# Perception:
# Pose Estimation

# Recall: Perspective N Points

Given N correspondences of points:

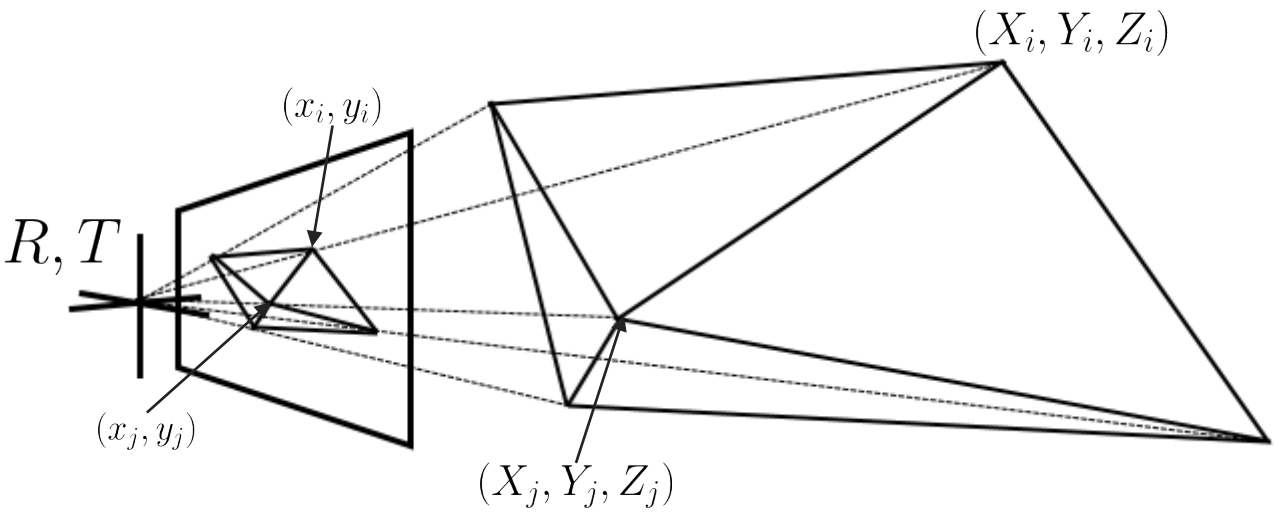
$$(x_i, y_i) \leftrightarrow (X_i, Y_i, Z_i)$$

Find $R, T$ that satisfy (for some $\lambda_i$):

$$\lambda_i \begin{pmatrix} x_i \\ y_i \\ 1 \end{pmatrix} = R \begin{pmatrix} X_i \\ Y_i \\ Z_i \end{pmatrix} + T$$

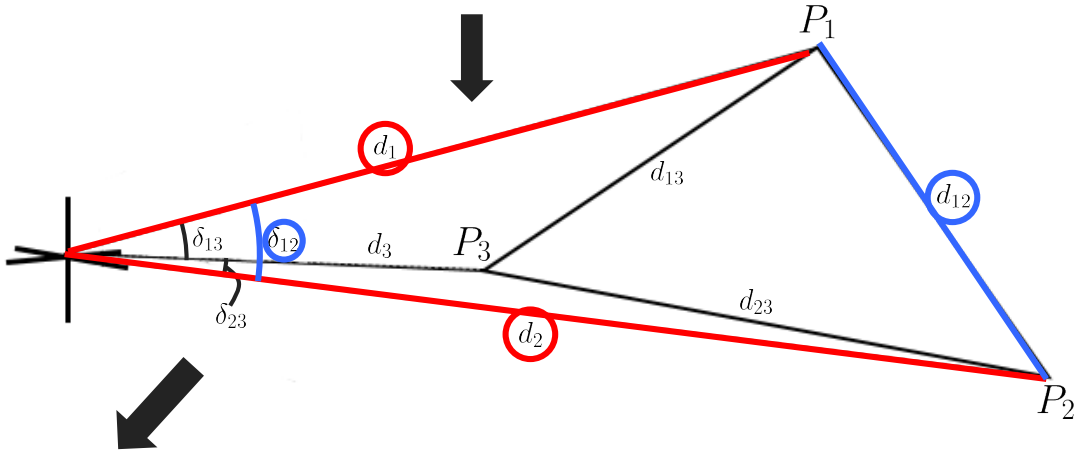
Assumed that 2D coordinates are calibrated, i.e. for pixel coordinates $u_i, v_i$ :

$$\begin{pmatrix} x_i \\ y_i \\ 1 \end{pmatrix} = K^{-1} \begin{pmatrix} u_i \\ v_i \\ 1 \end{pmatrix}$$



# Recall: Solving P3P

$$(x_i, y_i) \leftrightarrow (X_i, Y_i, Z_i)$$




$$\begin{aligned} d_1^2 + d_2^2 - 2d_1 d_2 \cos(\delta_{12}) &= d_{12}^2 \\ d_1^2 + d_3^2 - 2d_1 d_3 \cos(\delta_{13}) &= d_{13}^2 \\ d_2^2 + d_3^2 - 2d_2 d_3 \cos(\delta_{23}) &= d_{23}^2 \end{aligned}$$

$$\begin{aligned} d_{13}^2(u^2 + v^2 - 2uv\cos(\delta_{23})) &= d_{23}^2(1 + v^2 - 2v\cos(\delta_{13})) \\ d_{12}^2(1 + v^2 - 2v\cos(\delta_{13})) &= d_{13}^2(1 + u^2 - 2u\cos(\delta_{12})) \end{aligned}$$

$$\begin{aligned} & u^2 = a_1 v^2 + a_2 v + a_3 uv + a_4 \\ \Longrightarrow\ & u = (b_1 v^2 + b_2 v + b_3)/(b_4 v + b_5) \\ \Longrightarrow\ & \alpha v^4 + \beta v^3 + \gamma v^2 + \zeta v + \eta = 0 \end{aligned}$$

4 solutions

# Why PnP?

Finding the pose of an observed object

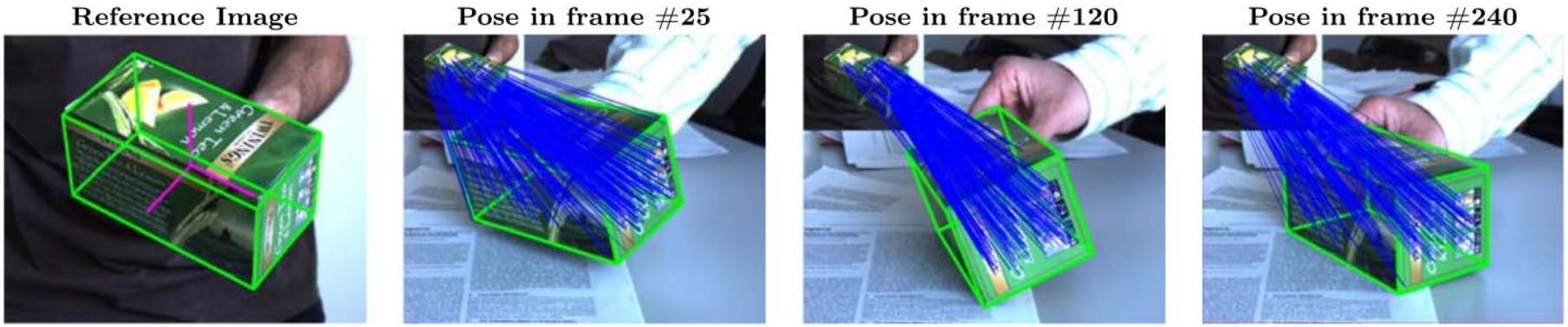



Image from: EPnP: An Accurate O(n) Solution to the PnP Problem, Lepetit et al. 2008

# Why PnP?

Converse problem: camera pose from an object of known dimensions

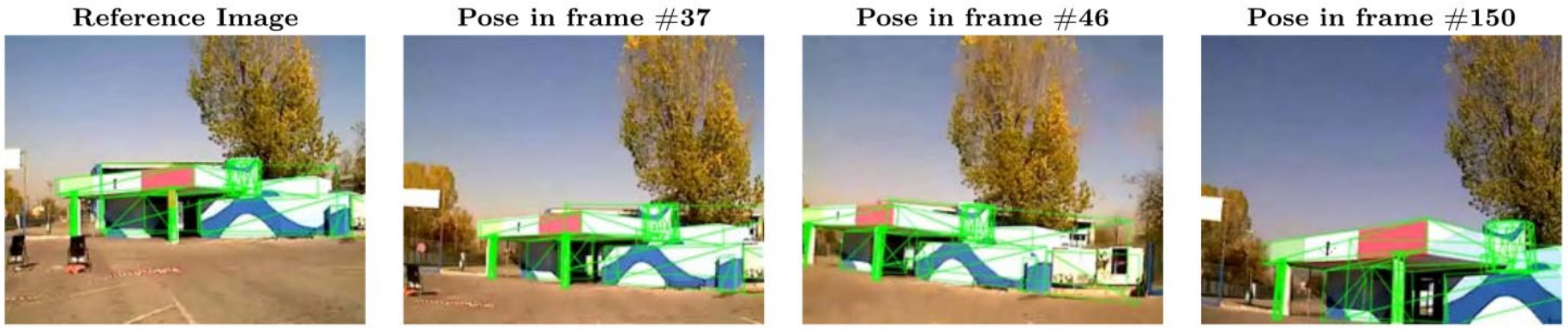
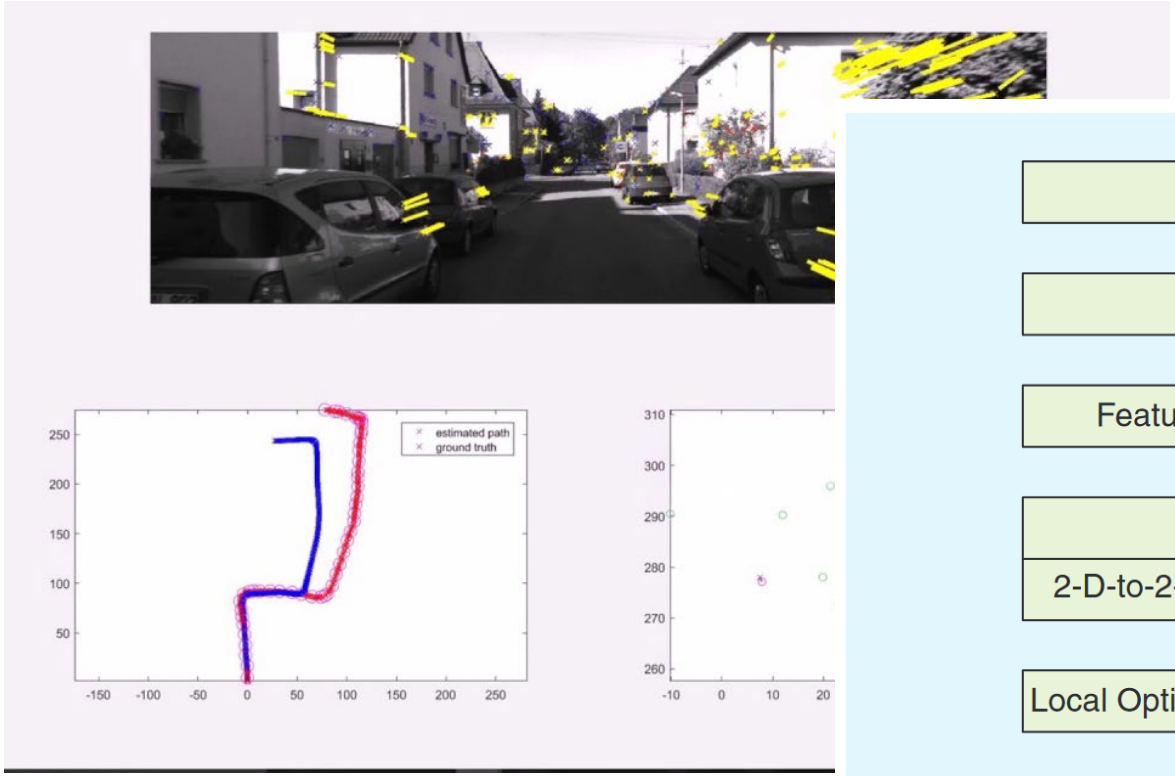



Image from: EPnP: An Accurate O(n) Solution to the PnP Problem, Lepetit et al. 2008

# Why PnP?

Used as an additional source of pose information in Visual Odometry pipelines (more explored later in the class)



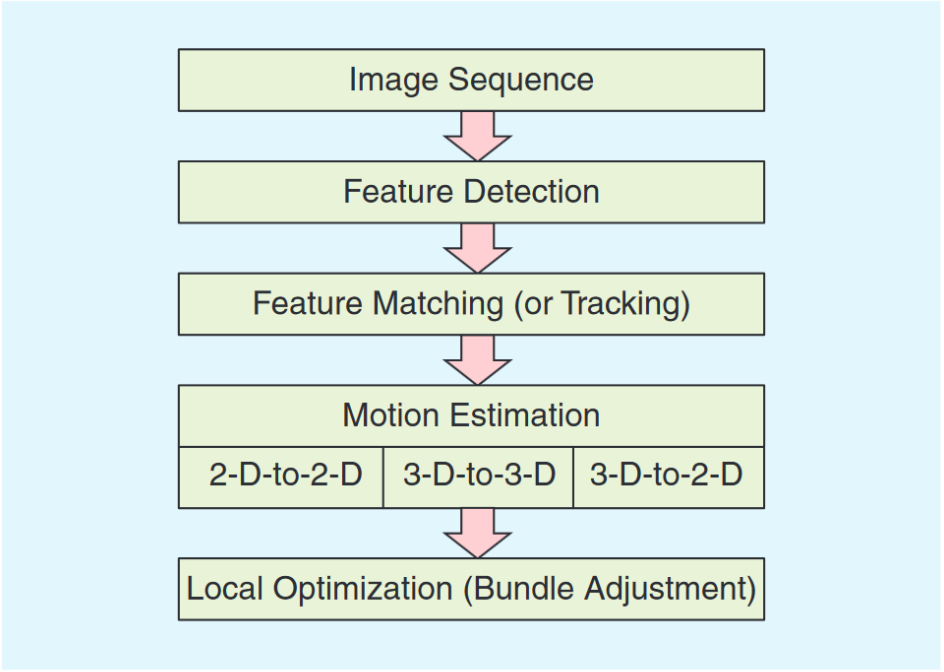




**Figure 2.** A block diagram showing the main components of a VO system.

Diagram from Visual Odometry, Scaramuzza and Fraundorfer, 2011

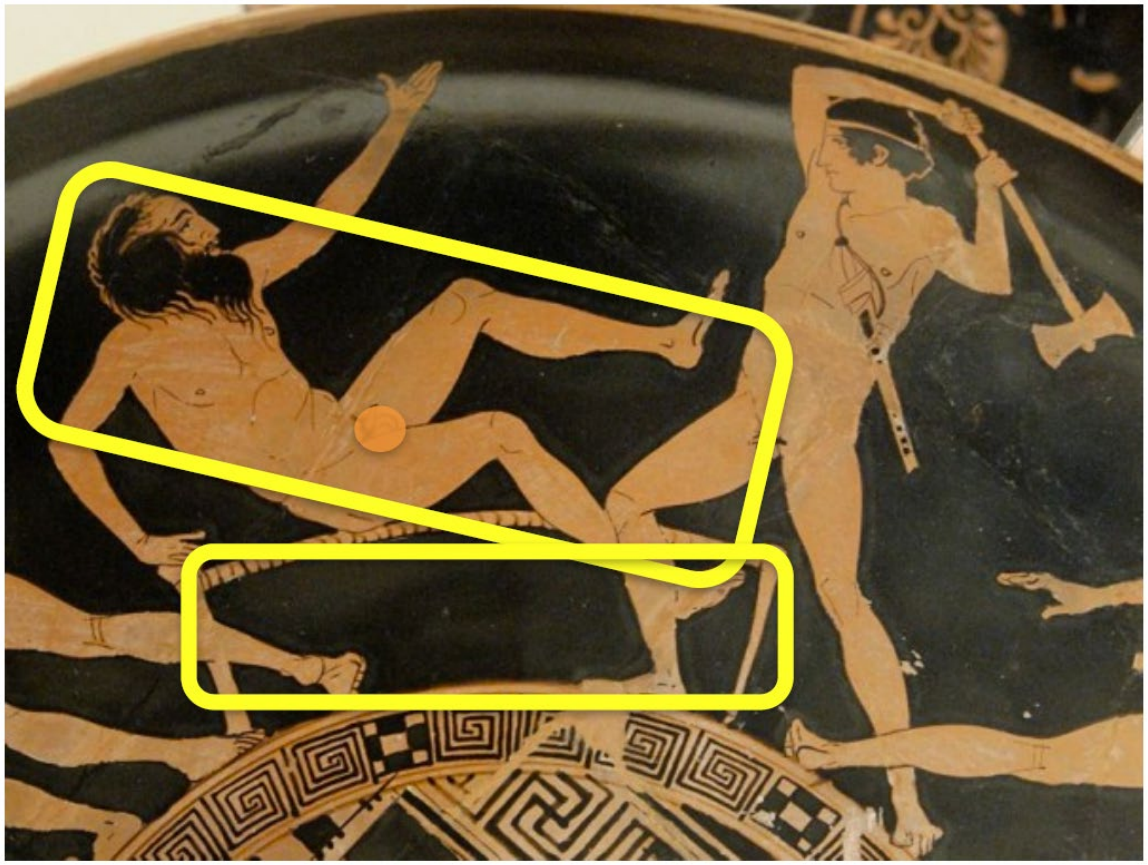
# Recall: 3D to 3D point matching

We solved for the distances of the points using P3P, but we didn't know what the actual rotation and translations were

$$d_i \begin{pmatrix} x_i \\ y_i \\ 1 \end{pmatrix} = R \begin{pmatrix} X_i \\ Y_i \\ Z_i \end{pmatrix} + T$$

Thus we need to match 3D points to 3D points using a rigid transformation – this is known as the Procrustes problem

# Historical Trivia: Bed of Procrustes



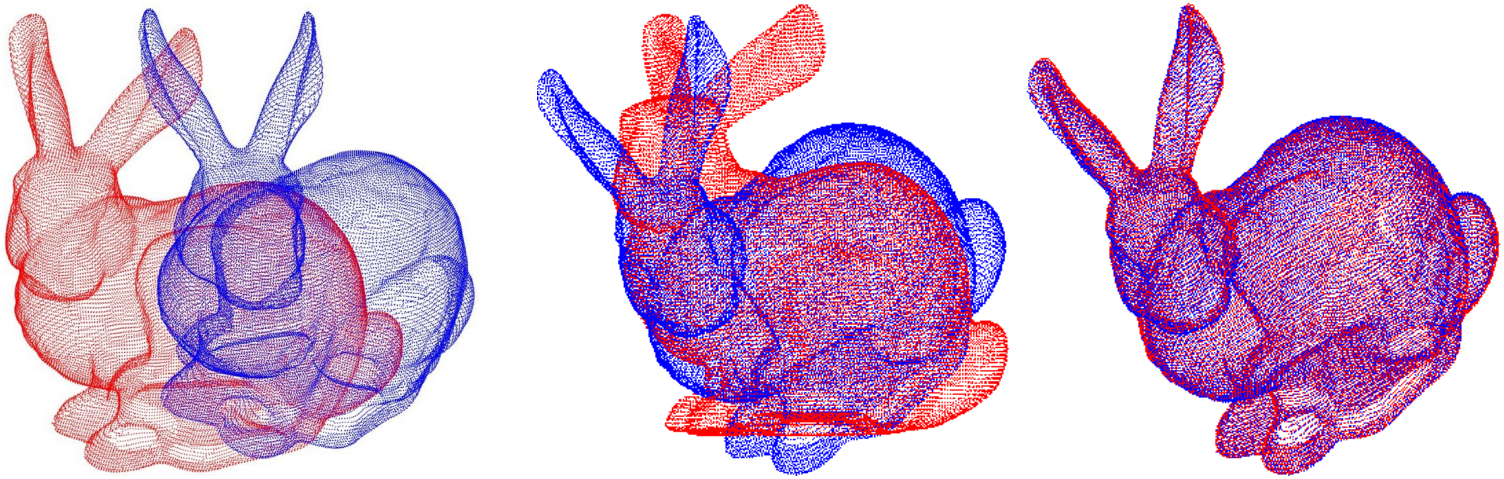
Namesake of this problem came from the ancient Greek story of Procrustes, a rogue smith who captured travelers and tried to make them fit into a bed by stretching or cutting them

Theseus defeated him

# Formal Problem Statement

Given corresponding sets of $N$ points $\{(P_i, P_i')\}$, we want to find the rotation and translation that minimizes their distance

$$\arg\min_{R \in \mathrm{SO}(3), T \in \mathbb{R}^3} \sum_i \|RP_i + T - P_i'\|^2$$



# Solving Procrustes

Let's expand the loss function:

$$\sum_i \|RP_i + T - P_i'\|^2$$

$$= \sum_i (RP_i + T - P_i')^T (RP_i + T - P_i')$$

$$= \sum_i (P_i^T R^T + T^T - P_i'^T)(RP_i + T - P_i')$$

$$= \sum_i P_i^T R^T RP_i + P_i^T R^T T - P_i^T R^T P_i' + T^T RP_i + T^T T - T^T P_i' - P_i'^T RP_i - P_i'^T T + P_i'^T P_i'$$

$$= \sum_i 2(P_i^T R^T - P_i'^T)T + N\,T^T T - 2P_i'^T RP_i + P_i^T P_i + P_i'^T P_i'$$

$$= 2\left(\sum_i RP_i - P_i'\right)^T T + N\,T^T T - 2\sum_i \left(P_i'^T RP_i\right) + \sum_i P_i^T P_i + \sum_i P_i'^T P_i'$$

# Solving Procrustes

We can actually solve for $T$ by taking the derivative:

$$\frac{\partial}{\partial T}\left(2\left(\sum_i RP_i - P_i'\right)^T T + N\,T^T T - 2\sum_i \left(P_i'^T RP_i\right) + \sum_i P_i^T P_i + \sum_i P_i'^T P_i'\right)$$

$$= \frac{\partial}{\partial T}\left(2\left(\sum_i RP_i - P_i'\right)^T T + N\,T^T T\right)$$

$$= 2\left(\sum_i RP_i - P_i'\right) + 2NT$$

Set the derivative to zero:

$$\Longrightarrow T = \frac{1}{N}\sum_i P_i' - RP_i = \overline{P'} - R\overline{P}$$

So the translation is just the difference between the centroids

# Solving Procrustes

By subtracting the centroids of the points beforehand, we can get rid of the translation term altogether:

$$\tilde{P}_i = P_i - \overline{P}, \quad \tilde{P}'_i = P'_i - \overline{P'}$$

With this, our new loss function becomes:

$$\arg\min_{R \in \mathrm{SO}(3)} \sum_i \| R\tilde{P}_i - \tilde{P}'_i \|^2$$

# Solving Procrustes

$$\arg\min_{R \in SO(3)} \sum_i \|R\tilde{P}_i - \tilde{P}'_i\|^2$$

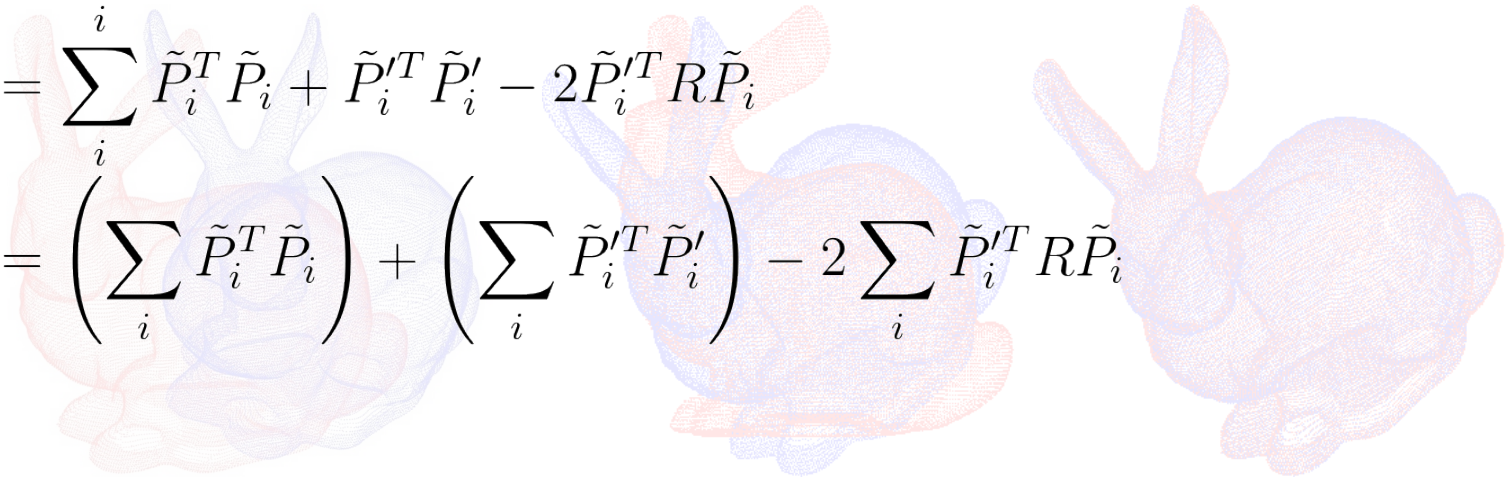
And once again we expand:

$$\sum_i \|R\tilde{P}_i - \tilde{P}'_i\|^2$$

$$= \sum_i (R\tilde{P}_i - \tilde{P}'_i)^T (R\tilde{P}_i - \tilde{P}'_i)$$

$$= \sum_i \tilde{P}_i^T \tilde{P}_i + \tilde{P}_i'^T \tilde{P}'_i - 2\tilde{P}_i'^T R\tilde{P}_i$$

$$= \left(\sum_i \tilde{P}_i^T \tilde{P}_i\right) + \left(\sum_i \tilde{P}_i'^T \tilde{P}'_i\right) - 2\sum_i \tilde{P}_i'^T R\tilde{P}_i$$



# Solving Procrustes

$$\arg\min_{R \in SO(3)} \sum_i \|R\tilde{P}_i - \tilde{P}'_i\|^2$$

And once again we expand:

$$\sum_i \|R\tilde{P}_i - \tilde{P}'_i\|^2$$

$$= \sum_i (R\tilde{P}_i - \tilde{P}'_i)^T (R\tilde{P}_i - \tilde{P}'_i)$$

$$= \sum_i \tilde{P}_i^T \tilde{P}_i + \tilde{P}_i'^T \tilde{P}'_i - 2\tilde{P}_i'^T R\tilde{P}_i$$

$$= \left(\sum_i \tilde{P}_i^T \tilde{P}_i\right) + \left(\sum_i \tilde{P}_i'^T \tilde{P}'_i\right) - 2\sum_i \tilde{P}_i'^T R\tilde{P}_i$$

These terms are constant given our input

So we just need to minimize this term – which is negative – so this is now a maximization problem

# Solving Procrustes

So now we have a modified objective:

$$\arg\max_{R\in \mathrm{SO}(3)} \sum_i \tilde{P}_i'^T R \tilde{P}_i = \mathrm{tr}\left(R\left(\sum_i \tilde{P}_i \tilde{P}_i'^T\right)\right) = \mathrm{tr}\left(RZ\right)$$

And as is a theme in this part of the class, we use the SVD to find the optimal solution

$$\begin{aligned} & \mathrm{tr}(RZ) \\ &= \mathrm{tr}(RUSV^T) \\ &= \mathrm{tr}((V^T RU)S) \\ &= \mathrm{tr}(QS) = \sum_k Q_{kk} s_k \leq \sum_k s_k \end{aligned}$$

So the best we can do is making $Q$ the identiy

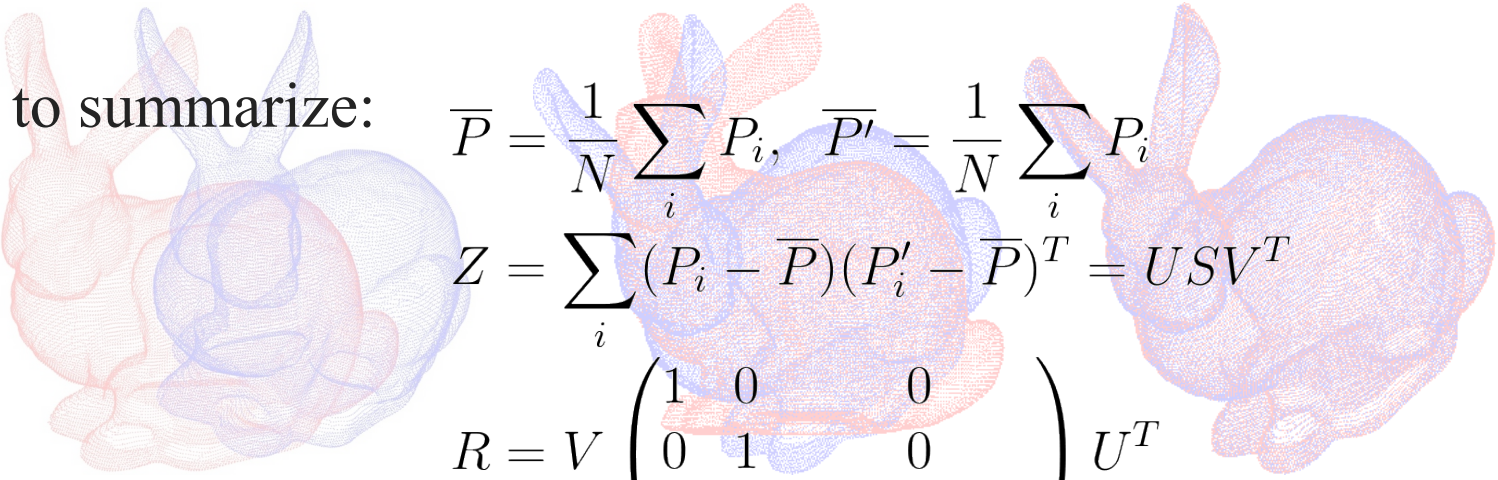
# Solving Procrustes

Can we make $Q$ the identity? Yes!

$$Q = V^T R U = I \implies R = V U^T$$

Again we use this trick to enforce that it a rotation and not a reflection:

$$R = V \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & \det(VU^T) \end{pmatrix} U^T$$



So to summarize:

$$\overline{P} = \frac{1}{N}\sum_i P_i, \quad \overline{P'} = \frac{1}{N}\sum_i P_i$$

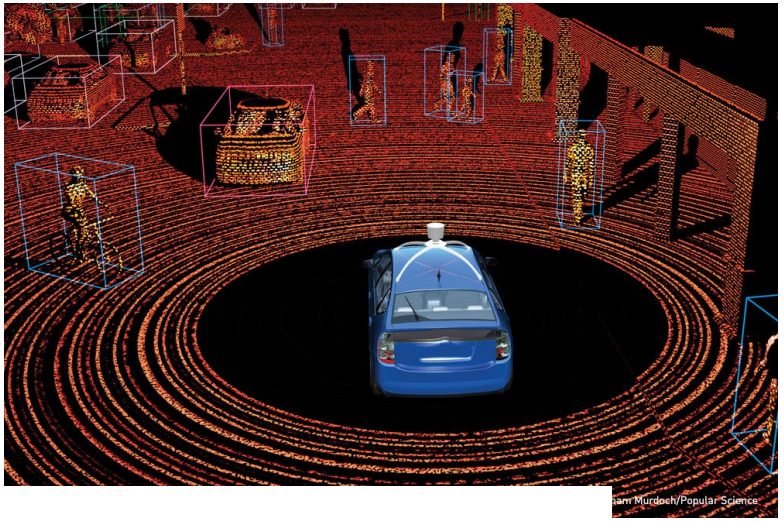
$$Z = \sum_i (P_i - \overline{P})(P_i' - \overline{P})^T = USV^T$$

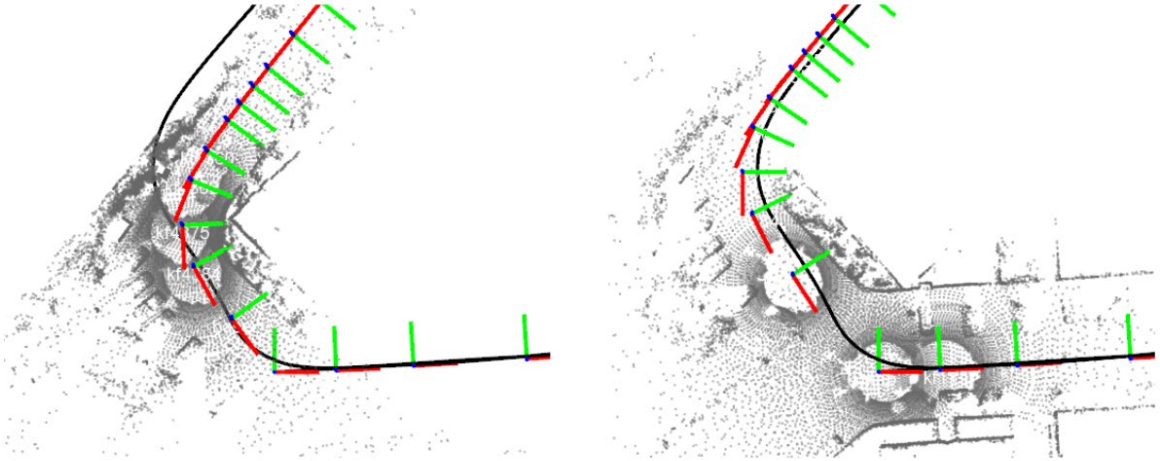
$$R = V \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & \det(VU^T) \end{pmatrix} U^T$$

$$T = \overline{P'} - R\overline{P}$$

# Procrustes in other contexts

Trying to find a robots pose Lidar measurements also needs 3D to 3D point matching.

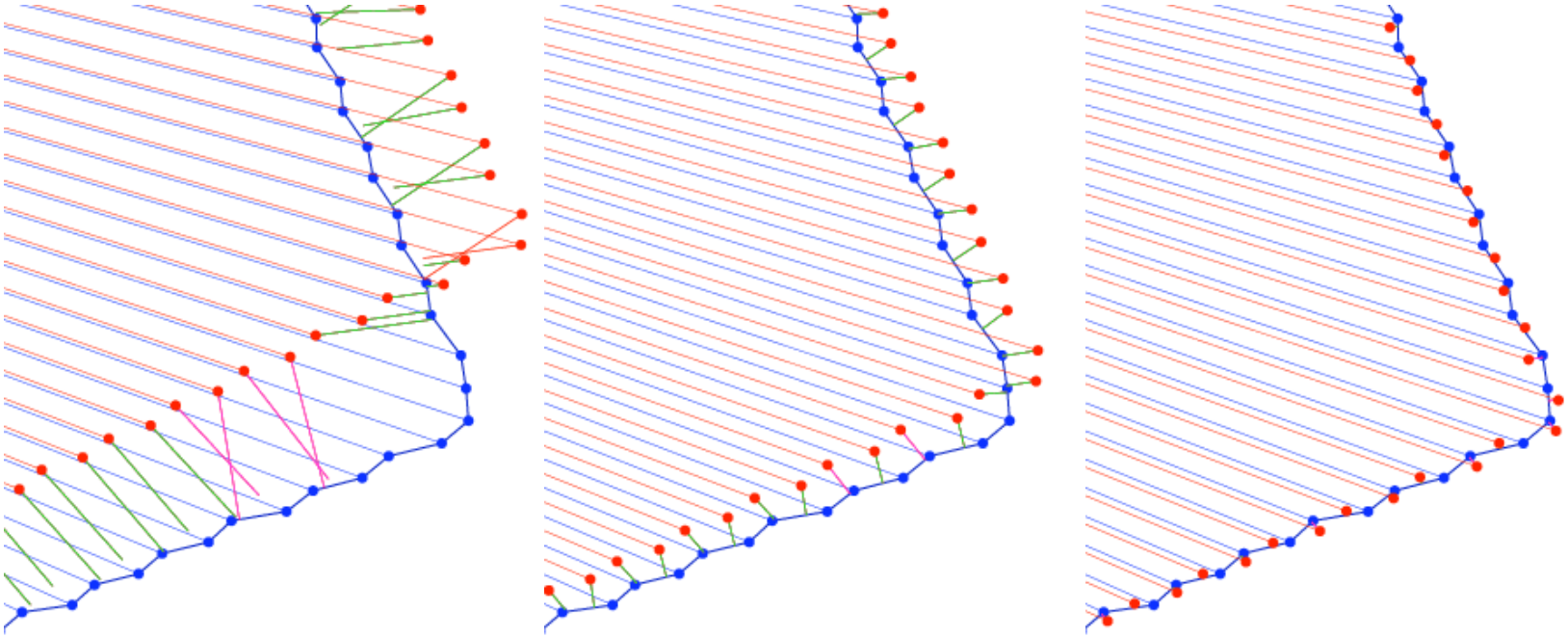




(a) before (b) after

# Procrustes in other contexts

However in this case we do not necessarily have point correspondences – solving both is known as Iterative Closest Point, which is a more common problem than Procrustes



# What if we have no 3D information?

We've thus far relied on our knowledge of the 3D world to extract the camera's pose – but what if we don't have that information?

# Epipolar Geometry

To solve this problem we use Epipolar Geometry – from this we can derive constraints between two points that correspond just from their 2D location

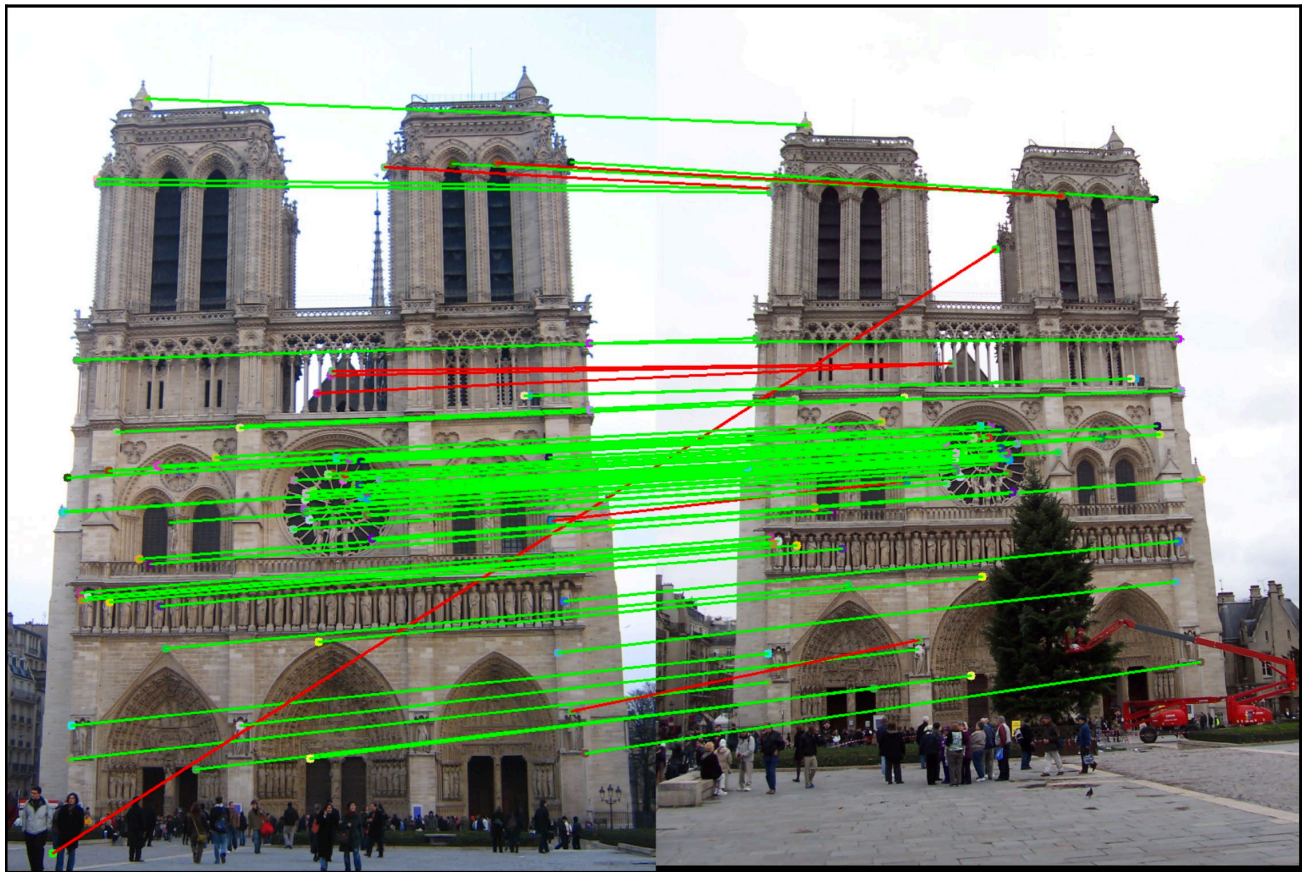


Image from https://www.cc.gatech.edu/~hays/compvision/proj2/

# Epipolar Geometry Setup

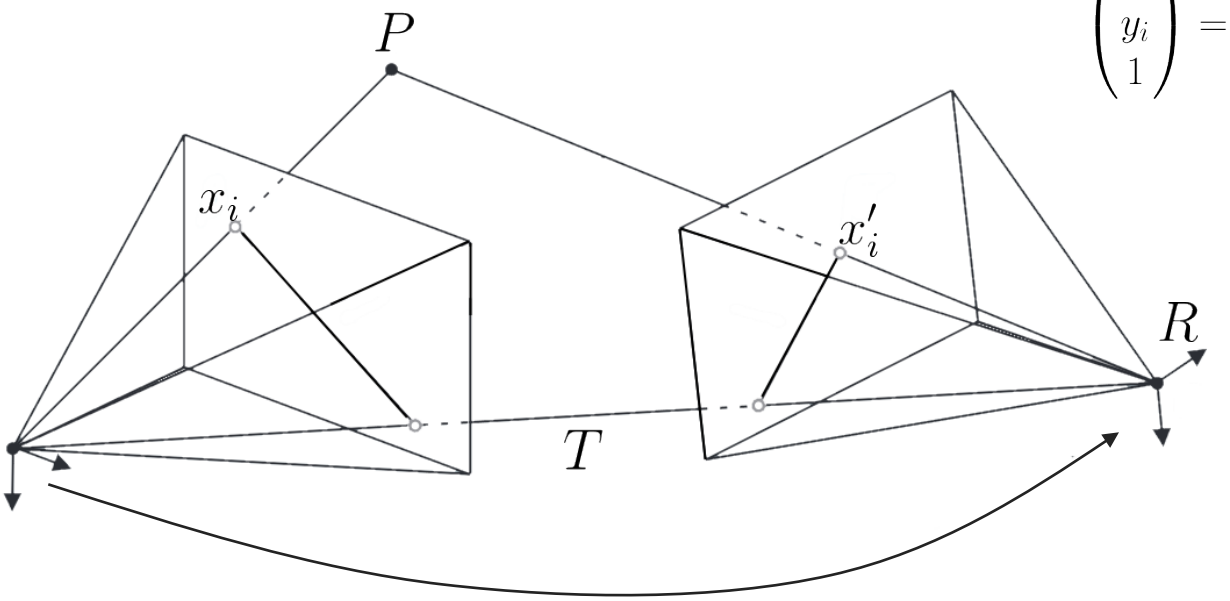
We have 2D correspondences $x_i \leftrightarrow x_i'$ from 2 images. For now we will assume they are calibrated. We want to find the rotation and translation between these images

$$x_i \leftrightarrow x_i' \implies R, T$$

Or: $$\lambda_i x_i = R(\lambda_i' x_i') + T$$

Also assumed that 2D coordinates are calibrated, i.e. for pixel coordinates $u_i, v_i$ :

$$\begin{pmatrix} x_i \\ y_i \\ 1 \end{pmatrix} = K^{-1} \begin{pmatrix} u_i \\ v_i \\ 1 \end{pmatrix}$$



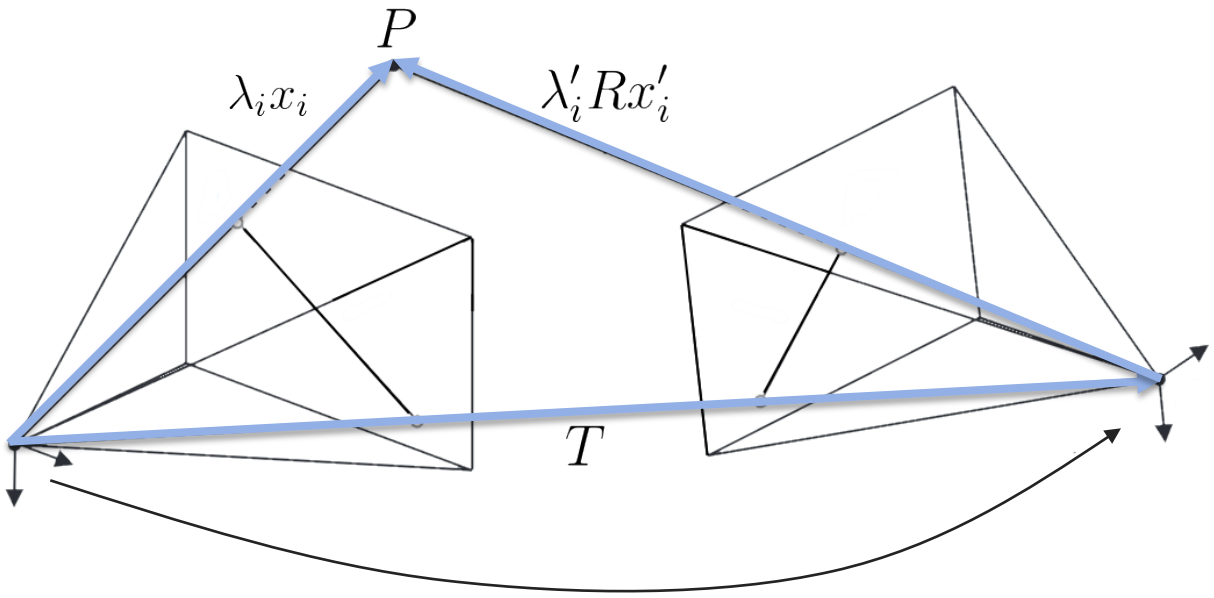
# Epipolar Constraint: Geometric Proof

Note that the vectors $\lambda_i x_i$ and $\lambda_i' R x_i'$ and $T$ all are coplanar – this is a non-trivial set of constraints. We can express it mathematically as the triple product

$$\lambda_i x_i^T (T \times \lambda_i' R x_i') = 0$$
$$\Longrightarrow \quad x_i^T [T]_\times R x_i' = 0$$

$$[T]_\times = \begin{pmatrix} 0 & -T_3 & T_2 \\ T_3 & 0 & -T_1 \\ -T_2 & T_1 & 0 \end{pmatrix}$$

This matrix encodes the cross product of $T$ with another vector



# Epipolar Constraint: Algebraic Proof

We can also derive this algebraically

$$
\begin{aligned}
& \lambda_i x_i = R(\lambda'_i x'_i) + T \\
\Longrightarrow\ & T \times (\lambda_i x_i) = R(\lambda'_i x'_i) + T \times T \\
\Longrightarrow\ & x_i^T (\lambda_i T \times x_i) = x_i^T (\lambda'_i T \times R x'_i) \\
\Longrightarrow\ & 0 = x_i^T T \times R x'_i = x_i^T [T]_\times R x'_i
\end{aligned}
$$

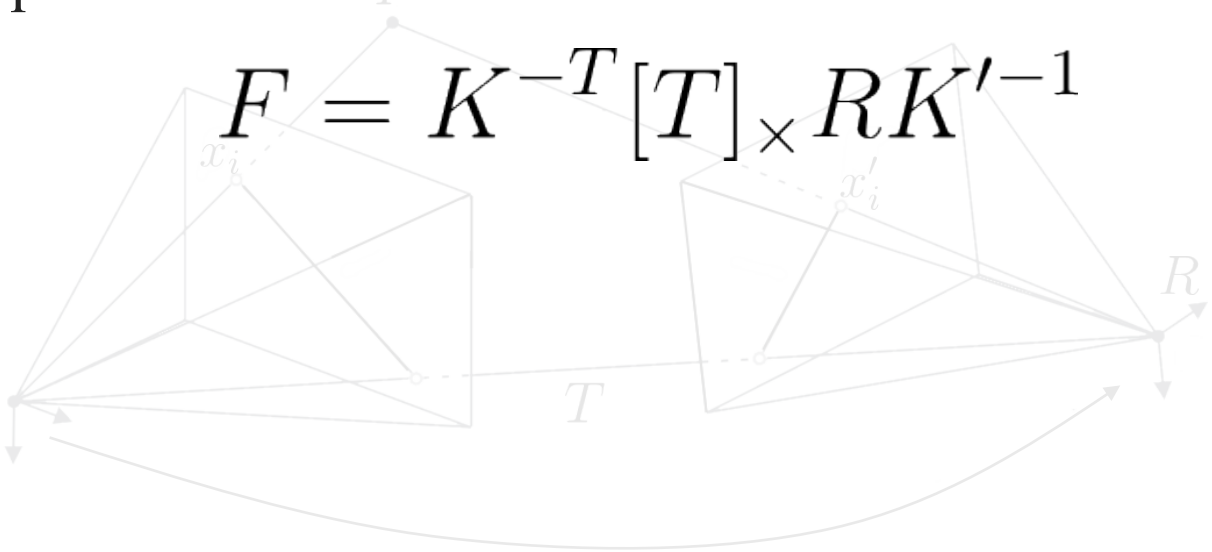
# Essential and Fundamental Matrix

The matrix we can get out of this is called the Essential Matrix:

$$E = [T]_\times R$$

This whole time we were assuming we had calibration. If we don't, then we have to include the inverse calibration matrices in the equation. This is known as the Fundamental Matrix:

$$F = K^{-T}[T]_\times R K'^{-1}$$



# Pose from Essential Matrix

How do we get the rotation and translation from the Essential matrix?

$$E = [T]_\times R$$

This should come as absolutely no surprise to you, but you need to use the Singular Value Decomposition. To show this, first note that we can express $[T]_\times$ as (assumed without proof)

$$[T]_\times = Q \begin{pmatrix} 0 & -\|T\| & 0 \\ \|T\| & 0 & 0 \\ 0 & 0 & 0 \end{pmatrix} Q^T$$

And:

$$\begin{pmatrix} 0 & -\|T\| & 0 \\ \|T\| & 0 & 0 \\ 0 & 0 & 0 \end{pmatrix} = \begin{pmatrix} \|T\| & 0 & 0 \\ 0 & \|T\| & 0 \\ 0 & 0 & 0 \end{pmatrix} \begin{pmatrix} 0 & -1 & 0 \\ 1 & 0 & 0 \\ 0 & 0 & 1 \end{pmatrix} = \begin{pmatrix} 0 & -1 & 0 \\ 1 & 0 & 0 \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} \|T\| & 0 & 0 \\ 0 & \|T\| & 0 \\ 0 & 0 & 0 \end{pmatrix}$$

# Pose from Essential Matrix

From this, we can see the decomposition of the Essential matrix:

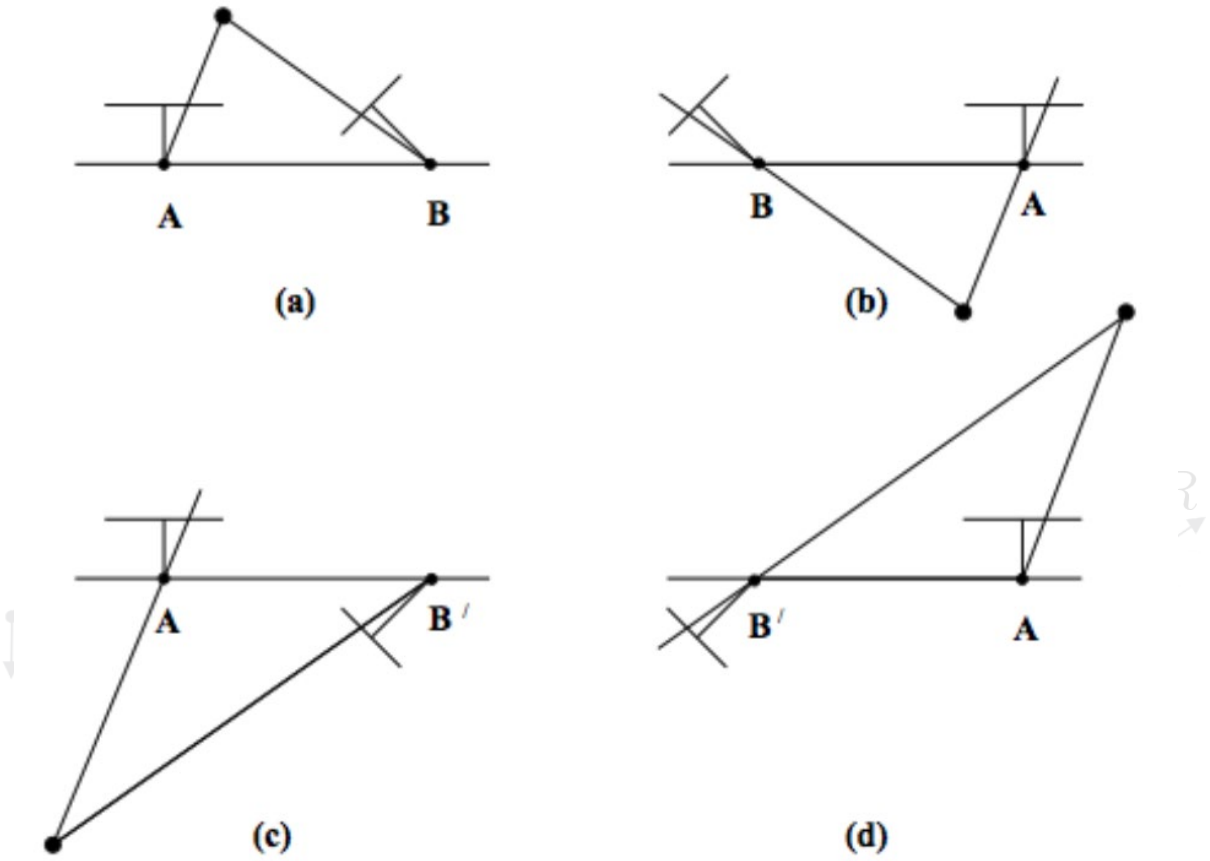
$$[T]_\times R = Q \underbrace{\begin{pmatrix} \|T\| & 0 & 0 \\ 0 & \|T\| & 0 \\ 0 & 0 & 0 \end{pmatrix}}_{S} \underbrace{\begin{pmatrix} 0 & -1 & 0 \\ 1 & 0 & 0 \\ 0 & 0 & 1 \end{pmatrix}}_{V^T} Q^T R$$

Or:

$$= Q \underbrace{\begin{pmatrix} 0 & 1 & 0 \\ -1 & 0 & 0 \\ 0 & 0 & 1 \end{pmatrix}}_{U} \underbrace{\begin{pmatrix} \|T\| & 0 & 0 \\ 0 & \|T\| & 0 \\ 0 & 0 & 1 \end{pmatrix}}_{S} \underbrace{Q^T R}_{V^T}$$

# Pose from Essential Matrix

We can extract $T$ and $R$ from this but the solution is ambiguous– but only one of them corresponds to a real solution (similar to P3P solution)



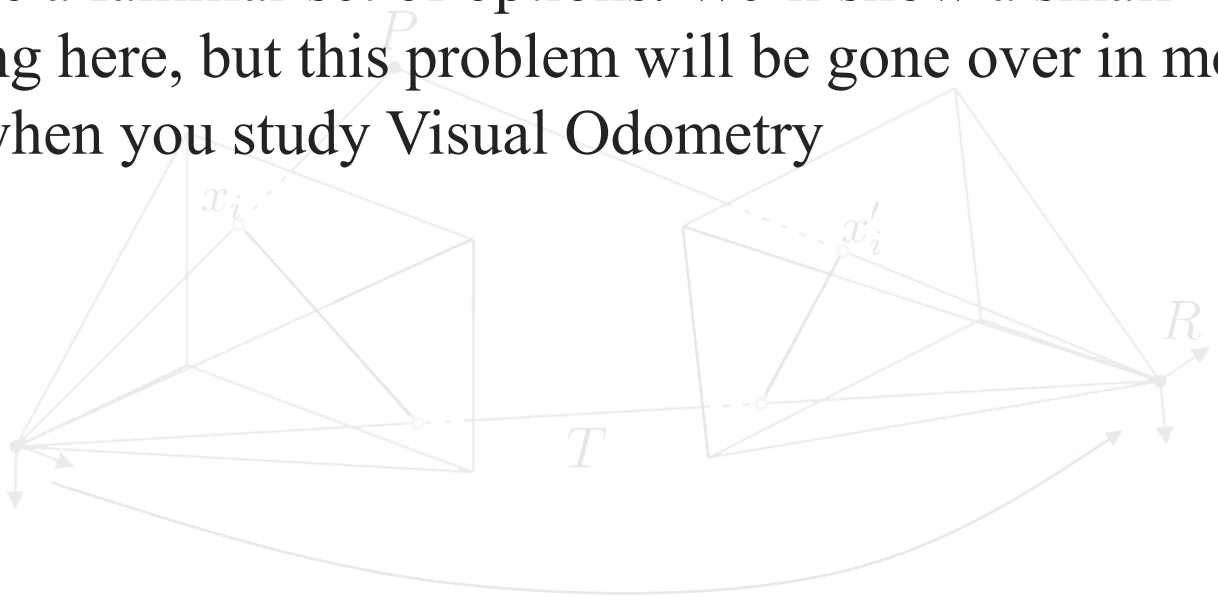

https://web.stanford.edu/class/cs231a/course_notes/04-stereo-systems.pdf

# Estimating Essential Matrix

So the question is how do we find *E*? We need to optimize over the set of Essential matrices (5 degrees of freedom – Why?)

$$\arg\min_{T\in S^2, R\in SO(3)} \sum_i (x_i^T [T]_\times R x_i')^2$$

We have a familiar set of options. We'll show a small sampling here, but this problem will be gone over in more detail when you study Visual Odometry



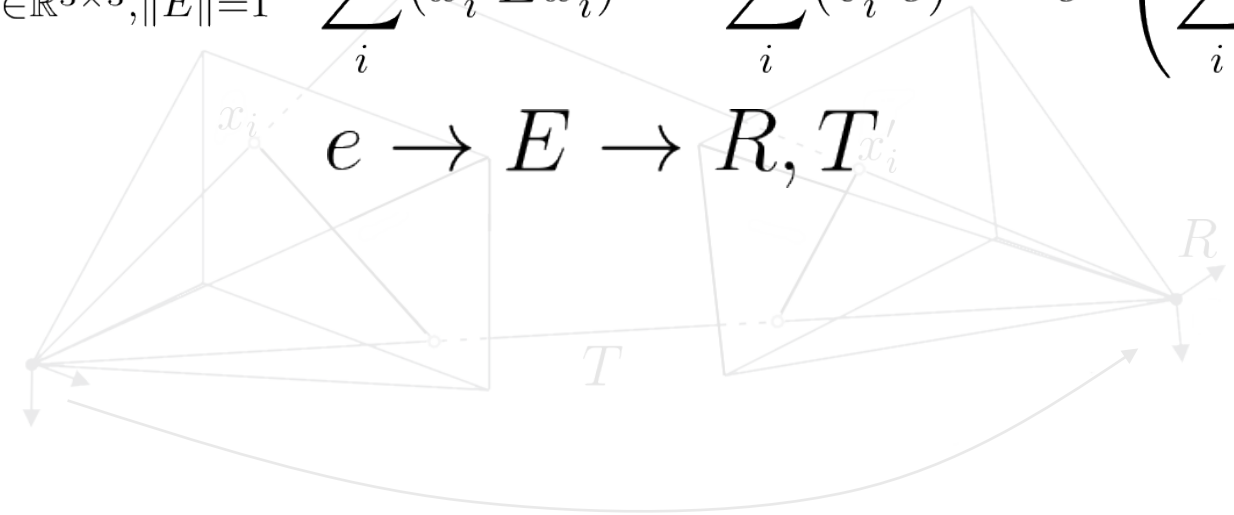
# Estimating Essential Matrix

Option #1: A linear relaxation to find a solution then enforcing the output to be a proper Essential Matrix.

This is called 8-point algorithm as the relaxation requires needs a minimum 8 point correspondences

$$\arg\min_{E\in\mathbb{R}^{3\times 3},\|E\|=1} \sum_i (x_i^T E x_i')^2 = \sum_i (v_i^T e)^2 = e^T \left(\sum_i (v_i v_i^T)\right) e$$

$$e \rightarrow E \rightarrow R, T$$

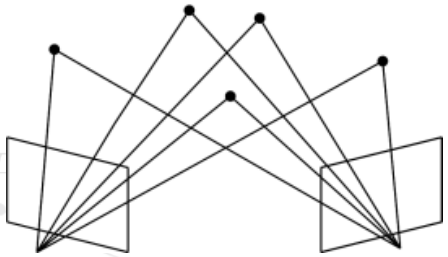


# Estimating Essential Matrix

Option #2: An exact solution using 5 point correspondences

If you thought the solution to P3P was complicated, this one is much more so.

http://cmp.felk.cvut.cz/~kukelova/minimal/5_pt_relative.php

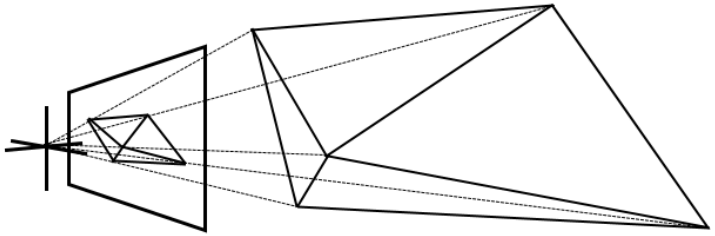


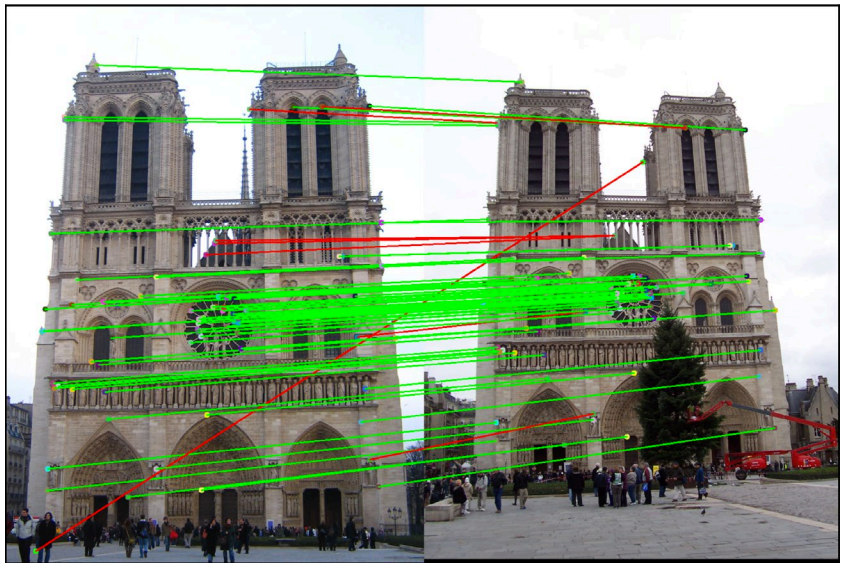
Option #3: There are intermediate solutions algorithm which allows you to use less than 8 points but don't need to solve for the full 5-point problem, such as the 7-point algorithm

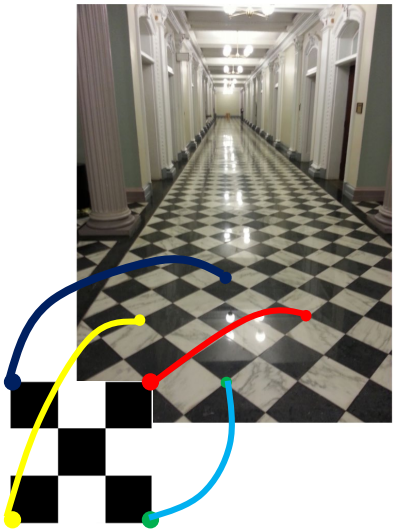
$$\det(E_1 + \lambda E_2) = a_3\lambda^3 + a_2\lambda^2 + a_1\lambda + a_0 = 0$$

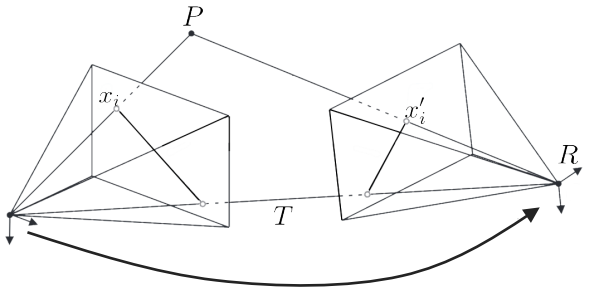
# Reflection: Point Correspondences

In all the algorithms we have covered thus far we have simply assumed correspondences were given – but how do we compute them?



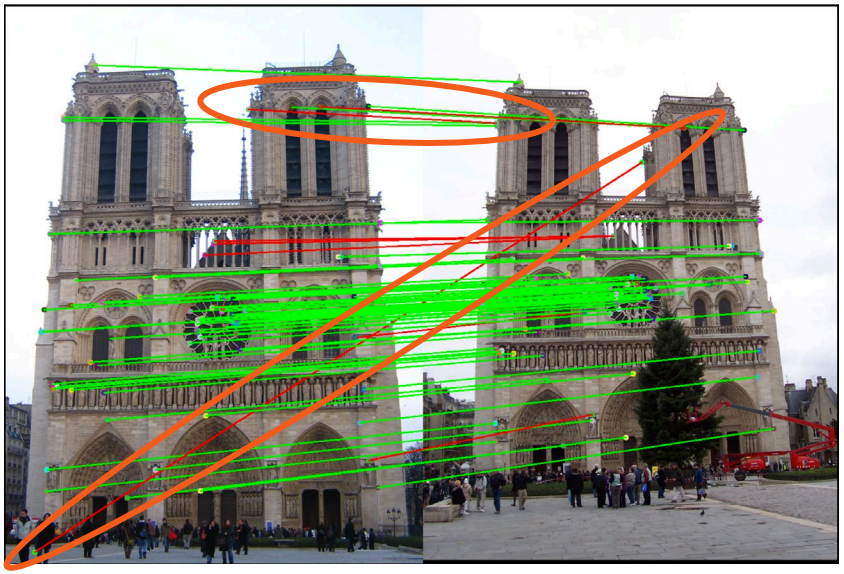


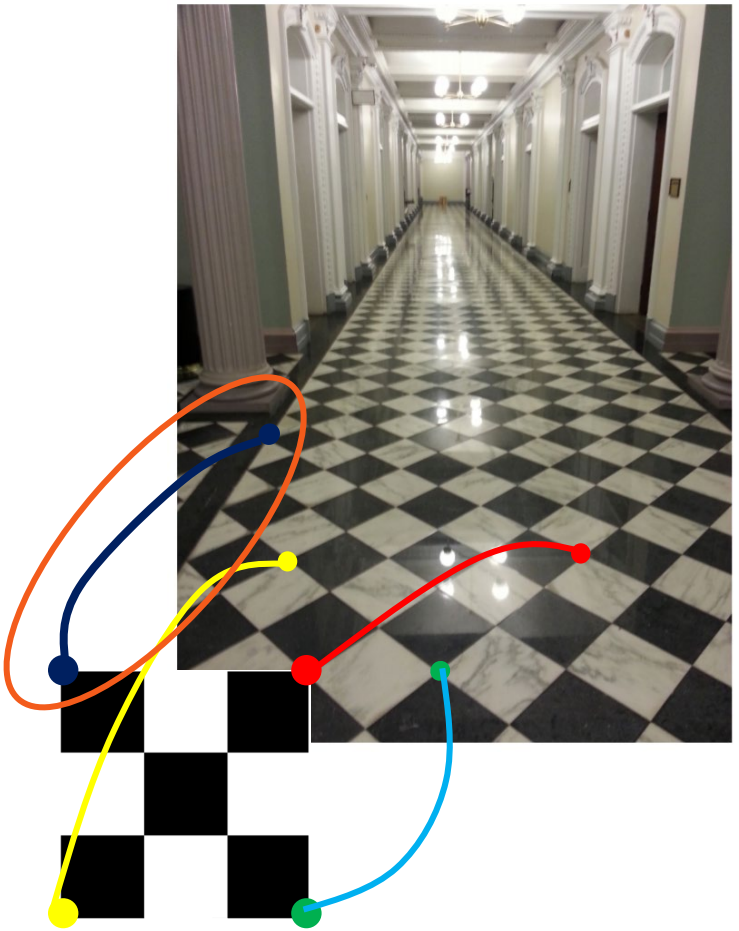




# Reflection: Outliers

And what do we do if on of our matches ends up being extremely wrong, i.e. not just small noise?





# Next Time

How do we find correspondences?

- Our solution: Optical Flow

And once we do, how do we make sure they are not completely wrong?

- Our solution: RANSAC

# Final Notes on the Project

- Note that there will be 2 SVDs you have to compute, one for estimating the projective transformation and one for extracting the rotation
- For estimating the projective transformation, it is better if you calibrate the 2D positions i.e. don't use pixel coordinates