3D Computer Vision

Visual Motion Estimation

Ivan Marković

University of Zagreb Faculty of Electrical Engineering and Computing
Department of Control and Computer Engineering
Laboratory for Autonomous Systems and Mobile Robotics (lamor.fer.hr)

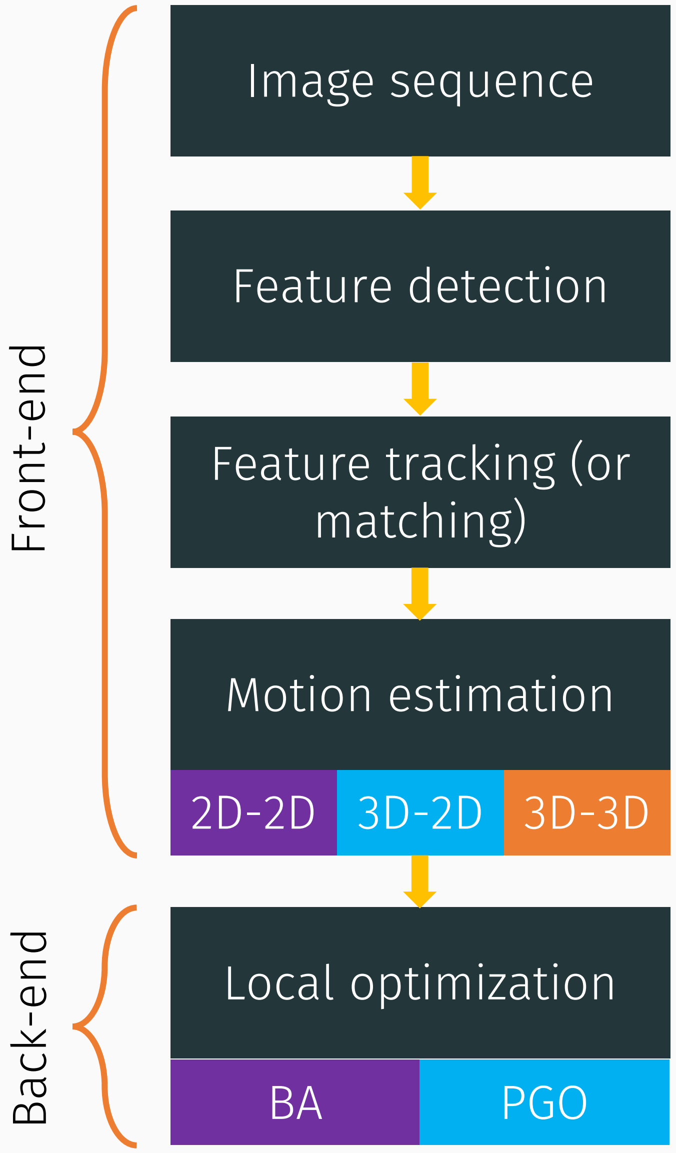
Outline

- Problem formulation
- 3D-3D motion estimation
- 3D-2D motion estimation
  - Perspective from *n* points (PnP)
  - Stereo and mono odometry using PnP
- 2D-2D motion estimation
  - Estimating the essential matrix
  - Relative scale
- Keyframe selection

Outline

- Problem formulation
- 3D-3D motion estimation
- 3D-2D motion estimation
  - Perspective from *n* points (PnP)
  - Stereo and mono odometry using PnP
- 2D-2D motion estimation
  - Estimating the essential matrix
  - Relative scale
- Keyframe selection

VO – front-end vs. back-end



The part of visual odometry in charge of detecting and tracking (or matching) features and estimating the relative motion is called the front-end.

The part of visual odometry in charge of refining motion estimation and ensuring local consistency over a window of past frames is called the back-end.

The motion estimation block usually involves a robust model estimation procedure such as RANSAC that is used to determine the set of inliers based on which the final motion parameters are computed.

VO assumptions

To estimate the motion between two views, we need to be able to detect and match features of one or several subsequent frames; thus, the following assumptions are made:

1. Adequate illumination of the scene – if the images are over– or undersaturated it will be nearly impossible to detect and track features.
2. Dominance of static objects over the moving objects – to estimate *ego-motion* we must use features on static parts of the scene; otherwise, we do not know if the scene is moving and the camera is static or vice-versa.
3. Enough texture to allow apparent motion to be estimated – moving in featurless environments is almost the same as moving in darkness
4. Sufficient scene overlap between subsequent frames – otherwise, we do not have any features to match

Problem formulation

A camera is moving through an environment and taking images at discrete time steps k. It can be handheld or rigidly attached to a platform (such as a mobile robot or a vehicle).

In the case of a monocular system, we denote the set of k images as

$$I_{0:k} = \{I_0, I_1, \ldots, I_k\}\,.$$

In the case of a stereo system, at each time step we have a left and a right image that we denote by

$$\begin{aligned} I_{l,0:k} &= \{I_{l,0}, I_{l,1}, \ldots, I_{l,k}\} \\ I_{r,0:k} &= \{I_{r,0}, I_{r,1}, \ldots, I_{r,k}\}\,. \end{aligned}$$

In the stereo case, we can use the coordinate system of the left camera as the origin frame.

Pose as a rigid body transformation

Two camera positions at subsequent time steps *k-1* and *k* are related by the rigid body transformation

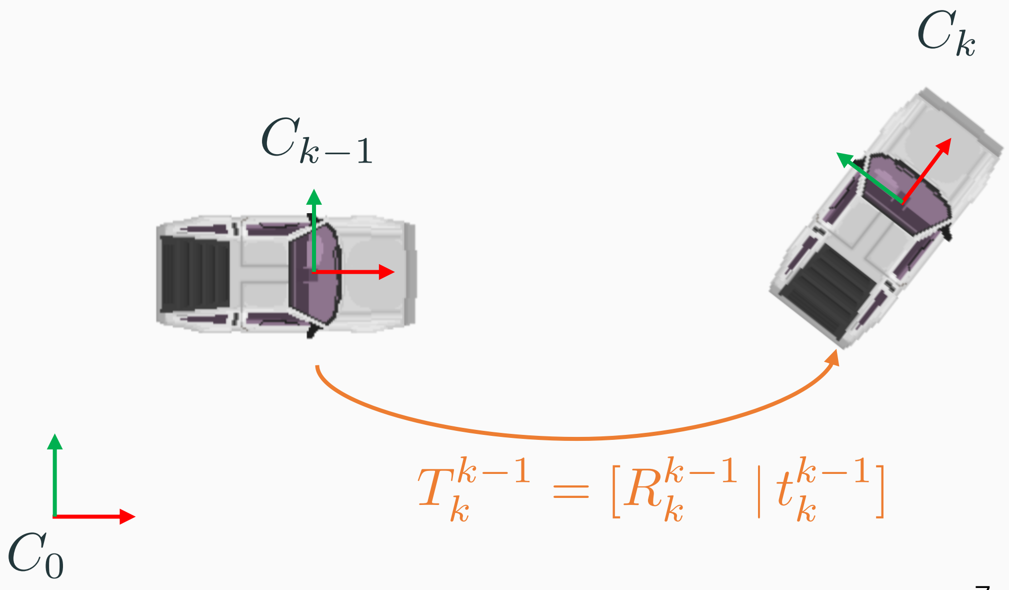
$$T_k^{k-1} = \begin{bmatrix} R_k^{k-1} & t_k^{k-1} \\ 0 & 1 \end{bmatrix}.$$

The set $T_{0:k} = \{T_1^0, T_2^1, \ldots, T_k^{k-1}\}$ contains all subsequent motions.

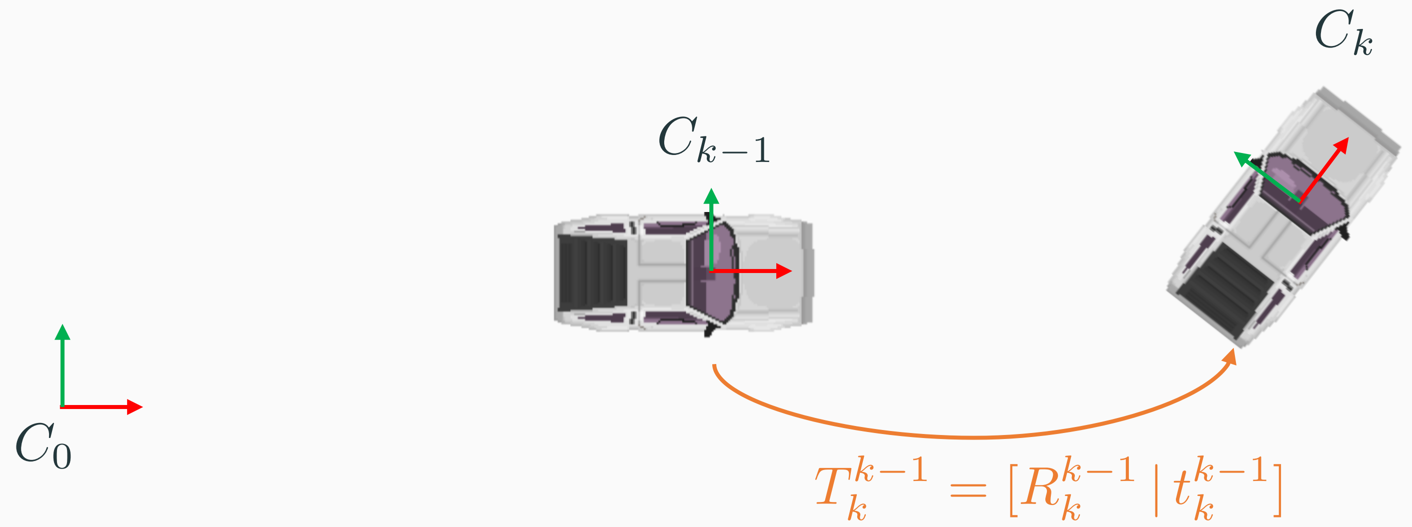
The set of camera poses

$$C_{0:k} = \{C_0, C_1, \ldots, C_k\}$$

contains the transformations of the camera coordinate frame with respect to the initial coordinate frame at *k=0*.



Concatenating rigid body transformations



The current pose of the camera C_k , or of the mobile robot or a vehicle, is computed by concatenating all the transformations from *0:k*

$$C_k = C_{k-1}T_k = C_{k-2}T_{k-1}T_k = \cdots = C_0T_1T_2\ldots T_{k-1}T_k.$$

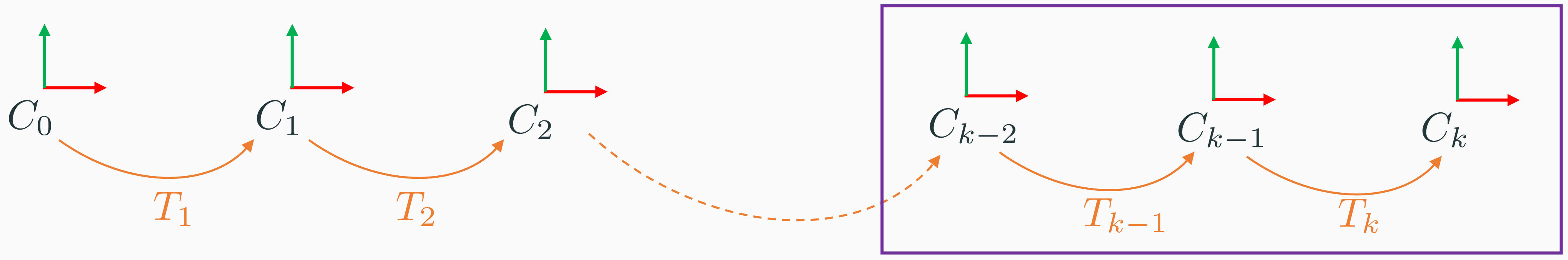
Note that for brevity we omit the previous frame index, i.e., $T_k \leftarrow T_k^{k-1}$.

The goal of VO

The main task of visual odometry is to compute the relative rigid body transformations T_k from images I_k and I_{k-1} and then to concatenate them in order to recover the full trajectory $C_{0:k}$ of the camera.

Evidently, visual odometry recovers the trajectory incrementally, pose after pose.

As discussed earlier, the optional step includes optimization over a local window of frames that aims to enforce local consistency and hopefully result with a more accurate estimate of the total trajectory.



Local optimization over *m* frames

Motion estimation approaches

Depending whether point feature correspondence p_k and p_{k-1} are specified in 2D or 3D we have three groups of motion estimation approaches:

1. 2D-2D: both p_k and p_{k-1} are specified in 2D image coordinates
2. 3D-2D: point features p_{k-1} are specified in 3D coordinates X_{k-1} and p_k are their corresponding 2D image reprojections on the image I_k .
3. 3D-3D: both p_k and p_{k-1} are specified in 3D coordinates X_k and X_{k-1}, respectively. For this approach, we have to triangulate 3D points and each time step, e.g., by using a stereo or depth camera.

In the following, we discuss each of these approaches and their minimal case solutions.

Outline

- Problem formulation
- 3D-3D motion estimation
- 3D-2D motion estimation
  - Perspective from n points (PnP)
  - Stereo and mono odometry using PnP
- 2D-2D motion estimation
  - Estimating the essential matrix
  - Relative scale
- Keyframe selection

3D-3D motion estimation

The 3D-3D approach estimates motion from 3D-to-3D point feature correspondences - X_k and X_{k-1} - it is also called the point cloud registration problem.

The minimal-case solution involves 3 non-collinear correspondences.

In general, the goal is to find the aligning transformation T_k that minimizes the following distance between the two 3D features sets

$$\arg\min_{T_k} \sum_i ||X_k^i - T_k X_{k-1}^i||^2,$$

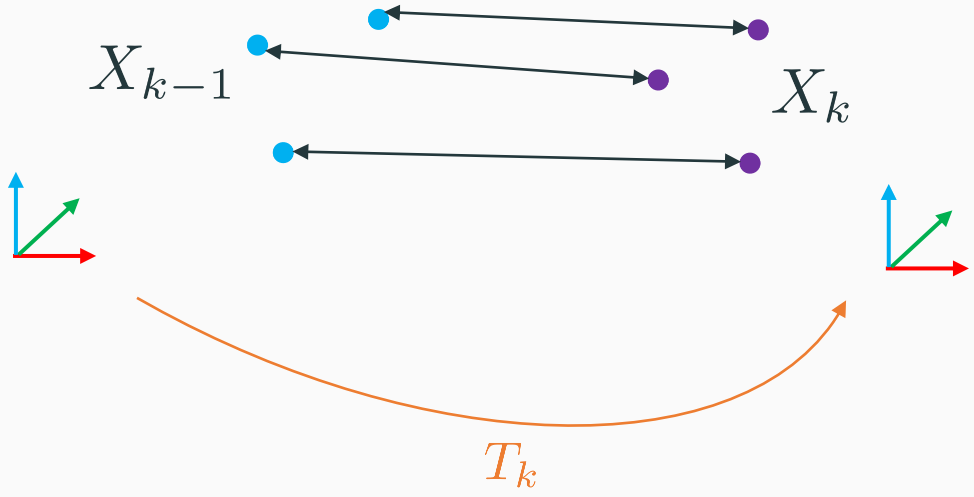
where the superscript *i* denotes the *i*-th point feature.

3D-3D motion estimation

A closed-form solution for registration of 3D-3D correspondences exists and is based on least square fitting[1].

Another class of solutions are the iterative closest points (ICP) algorithms that minimize point-to-point, point-to-line, or point-to-plane distances (sensitive to initial guess). An algorithm that joins multiple criteria in the minimization function is called the Generalized ICP algorithm[2].

Essentially, any point cloud registration approach discussed in earlier lectures can be used at this point, but keep in mind the final application – if its robotics or autonomous vehicles, then the algorithm should be executed in real-time, i.e., at least at the rate of 10 Hz.



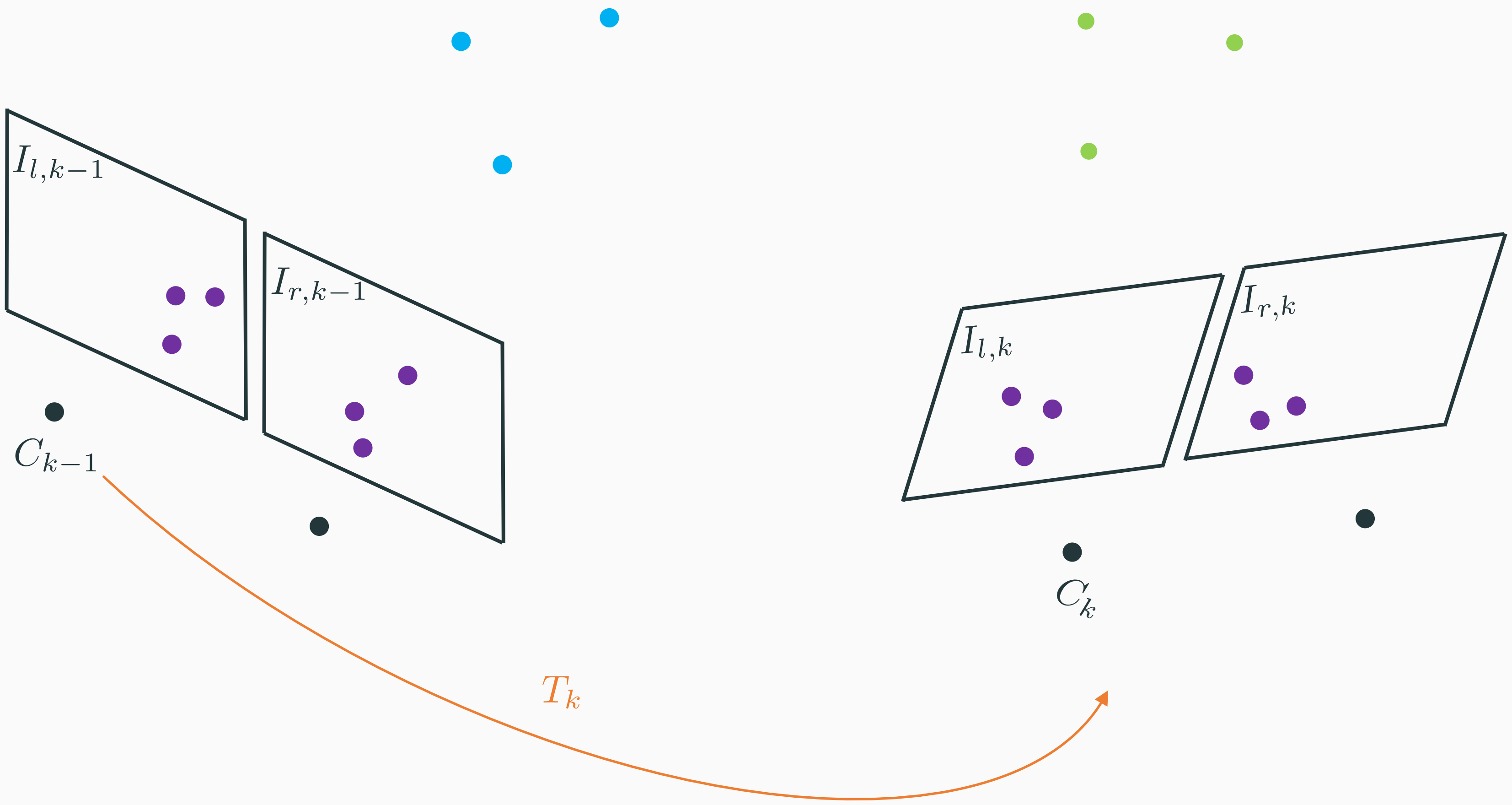
[1] A. S. Arun, T. S. Huang and S. D. Blostein (1987). „Least-Squares Fitting of Two 3-D Point Sets"

[2] A. Segal, D. Hehnel, S. Thrun (2005). „Generalized-ICP."

Aligning 3D point clouds = motion estimation

3D-3D

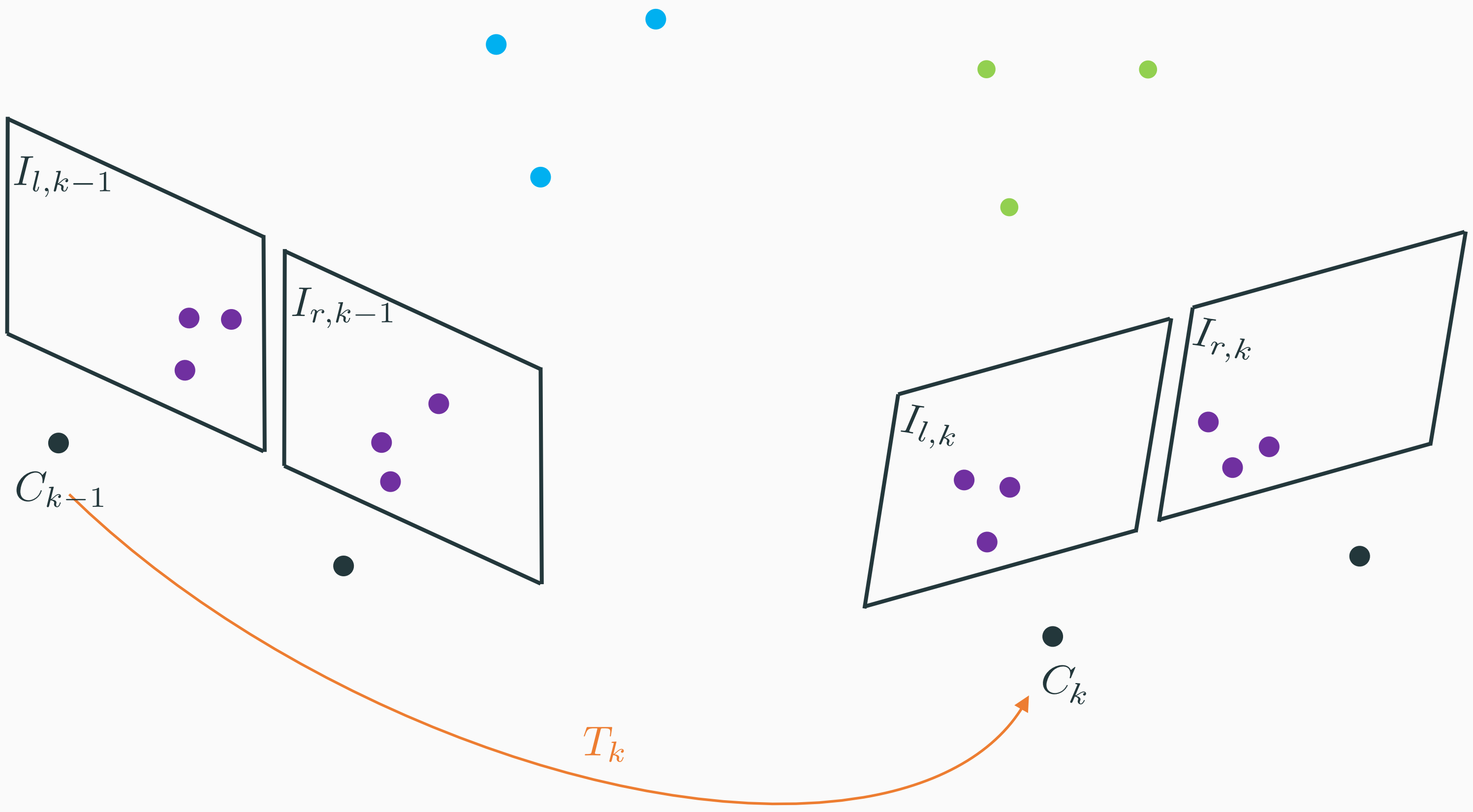
As stated earlier, to compute motion estimation from 3D-3D correspondences we triangulate the point features at steps *k-1* and *k* . Thereafter, we align the 3D points using any available registration method.



Aligning 3D point clouds = motion estimation

3D-3D

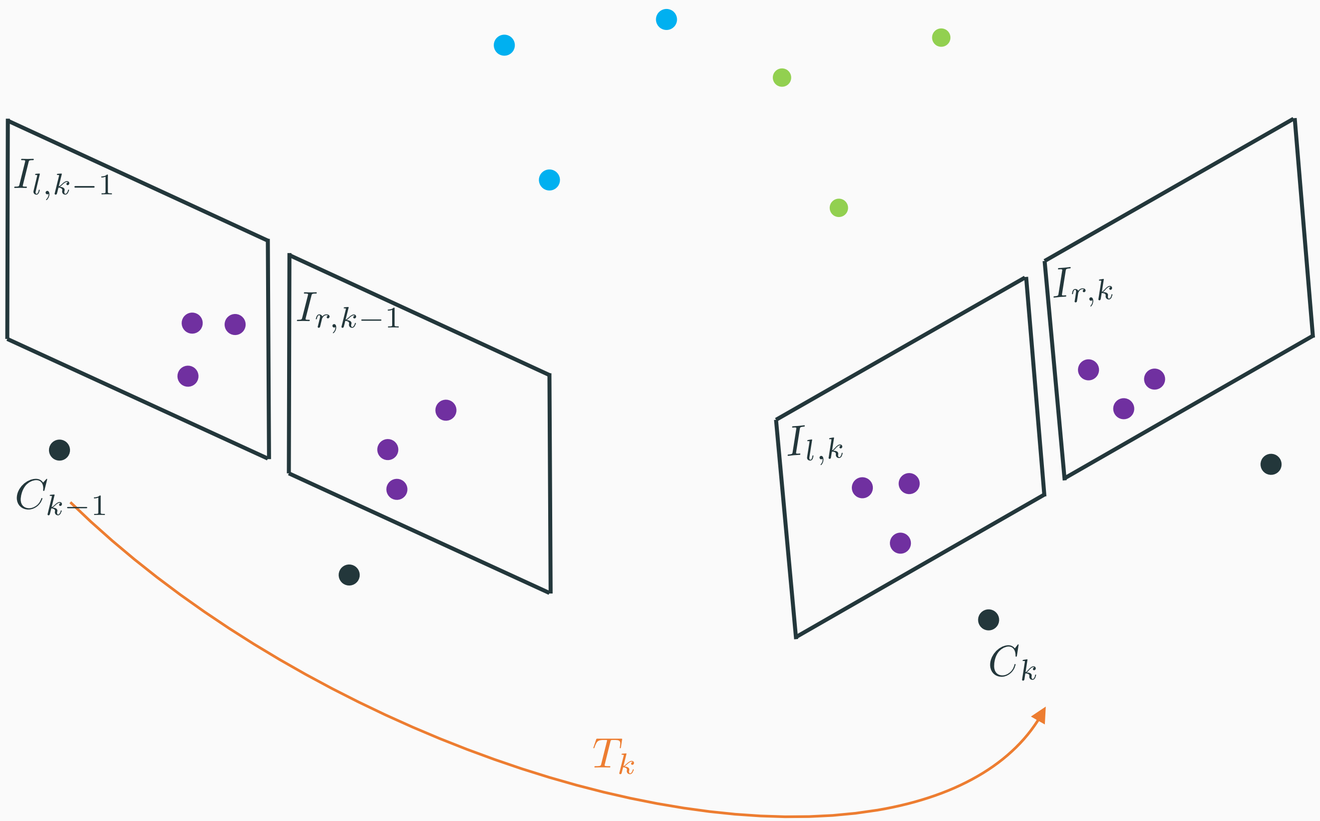
As stated earlier, to compute motion estimation from 3D-3D correspondences we triangulate the point features at steps *k-1* and *k* . Thereafter, we align the 3D points using any available registration method.



Aligning 3D point clouds = motion estimation

3D-3D

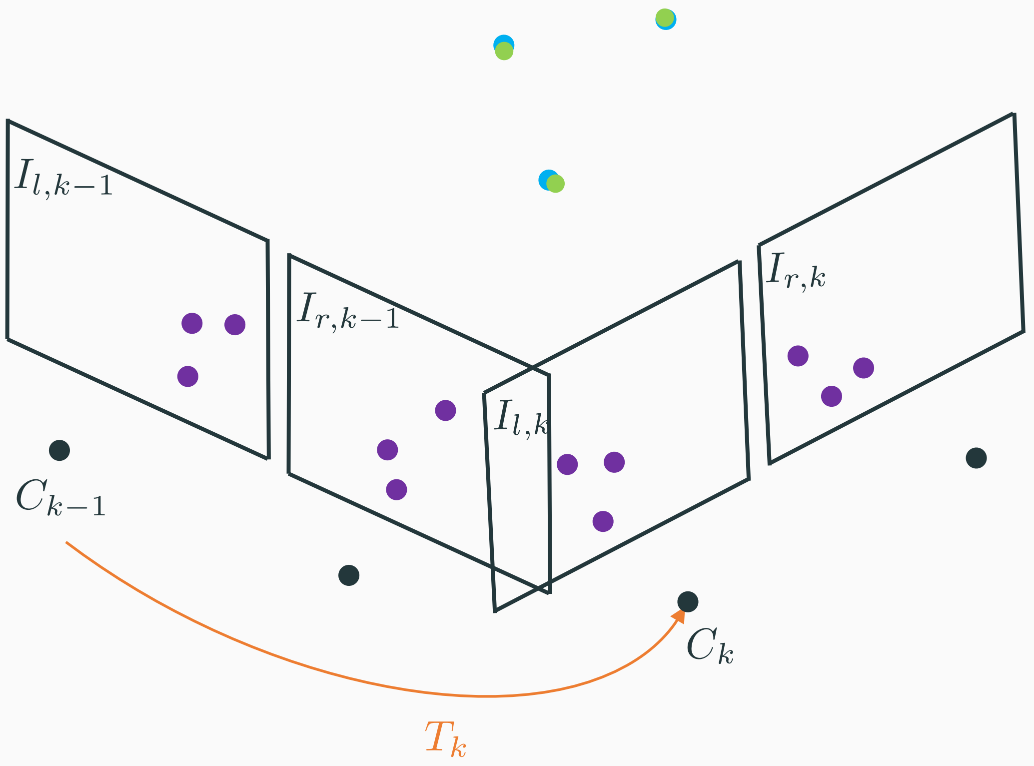
As stated earlier, to compute motion estimation from 3D-3D correspondences we triangulate the point features at steps *k-1* and *k* . Thereafter, we align the 3D points using any available registration method.



Aligning 3D point clouds = motion estimation

3D-3D

As stated earlier, to compute motion estimation from 3D-3D correspondences we triangulate the point features at steps *k-1* and *k* . Thereafter, we align the 3D points using any available registration method.



3D-3D VO pipeline

The pipeline of a 3D-3D odometry might have the following structure:

1. Capture two stereo image pairs $I_{l,k-1}, I_{r,k-1}$ and $I_{l,k}, I_{r,k}$.
2. Detect and match features between $I_{l,k}$ and $I_{l,k-1}$.
3. Match and triangulate matched features for each stereo pair
4. Compute the relative transformation T_k from triangulated 3D point feature sets X_{k-1} and X_k.
5. Concatenate camera transformation by computing $C_k = C_{k-1}T_k$.

Outline

- Problem formulation
- 3D-3D motion estimation
- 3D-2D motion estimation
  - Perspective from n points (PnP)
  - Stereo and mono odometry using PnP
- 2D-2D motion estimation
  - Estimating the essential matrix
  - Relative scale
- Keyframe selection

3D-2D motion estimation

The 3D-2D approach estimates motion from 3D-to-2D point feature correspondences - X_{k-1} and p_k - the reprojections of the 3D point features to the image I_k - it is also called the perspective from *n* points (PnP).

The minimal-case solution involves 3 non-collinear points (+1 for disambiguation).

In general, the goal is to find the transformation T_k that minimizes the image reprojection error

$$\arg\min_{T_k} \sum_i ||p_k^i - \pi_k(T_k X_{k-1}^i)||^2,$$

where the superscript *i* denotes the *i*-th point feature and $\pi_k(.)$ is the projection function that projects a 3D point to image frame I_k.

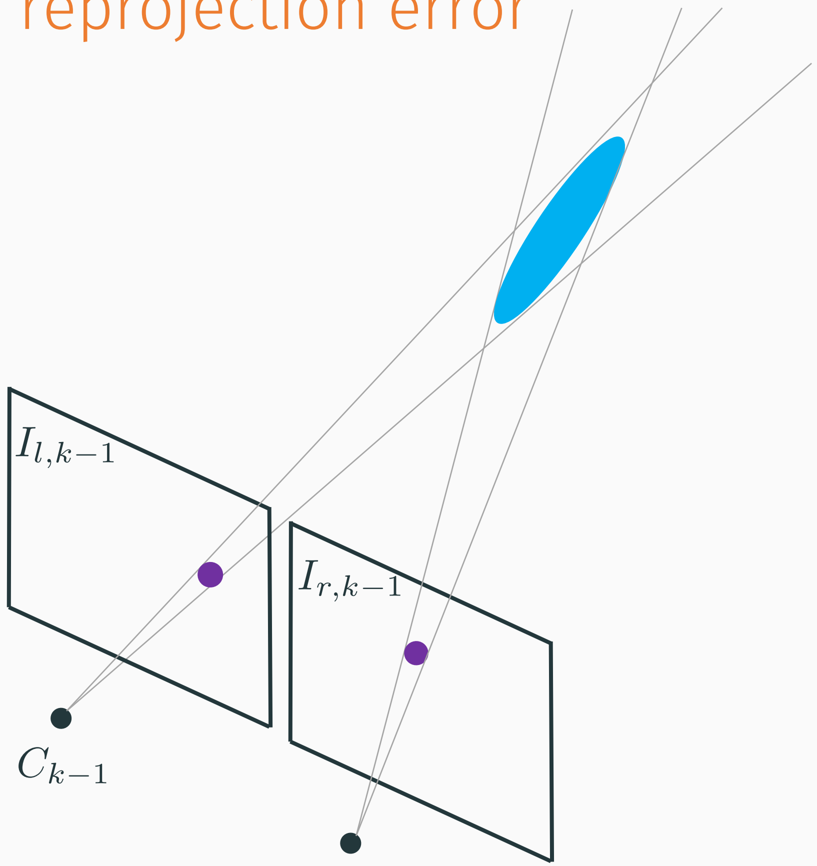
Feature depth uncertainty

3D-2D

The 3D-2D approach has been found to yield more accurate results than 3D-3D[3] correspondences because it minimizes the image reprojection error instead of the 3D-3D feature position error.

The reason behind this lies in the fact that the uncertainty of feature 3D positions is highly anisotropic and the depth uncertainty increases the larger the ratio of object's distance with respect to cameras baseline.

The uncertainty of reprojections, on the other hand, is highly isotropic and it is less error sensitive to minimize the image reprojection error than alignment of triangulated 3D-3D point feature sets[4].



[3] Nister, D., Naroditsky, O., and Bergen, J. (2004). „Visual odometry."
[4] Badino, H., Yamamoto, A., and Kanade, T. (2013). „Visual odometry by multi-frame feature integration."

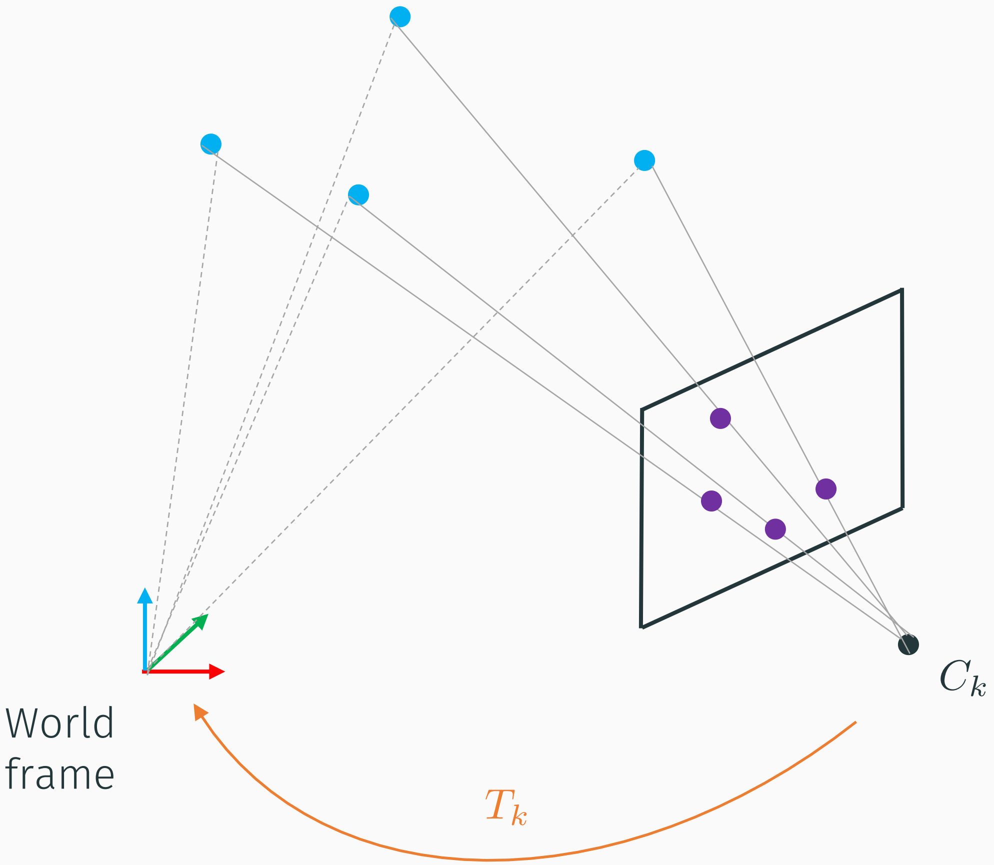
PnP problem

3D-2D

Perspective from *n* points (PnP) is actually a camera localization problem, where we aim to estimate the 6DoF pose of the camera with respect to the world frame from a set of 3D-2D point feature correspondences.

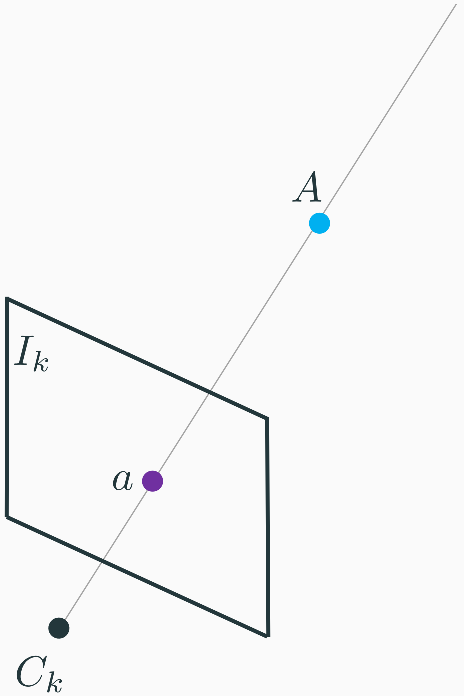
Minimal-case solution:

1. 1 point – infinite solutions
2. 2 points – infinitely many solution but on a circular arc
3. 3 points (non-collinear) – up to 4 possible solutions
4. 4 points – unique solution.

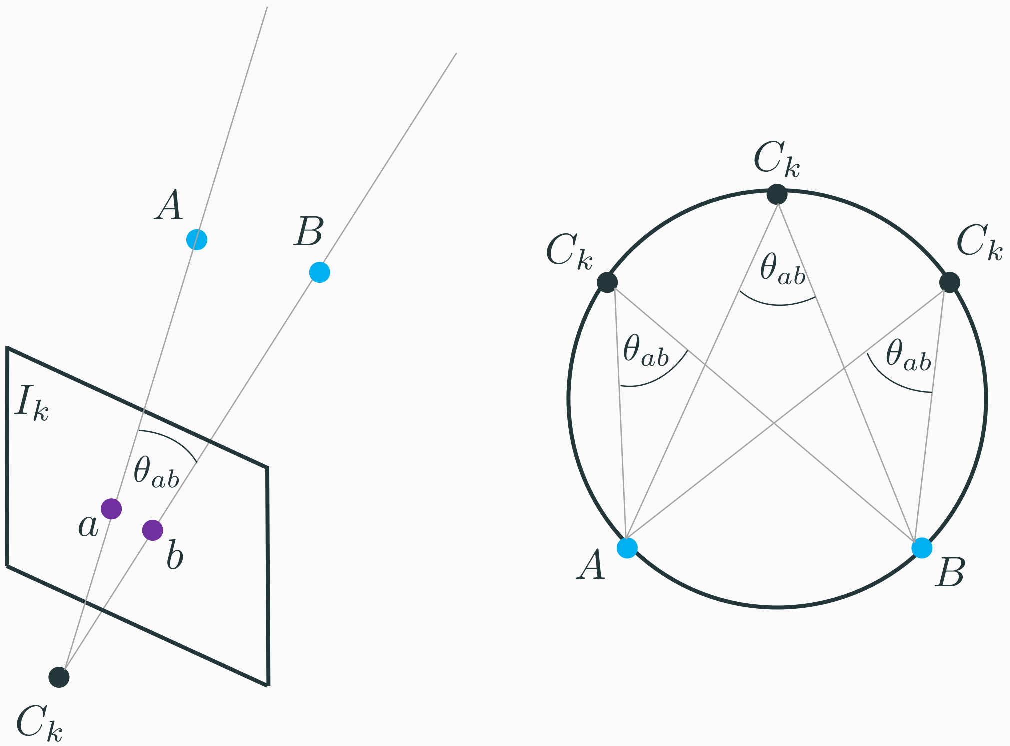


1 point and 2 points analysis

1 point – infinetly many solutions



2 points – bounded on a circular arc



P3P method

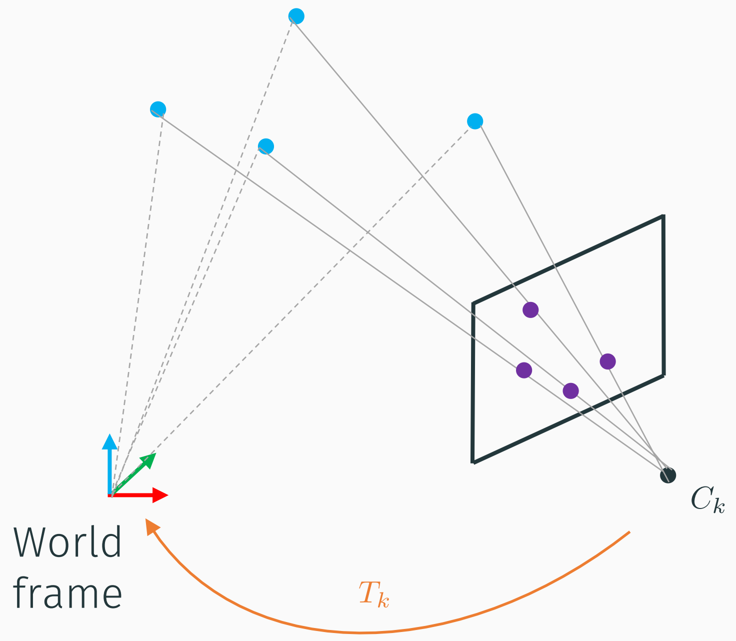
3D-2D

P3P problem was first introduced and solved in 1841. It was discovered that this problem typically does not lead to a unique mathematical solution, but rather may yield up to four distinct solutions, any of which could be the pose of the camera[5].

There are multiple methods for solving the P3P analytically and most are a two-step process:

1. Determine the length of projection rays
2. Estimate the camera pose.

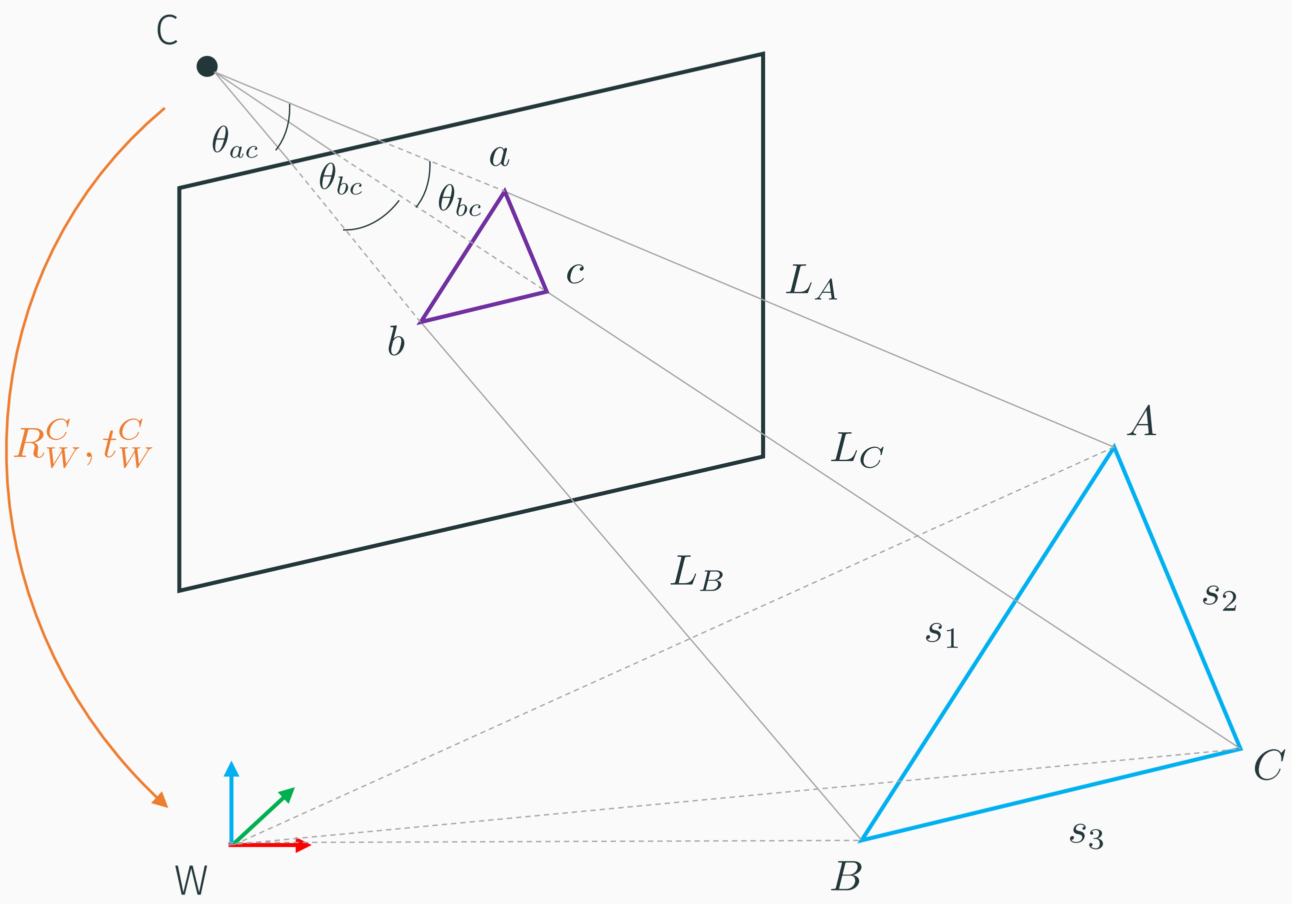
We will explore the approach that systematically analyzes all the possible solutions and takes into account constraints that correspond to geometrically nondegenerate solutions[6].



[5] J. A. Grunert (1841). „Das Pothenotische Problem in erweiterter Gestalt nebst Über seine Anwendungen in der Geodäsie."
[6] Gao, Hou, Tang, Cheng (2003). „Complete Solution Classification for the Perspective-Three-Point Problem."

P3P method

3D-2D



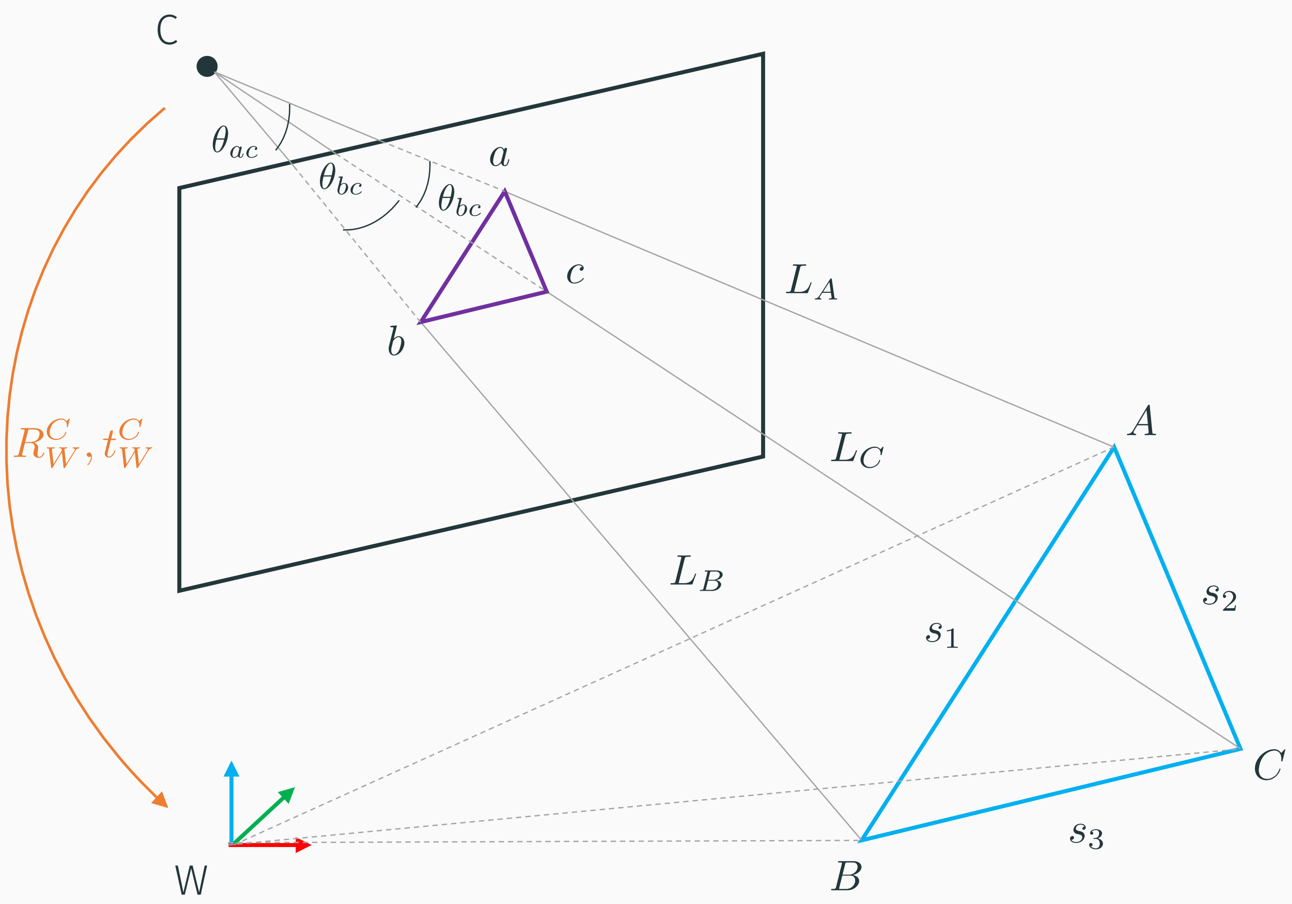
Simply put, the goal of P3P is to find the coordinates of points A, B, C in the camera coordinate frame

$$A^C = R_W^C A^W + t_W^C.$$

Note that positions of the points are known in the world frame and we assume to detect the point reprojections in the image.

Early approaches solved for positions of A, B, C in the camera frame and then used a 3D point cloud registration method for the camera pose.

P3P method



Direct approaches aim at solving the geometry of tetrahedron and start with the law of cosines to obtain set of polynomial eqs:

$$s_1^2 = L_B^2 + L_A^2 - 2L_B L_A \cos\theta_{ab}$$
$$s_2^2 = L_A^2 + L_C^2 - 2L_A L_C \cos\theta_{ac}$$
$$s_3^2 = L_B^2 + L_C^2 - 2L_B L_C \cos\theta_{bc}$$

Note that L_A, L_B, L_C represent unknowns - angles determined from the image points (the camera is calibrated) and triangle sides are known from the world frame 3D coordinates.

P3P method

3D-2D

With some algebra we can reduce the problem to a set of two polynomial equation of the 2nd order

$$(1-u)^2y^2 - ux^2 - 2y\cos\theta_{bc} + 2xyu\cos\theta_{ab} + 1$$
$$(1-v)^2x^2 - vy^2 - 2x\cos\theta_{ac} + 2xyv\cos\theta_{ab} + 1$$

where $x = L_A/L_C, y = L_B/L_C, u = s_3^2/s_1^2, v = s_2^2/s_1^2.$

For a system of *n* polynomial equations in *n* variables, the number of solutions is equal to the product of the equation degrees[7].

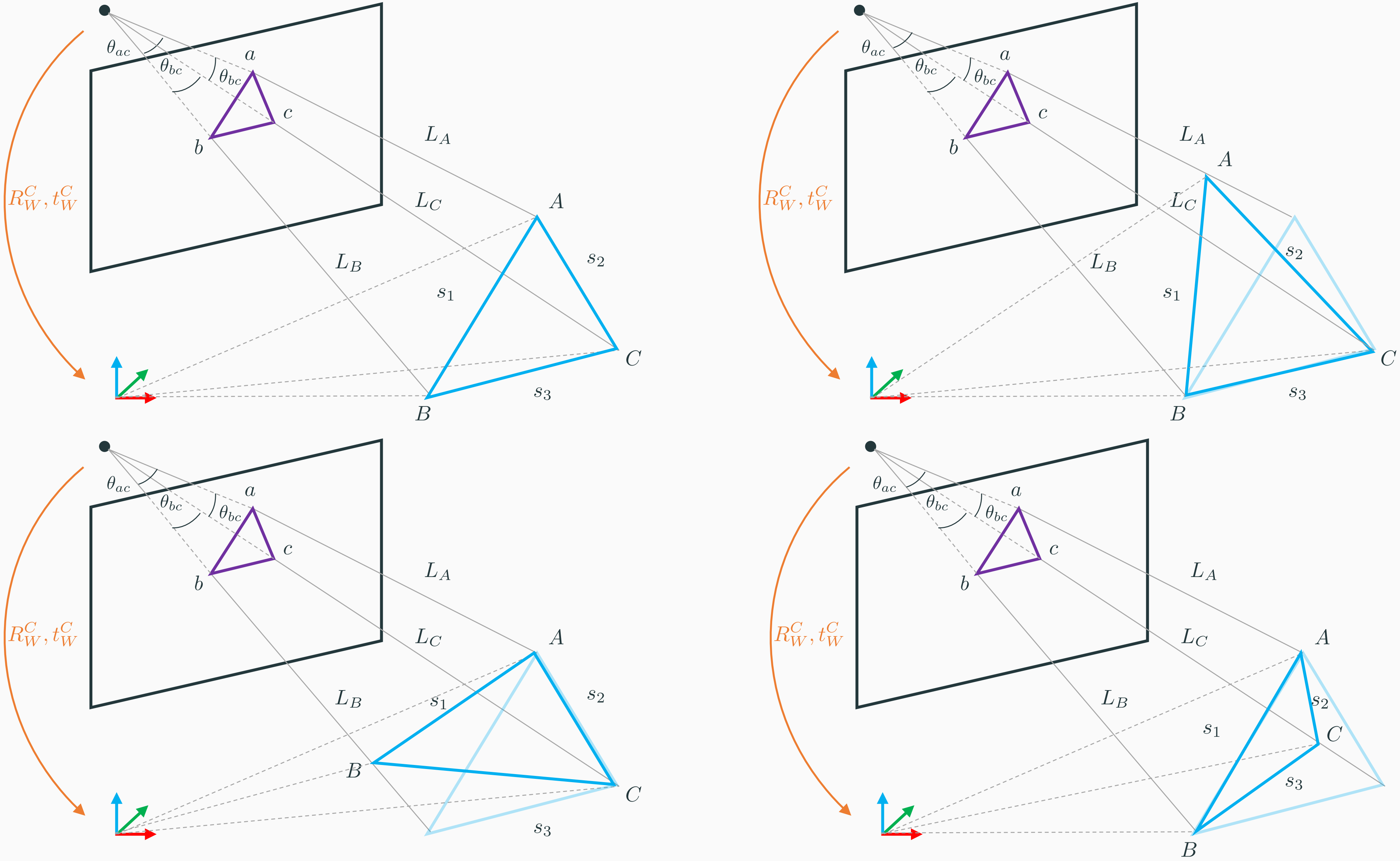
In our case, all unknowns are either linear or quadratic, thus we have 4 possible solutions, i.e., 4 possible combinations of L_A, L_B, L_C that satisfy the equations - to disambiguate the solution a 4th point is usually used.

.

[7] C. B. Garcia, T. Y. Li (1980). „On the Number of Solutions to Polynomial Systems of Equations."

P3P ambiguity

3D-2D



For example, assume that triangle sides and and projection ray angles are equal.

By rotating the triangle along one of its sides the opposite vertice will at one point intersect the projection and an equilateral triangle will be formed again.

An equivavlent effect can be also obtained my moving the camera’s coordinate system.

The 4th point can be used to solve another P3P and see which of the 8 solutions overlap.

Robust P3P

3D-2D

With the unique solution for L_A, L_B, L_C coordinates in the camera frame, we can now compute the rigid body transformation between the camera and world frame, i.e., localize the camera, by using any of the existing 3D point cloud registration algorithms, e.g., the closed-form least squares fitting.

In reality, we usually have much more than 3 points available and the correspondences that are not perfect. To obtain a robust solution in practice, we couple the P3P method with RANSAC:

1. Select 3 points randomly
2. Estimate the camera pose using P3P
3. Count the points that support this hypothesis (by comparing their reprojections from 3D position and detected points in the image)

Repeat

4. Select the best solution as the final solution.

One-stage solvers

3D-2D

The above-described method is a two-stage method: it first determines the lengths of projection rays and then obtains the rotation and translation via point alignment method.

The second stage usually involves matrix decomposition which is time-consuming and highly sensitive to the distances obtained from the first step, thus reducing the efficiency and accuracy of the final solutions.

There exists approaches that directly solve for the rotation and translation parameters and are called one-stage solvers. They do not experience alignment issues, achieve higher speed and accuracy. They are usually based on clever parametrization and change of coordinate frames[8].

[8] L. Kneip, D. Scaramuzza, R. Siegwart (2011). “A novel parametrization of the perspective-three-point problem for a direct computation of absolute camera position and orientation.”

Using more than 3 points

3D-2D

All the aforementioned approaches are 3-point methods. Note that the RANSAC approach only enables us to find the set of inliers, but in the end, we still have only methods that return solutions based only on three points.

To estimate the camera pose from $n \geq 4$ points, e.g., the inlier set returned by RANSAC, we can use the EPnP algorithm[9]. The idea is to express the *n* 3D points as a weighted sum of 4 virtual control points. The problem then reduces to estimating the coordinates of these control points in the camera frame, which can be done in *O(n)*.

The EPnP method expresses these coordinates as weighted sum of the eigenvectors of a 12 x 12 matrix and solves a small constant number of quadratic equations to pick the right weights.

[9] V. Lepetit, F. Moreno-Noguer, P. Fua (2009). „EPnP: An Accurate O(n) Solution to the PnP Problem."

Refinement via nonlinear optimization

3D-2D

Finally, note that the optimization problem that we were solving had the following form

$$\arg\min_{T_k} \sum_i ||p_k^i - \pi_k(T_k X_{k-1}^i)||^2,$$

and all the PnP variants that we presented were closed-form methods for minimizing the reprojection error.

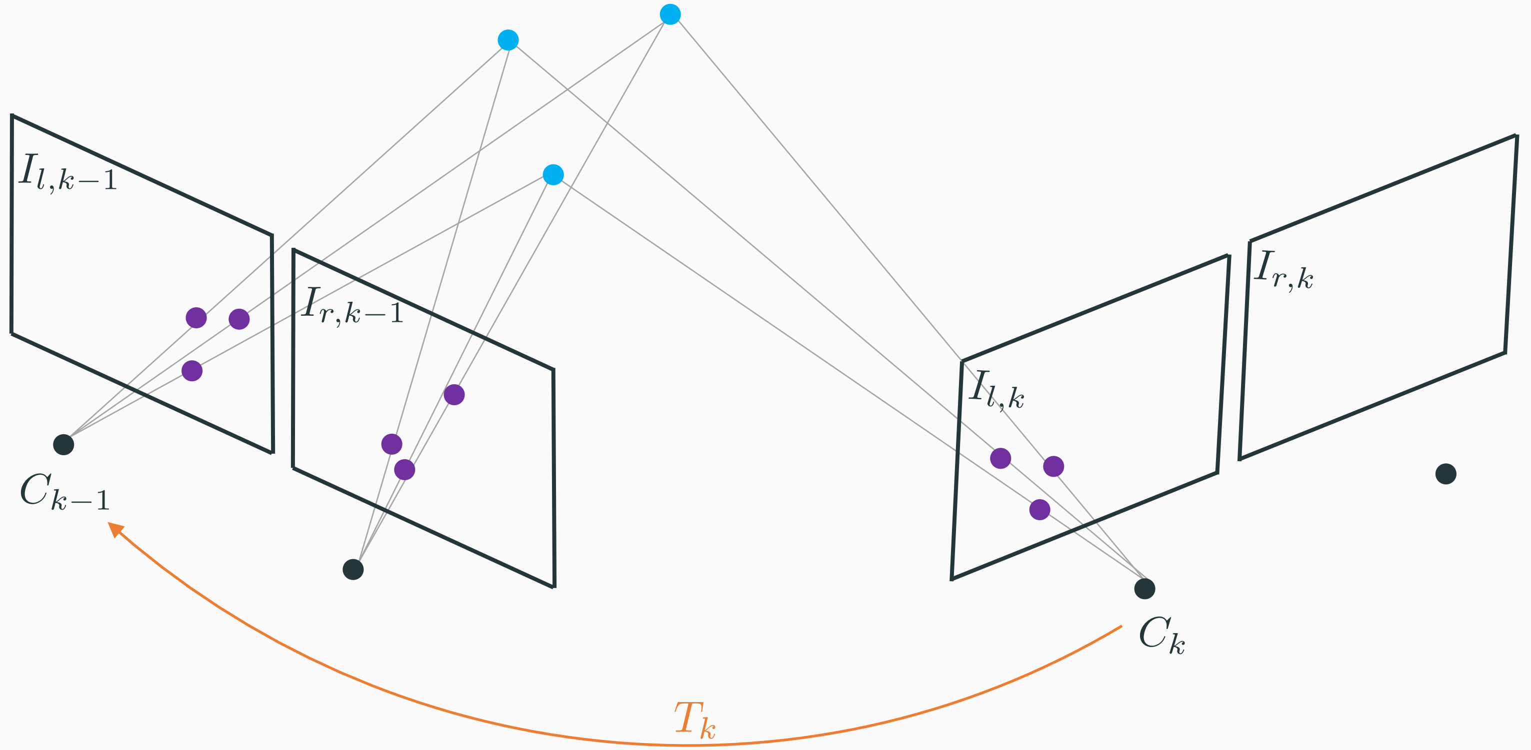
Another way to utilize more than 3 points, would be to minimize directly the above cost function using a non-linear optimization approach such as Gauss-Newton or Levenberg-Marquardt.

Under the assumption that errors are zero-mean Gaussian random vector, these solvers return the optimal solution to this non-linear least squares problem. However, they are sensitive to initial conditions and usually solution obtained by P3P or EPnP is used as the initial guess.

VO and PnP

3D-2D

How is this approach relevant for visual odometry? The stereo images from *k-1* are used to reconstruct the 3D points in the camera frame C_{k-1} that acts as the world frame from the previous slides. Correspondences are then found in the image $I_{l,k}$ or $I_{r,k}$ and the camera frame C_k is aligned to C_{k-1} using a PnP method to obtain the final transformation T_k.



Stereo 3D-2D odometry

The pipeline of a 3D-2D odometry might have the following structure:

1. Capture two stereo image pairs $I_{l,k-1}, I_{r,k-1}$ and $I_{l,k}, I_{r,k}$.
2. Detect and match features between $I_{l,k}$ and $I_{l,k-1}$.
3. Match and triangulate matched features in $I_{l,k-1}, I_{r,k-1}$
4. Compute the relative transformation T_k using 3D-2D method, e.g., P3P from the triangulated 3D point features $X_{k-1}$and their reprojections p_k
5. Concatenate camera transformation by computing $C_k = C_{k-1}T_k$.

This pipeline models stereo 3D-2D odometry, the monocular case is discussed in the next slide.

Monocular 3D-2D odometry

3D-2D

The pipeline of a 3D-2D odometry might have the following structure:

1. Do only once:
    1. Capture two image frames I_{k-2}, I_{k-1}.
    2. Detect and match features between them.
    3. Triangulate matched features from .
2. Do at each iteration:
    1. Capture new frame I_k .
    2. Extract and match with previous frame I_{k-1}.
    3. Compute camera pose (PnP) from 3D-2D matches C_k .
    4. Triangulate new feature matches between I_k, I_{k-1} .
    5. Concatenate transformation by computing $C_k = C_{k-1} T_k$.

Outline

- Problem formulation
- 3D-3D motion estimation
- 3D-2D motion estimation
  - Perspective from *n* points (PnP)
  - Stereo and mono odometry using PnP
- 2D-2D motion estimation
  - Estimating the essential matrix
  - Relative scale
- Keyframe selection

2D-2D motion estimation

The 2D-2D approach estimates motion from 2D-to-2D image point feature correspondences - p_{k-1} and p_k .

The minimal-case solution involves 5 points.

In general, the goal is to estimate the essential matrix from 2D point correspondences

$$E_k \simeq [t_k]_\times R_k,$$

Where $[t_k]_\times$ is the skew-symmetric matrix representation of the translation vector and symbol $\simeq$ means that the equivalence is valid up to a multiplicative scalar.

In other words, we can determine translation only up to scale, i.e., we have a unit length translation vector pointing in the direction of motion.

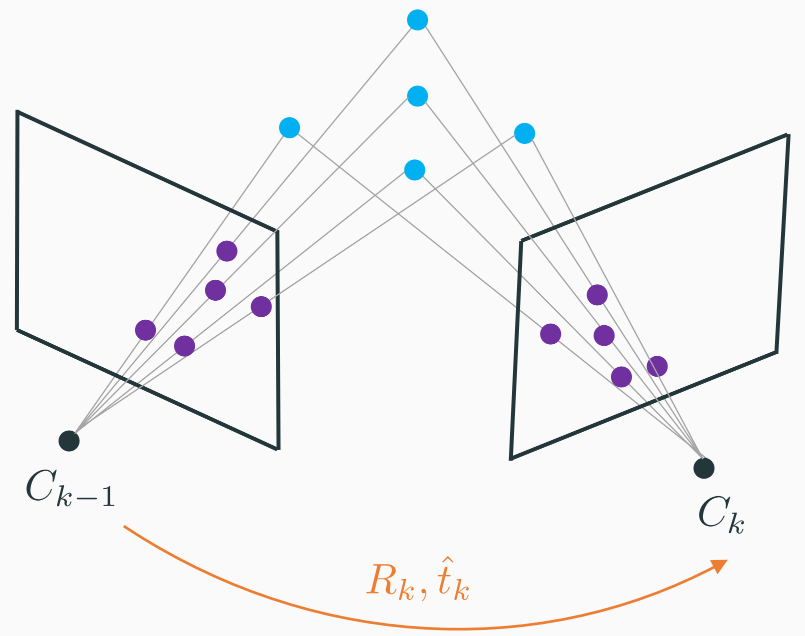
Essential matrix via 5 points

2D-2D

The main property of 2D-2D motion estimation methods is the epipolar constraint, which determines the line on which the corresponding feature point p_k^i lies in the other image.

The minimal case solution involves five 2D-2D correspondences and an efficient implementation is called the Nister's algorithm[10].

It has become the standard for 2D-2D motion estimation in the presence of outliers.



[10] D. Nister (2003). „An efficient solution to the five-point relative pose problem"

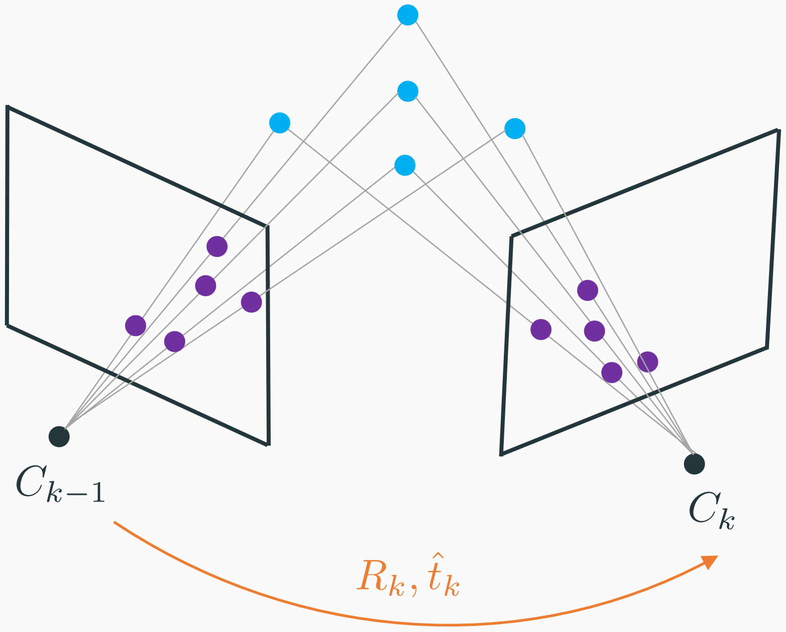
Essential matrix via 8 points

2D-2D

A simple and straightforward solution for $n \geq 8$ noncoplanar points is the Longuet-Higgins' eight-point algorithm[11] that we covered in the multiple view geometry part of the course.

Eight-point algorithm - degenerate when the 3D points are coplanar, works for both calibrated and uncalibrated cameras.

Five-point algorithm - works also for coplanar points but assumes that the camera is calibrated.



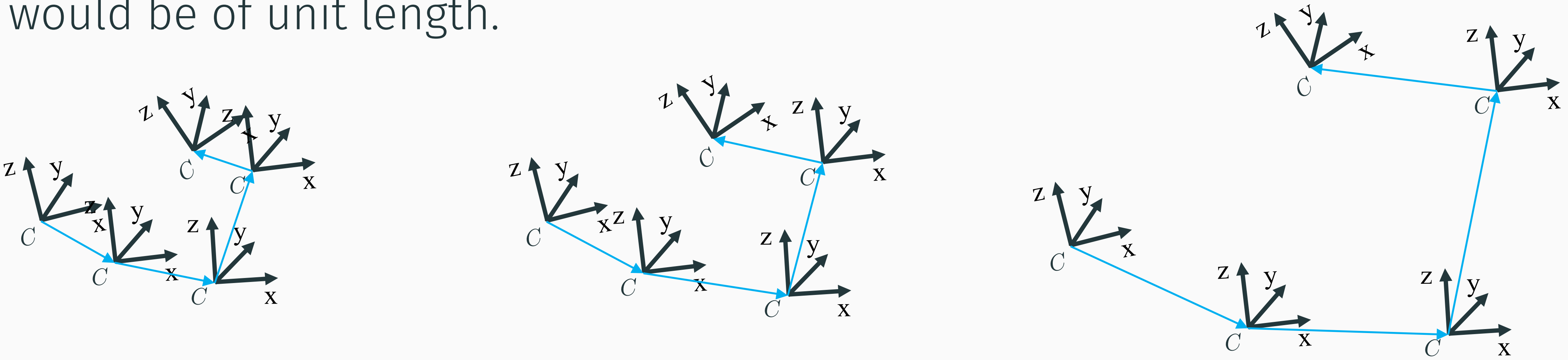
[11] H. Longuet-Higgins (1981). "A computer algorithm for reconstructing a scene from two projections"

Relative scale of the translation

2D-2D

Once we have obtained the essential matrix, we can obtain the relative rotation and translation $R_k, \hat{t}_k$. To recover the trajectory of an image sequence, recall that the different transformations $T_{0:k}$ have to be concatenated.

Since translation is a unit vector, by using this approach frame-by-fame we would obtain a camera trajectory where all the relative translation vectors would be of unit length.



However, not all translations are of the same magnitude, which begs the question if it is possible to take this into account within a 2D-2D framework?

Relative scale of the translation

2D-2D

This brings us to the notion of the relative scale.

Common approach to determinig the relative scale is to triangulate 3D points $X_{k-1,k}$ and $X_{k,k+1}$ from two subsequent image pairs and the relative distances between any combination of two 3D points can be computed.

The proper scale can then be determined from the distance ratio r between a point pair in $X_{k-1,k}$ and a pair in $X_{k,k+1}$

$$r_k = \frac{||X^i_{k-1,k} - X^j_{k-1,k}||}{||X^i_{k,k+1} - X^j_{k,k+1}||}.$$

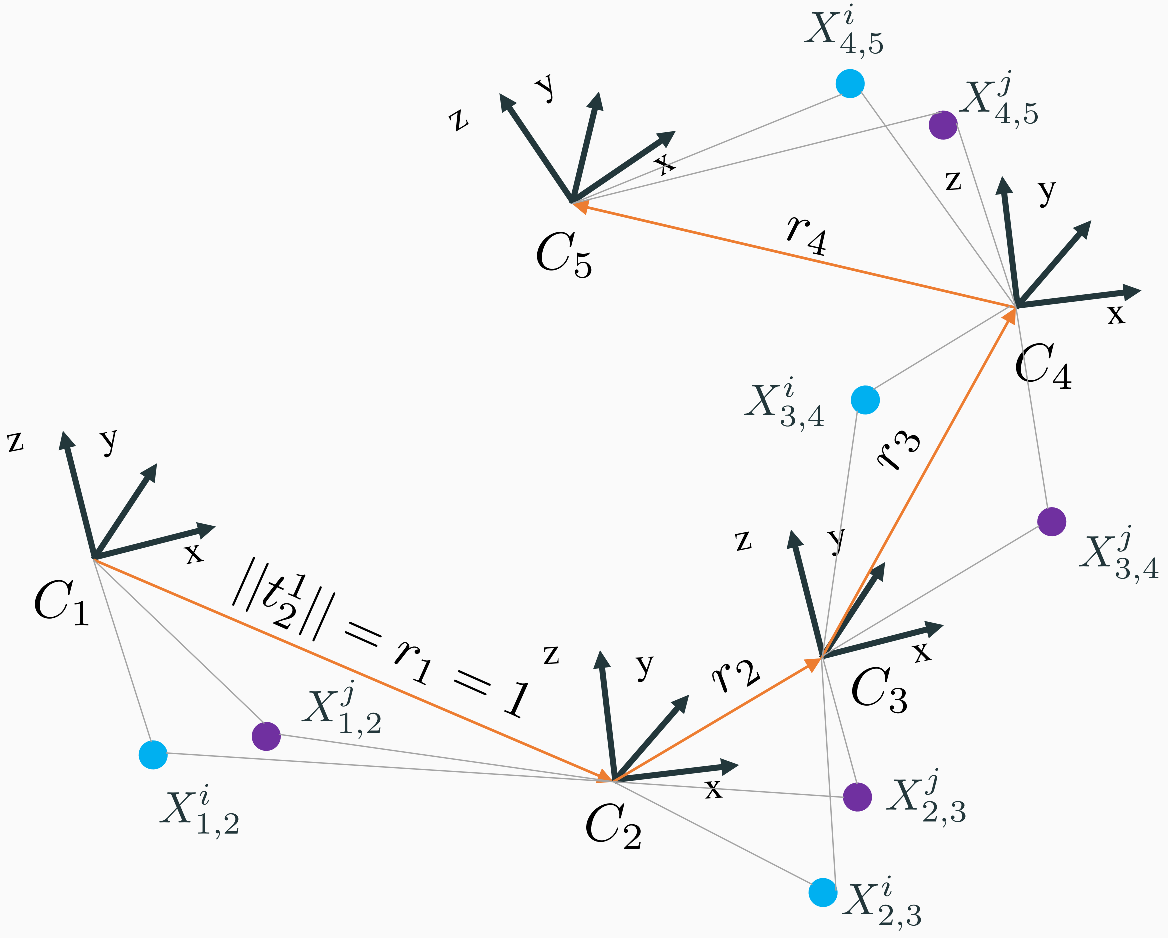
For robustness, the scale ratios for many point pairs are computed and the mean is used. The translation vector is then scaled with this distance ratio - requires features to be matched over multiple frames (at least three).

Relative scale of the translation

2D-2D

We bootstrap the monocular odometry by computing the relative transformation between the first two views via essential matrix decomposition (translation norm is unit). Then, we triangulate points from the first two views, and as the third image is captured, we triangulate from the second and third view.

The ratio of distance norms between image pairs gets us the relative scale w.r.t to the first pair etc.



2D-2D odometry pipeline

The pipeline of a 2D-2D odometry might have the following structure:

1. Capture new frame I_k .
2. Detect and match features between I_k and I_{k-1} .
3. Compute essential matrix for image pair I_{k-1}, I_k
4. Decompose essential matrix into R_k, t_k and form T_k .
5. Compute relative scale and rescale t_k accordingly .
6. Concatenate transformation by computing $C_k = C_{k-1}T_k$.

Mmotion estimation based on $E_k \simeq [t_k]_\times R_k$, cannot handle pure rotation, there must exists at least some translational component in the overall motion (to handle this homography estimation is classically used).

Feature matches – camera shift with frontoparallel scene

2D-2D

feature matches depending on the camera displacement

translation to the right

rotation to the right

move forward

rotation around the optical axis

Outline

- Problem formulation
- 3D-3D motion estimation
- 3D-2D motion estimation
  - Perspective from *n* points (PnP)
  - Stereo and mono odometry using PnP
- 2D-2D motion estimation
  - Estimating the essential matrix
  - Relative scale
- Keyframe selection

Keyframe selection

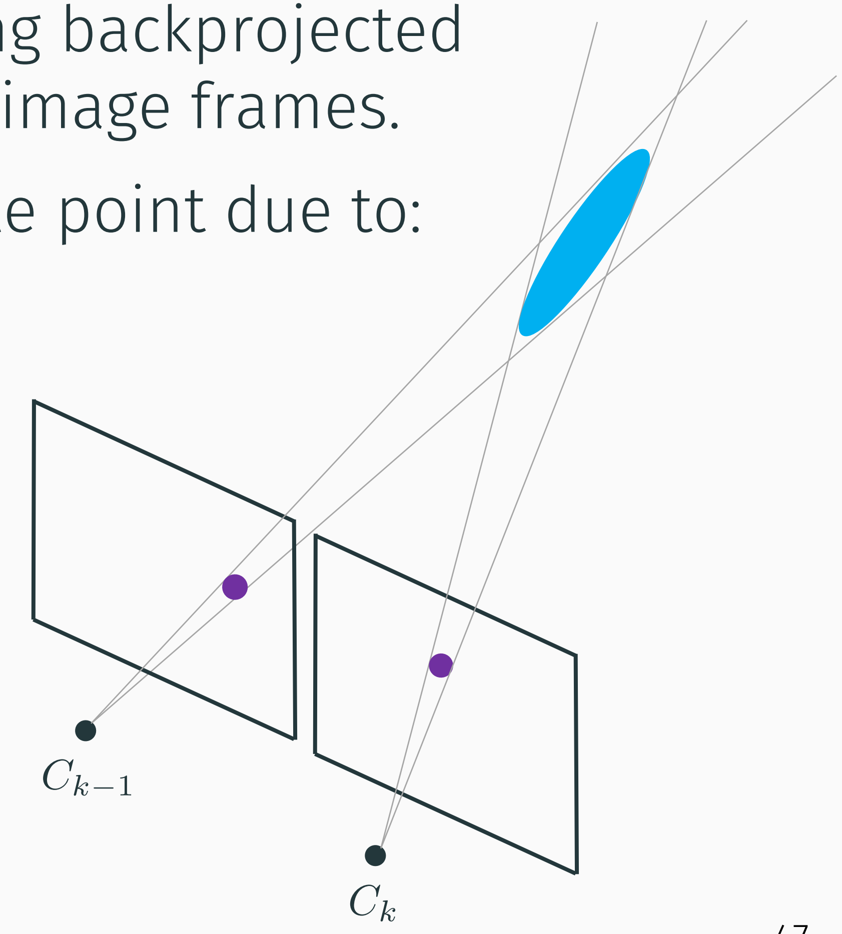
Some of the previous motion estimation methods require triangulation of 3D points.

Triangulated 3D points are determined by intersecting backprojected rays from 2D image correspondences of at least two image frames.

In reality, more than 2 lines never intersect in a single point due to:

- image noise
- camera model and calibration errors
- and feature matching uncertainty.

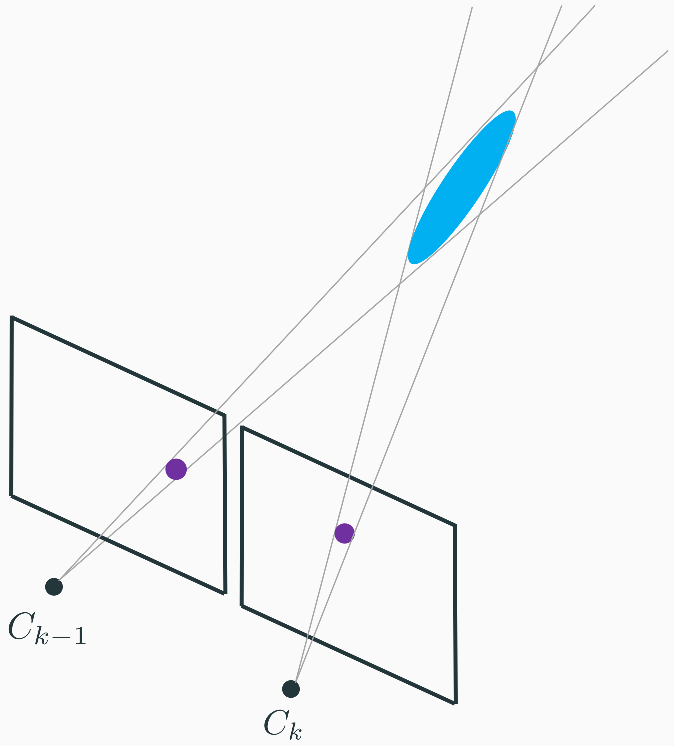
The point at minimal distance from all intersecting rays can be taken as an estimate of the 3D point position or a least squares solution can be found.

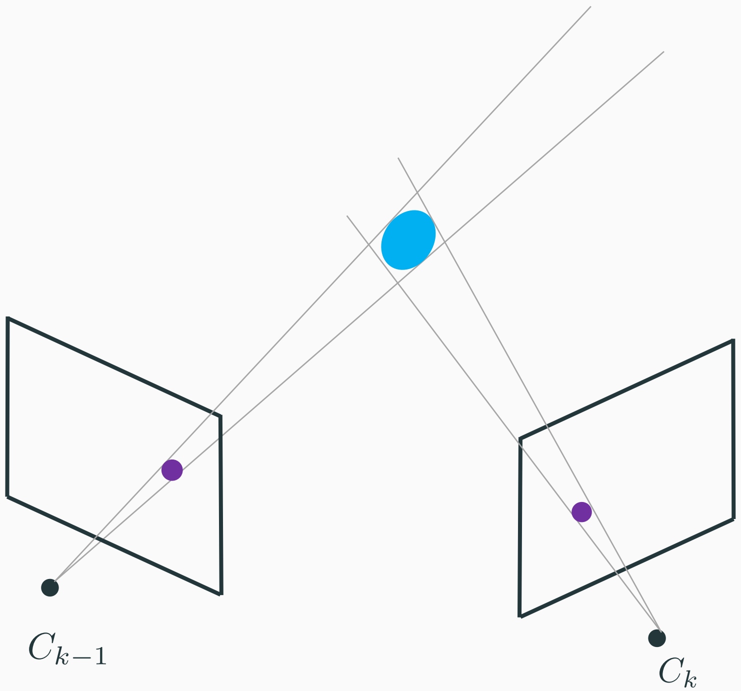


Keyframe selection

When frames are taken at relatively close positions with respect to scene depth (i.e., baseline is much smaller than the feature depth), 3D triangulated points will exhibit large depth uncertainty.

This can affect motion estimation accuracy and in general features should be triangulated at good baseline to depth ratios.





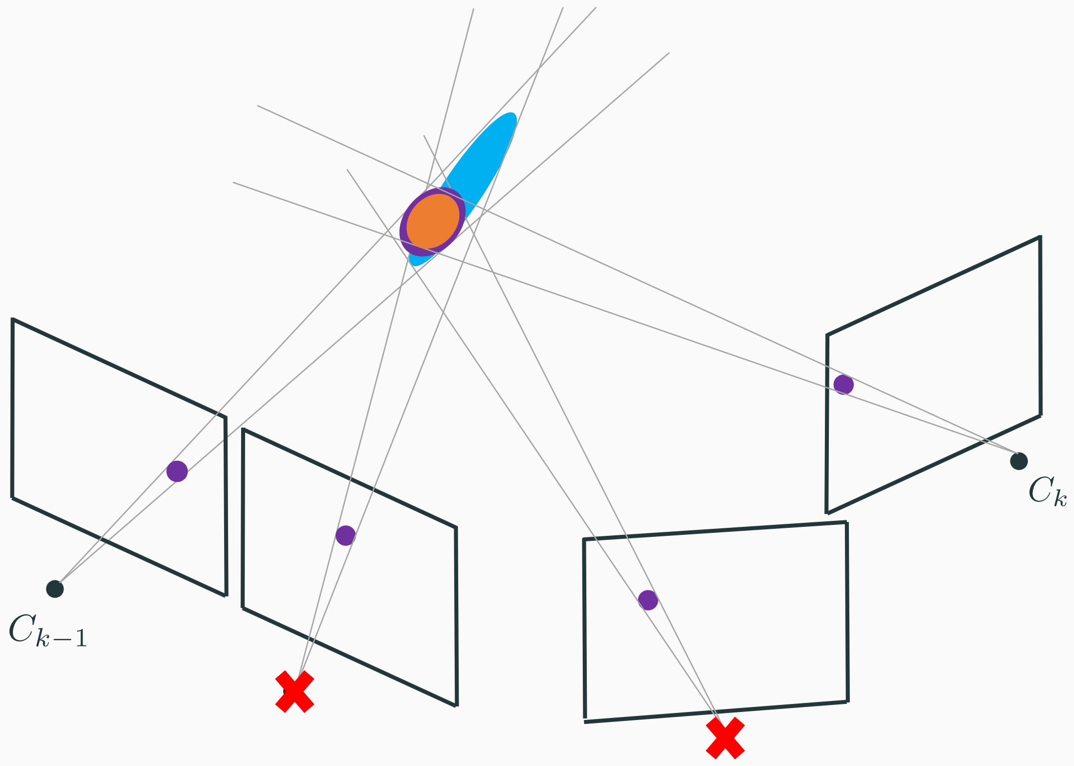
Keyframe selection

One way to avoid this consists of skipping frames until the average uncertainty of the 3D points decreases below a certain threshold. The selected frames are called keyframes.

Keyframe selection is a very important step in VO and should always be done before updating the motion.

Rule of the thumb: add a keyframe when the following threshold is met *(e.g., 10-20%)*

$$\frac{\text{keyframe\_distance}}{\text{average\_depth}} > \tau.$$



Questions?