{\mathbf{X}} = \begin{bmatrix} u \\ v \\ 1 \end{bmatrix}_\times \begin{bmatrix} \mathbf{P}_1\tilde{\mathbf{X}} \\ \mathbf{P}_2\tilde{\mathbf{X}} \\ \mathbf{P}_3\tilde{\mathbf{X}} \end{bmatrix} = \begin{bmatrix} 0 & -1 & v \\ 1 & 0 & -u \\ -v & u & 0 \end{bmatrix} \begin{bmatrix} \tilde{\mathbf{X}}^\mathsf{T} & \mathbf{0}_{1\times 4} & \mathbf{0}_{1\times 4} \\ \mathbf{0}_{1\times 4} & \tilde{\mathbf{X}}^\mathsf{T} & \mathbf{0}_{1\times 4} \\ \mathbf{0}_{1\times 4} & \mathbf{0}_{1\times 4} & \tilde{\mathbf{X}}^\mathsf{T} \end{bmatrix} \begin{bmatrix} \mathbf{P}_1^\mathsf{T} \\ \mathbf{P}_2^\mathsf{T} \\ \mathbf{P}_3^\mathsf{T} \end{bmatrix} =$$

$$\begin{bmatrix} \mathbf{0}_{1\times 4} & -\tilde{\mathbf{X}}^\mathsf{T} & v\tilde{\mathbf{X}}^\mathsf{T} \\ \tilde{\mathbf{X}}^\mathsf{T} & \mathbf{0}_{1\times 4} & -u\tilde{\mathbf{X}}^\mathsf{T} \\ -v\tilde{\mathbf{X}}^\mathsf{T} & u\tilde{\mathbf{X}}^\mathsf{T} & \mathbf{0}_{1\times 4} \end{bmatrix} \begin{bmatrix} \mathbf{P}_1^\mathsf{T} \\ \mathbf{P}_2^\mathsf{T} \\ \mathbf{P}_3^\mathsf{T} \end{bmatrix} = \mathbf{0}$$

3x12 matrix

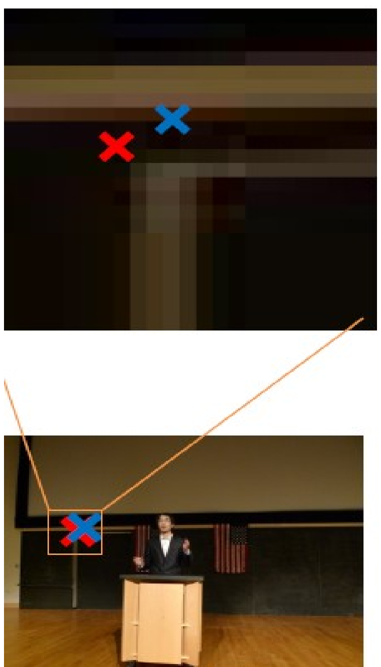
$$\mathbf{A} \; \mathbf{x} = \mathbf{0}$$

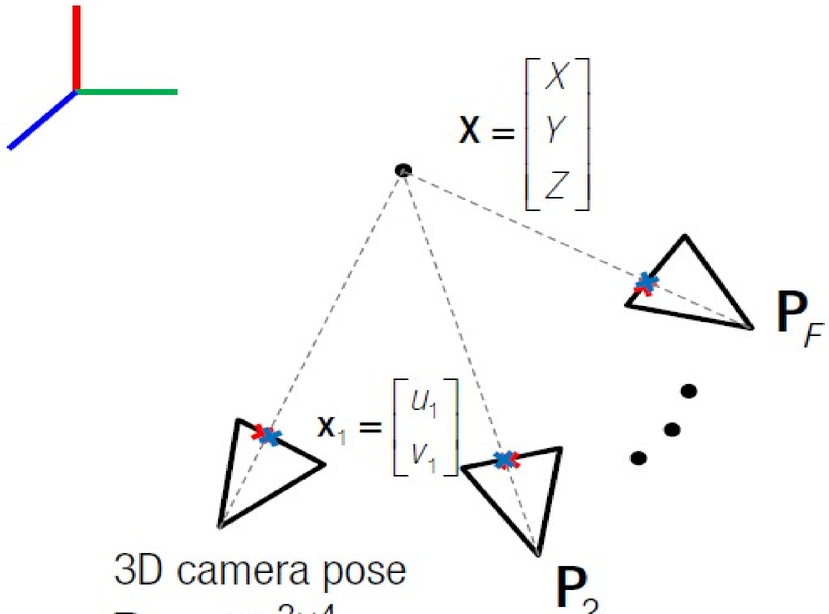
# 3D to 2D : motion & localization from a 3D map



# Bundle adjustment

## Reprojection Error (Geometric Error)






3D camera pose
$\mathbf{P}_1 \in \mathbb{R}^{3\times 4}$

✖ $(u, v)$

✖ $(u_{\text{repro}}, v_{\text{repro}}) = \left( \dfrac{\mathbf{P}^1\tilde{\mathbf{X}}}{\mathbf{P}^3\tilde{\mathbf{X}}}, \dfrac{\mathbf{P}^2\tilde{\mathbf{X}}}{\mathbf{P}^3\tilde{\mathbf{X}}} \right)$

where $\mathbf{P} = \begin{bmatrix} \mathbf{P}^1 \\ \mathbf{P}^2 \\ \mathbf{P}^3 \end{bmatrix}$ and $\tilde{\mathbf{X}} = \begin{bmatrix} \mathbf{X} \\ 1 \end{bmatrix}$

$$E_{\text{repro}} = \left(u - u_{\text{repro}}\right)^2 + \left(v - v_{\text{repro}}\right)^2$$
$$= \left(u - \frac{\mathbf{P}^1\tilde{\mathbf{X}}}{\mathbf{P}^3\tilde{\mathbf{X}}}\right)^2 + \left(v - \frac{\mathbf{P}^2\tilde{\mathbf{X}}}{\mathbf{P}^3\tilde{\mathbf{X}}}\right)^2$$

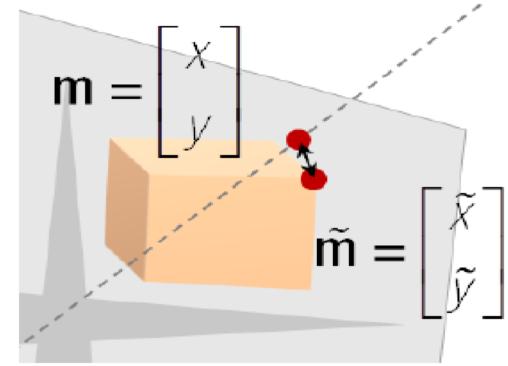
$$\mathbf{f}\left(\begin{bmatrix} X \\ Y \\ Z \end{bmatrix}\right) = \begin{bmatrix} u_1 \\ v_1 \\ \vdots \\ u_F \\ v_F \end{bmatrix}$$

Nonlinear least squares

# Bundle Adjustment
# (simplified single point, single pose)

**Reprojection error**

$$e = \begin{bmatrix} \tilde{x} \\ \tilde{y} \end{bmatrix} - \begin{bmatrix} u/w \\ v/w \end{bmatrix} \quad \text{where} \quad \begin{bmatrix} u \\ v \\ w \end{bmatrix} = \mathbf{P}\begin{bmatrix} \mathbf{X} \\ 1 \end{bmatrix} = \mathbf{K}\mathbf{R}\begin{bmatrix} \mathbf{I}_{3\times 3} & -\mathbf{C} \end{bmatrix}\begin{bmatrix} \mathbf{X} \\ 1 \end{bmatrix}$$

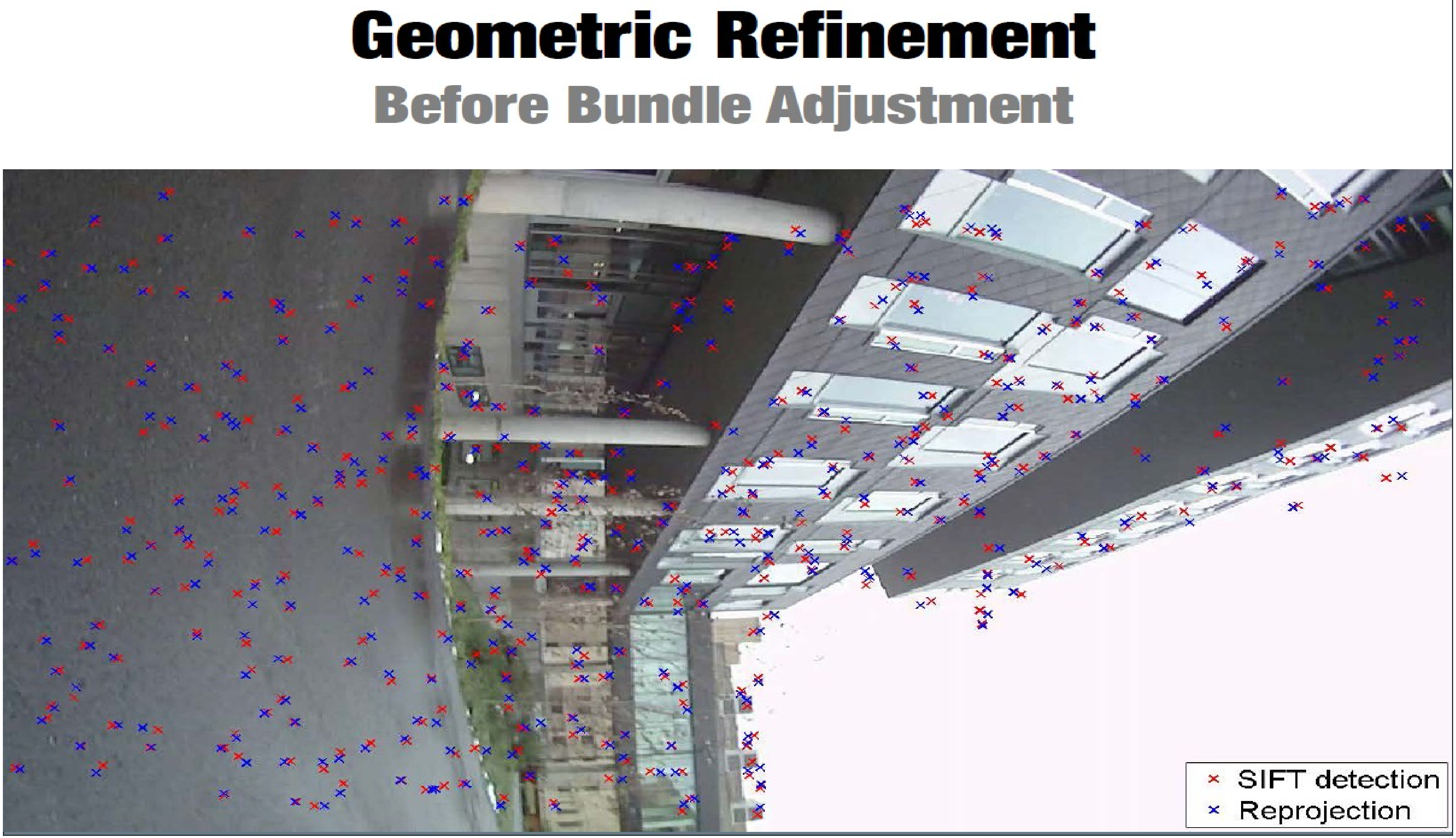



$$\underset{\mathbf{q},\mathbf{C},\mathbf{X}}{\text{minimize}} \left\| \begin{bmatrix} \tilde{x} \\ \tilde{y} \end{bmatrix} - \begin{bmatrix} u(\mathbf{R}(\mathbf{q}),\mathbf{C},\mathbf{X}) / w(\mathbf{R}(\mathbf{q}),\mathbf{C},\mathbf{X}) \\ v(\mathbf{R}(\mathbf{q}),\mathbf{C},\mathbf{X}) / w(\mathbf{R}(\mathbf{q}),\mathbf{C},\mathbf{X}) \end{bmatrix} \right\|^2 = \underset{\mathbf{q},\mathbf{C},\mathbf{X}}{\text{minimize}} \left\| \mathbf{b} - \mathbf{f}(\mathbf{R}(\mathbf{q}),\mathbf{C},\mathbf{X}) \right\|^2$$

$$\mathbf{J} = \begin{bmatrix} \dfrac{\partial \mathbf{f}(\mathbf{R}(\mathbf{q}),\mathbf{C},\mathbf{X})}{\partial \mathbf{q}} & \dfrac{\partial \mathbf{f}(\mathbf{R}(\mathbf{q}),\mathbf{C},\mathbf{X})}{\partial \mathbf{C}} & \dfrac{\partial \mathbf{f}(\mathbf{R}(\mathbf{q}),\mathbf{C},\mathbf{X})}{\partial \mathbf{X}} \end{bmatrix}$$

$$= \begin{bmatrix} \underbrace{\dfrac{\partial \mathbf{f}(\mathbf{R}(\mathbf{q}),\mathbf{C},\mathbf{X})}{\partial \mathbf{R}}}_{2\text{x}9} \underbrace{\dfrac{\partial \mathbf{R}}{\partial \mathbf{q}}}_{9\text{x}4} & \underbrace{\dfrac{\partial \mathbf{f}(\mathbf{R}(\mathbf{q}),\mathbf{C},\mathbf{X})}{\partial \mathbf{C}}}_{2\text{x}3} & \underbrace{\dfrac{\partial \mathbf{f}(\mathbf{R}(\mathbf{q}),\mathbf{C},\mathbf{X})}{\partial \mathbf{X}}}_{2\text{x}3} \end{bmatrix}$$

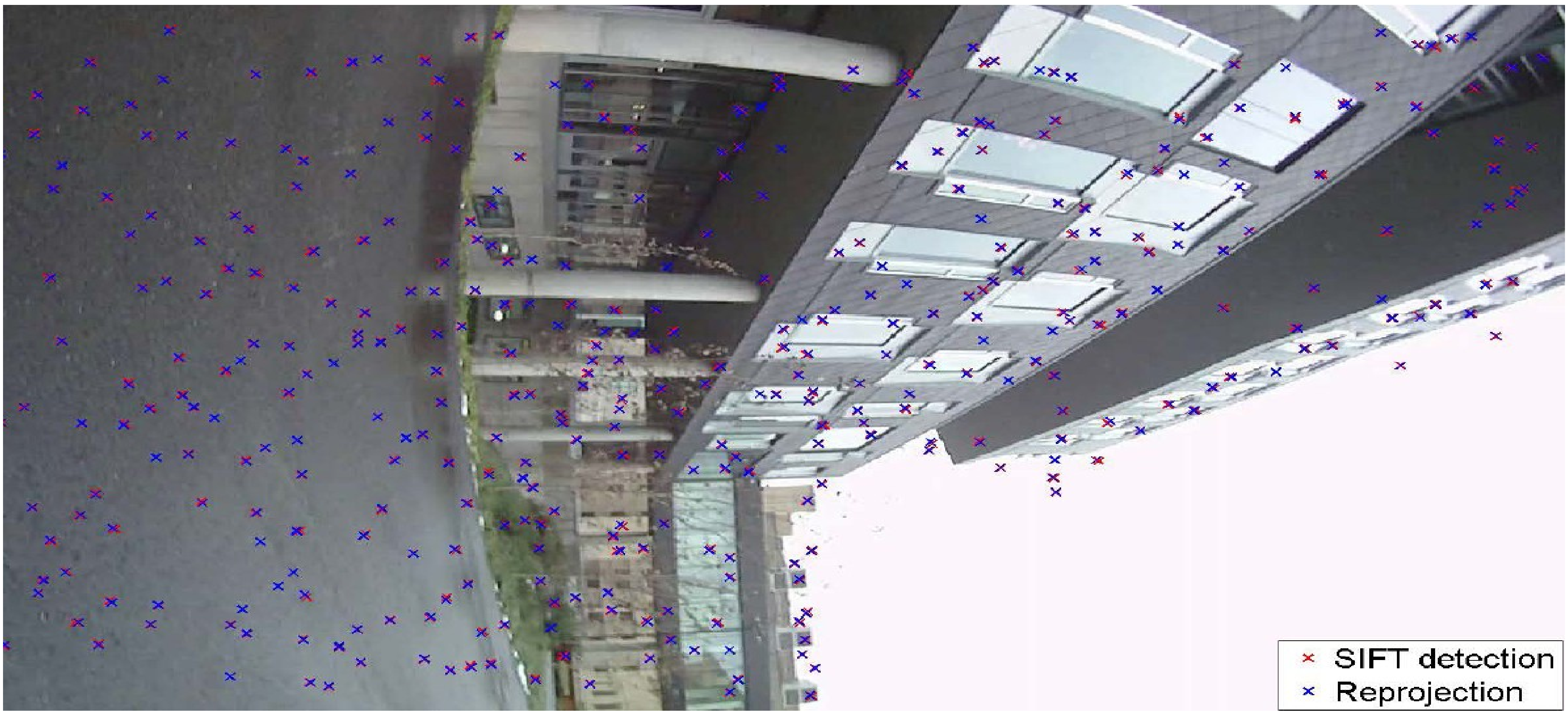
# Bundle adjustment



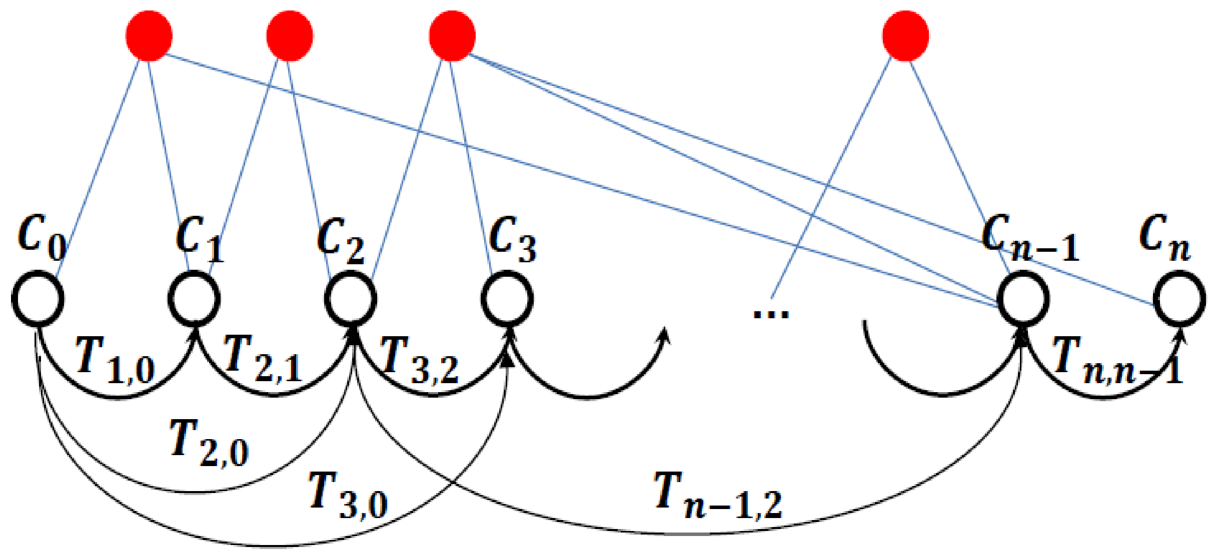
# Bundle adjustment

## Geometric Refinement

### After Bundle Adjustment



# Bundle Adjustment (BA)




- Similar to pose-graph optimization but it also optimizes 3D points

$$\arg\min_{X^i, C_k} \sum_{i,k} \| p_k^i - g(X^i, C_k) \|^2$$

- In order to not get stuck in local minima, the initialization should be close to the minimum
- Gauss-Newton or Levenberg-Marquadt can be used. For large graphs, efficient open-source software exists: GTSAM, g2o, Google Ceres can be used.

# Loop closure detection

- Typicaly done in 3 steps :
  - 1) bag of visual words : to identify visual similarity between 2 images (place recognition)
  - 2) After finding the top-n similar images, usually a geometric verification using the epipolar constraint is performed
  - 3) Rigid-body transformation is computed and added to the pose graph as an additional loop constraint for the pose-graph optimization

  => Enables to minimize errors in our parameter estimations

# Summary of Visual Odometry Tools

- Bundle Adjustment over a window
- Keyframe Selection
- RANSAC for 5-points or reduced minimal problem with 3 points.
- Visual Closing to produce unique trajectories when places are revisited

# Leverage on OpenCV, g2o, sba, orbslam...

- Most of the key building blocks are provided by OpenCV (features detection and matching, RANSAC, Essential and Fundamental matrix estimation, PnP triangulation …)
- Optimization problems are handled by existing open source libraries : https://github.com/RainerKuemmerle/g2o, http://users.ics.forth.gr/~lourakis/sba/
- State of the art V-SLAM framework : https://github.com/raulmur/ORB_SLAM2