TSBB15
Computer Vision

Lecture 12

Structure from Motion, PnP, Bundle adjustment, Project 2

Per-Erik Forssén

Calibrated 2-view geometry

Recap from previous lecture:

- Given corresponding points in two views, and known intrinsics **K**, we can determine **E**
- Given **E** we can determine the **relative pose** of the cameras (**R**,**t**) from camera 1 to camera 2
  - The **relative** **R**, and the **direction** of **t** can be determined, but **not its absolute scale**

To proceed we can e.g. choose the WCS origin as camera 1, and set |**t**|=1

Adding views with PnP

If we have correspondences also with a third view we can use the **Perspective n-Point problem** (PnP) to find the third camera:

given: a set of 2D ⇹ 3D correspondences

$$\{\mathbf{y}_n \leftrightarrow \mathbf{x}_n\}_{n=1}^{N}$$

sought: the **absolute camera pose**, such that

$$\mathbf{y}_n \sim [\mathbf{R}|\mathbf{t}]\,\mathbf{x}_n \text{ , for } n = 1, \ldots, N$$

Adding views with PnP

Minimal case: $N = 3$ (P3P)

- Gives up to 4 possible solutions for [**R** | **t**]
- Use together with RANSAC
- Refine the found solution by optimisation on the **inlier set**

For $N > 3$, we minimise a cost function over **R** and **t**:

$$J(\mathbf{R}, \mathbf{t}) = \sum_{n=1}^{N} d_{\mathrm{PP}}^{2}(\mathbf{y}_n, [\mathbf{R}|\mathbf{t}]\,\mathbf{x}_n)$$

Note: Here we also get a length on **t**. The **relative scale** of the translation compared to the model.

Adding views with PnP

- Minimisation over **R** and **t**…

$$J(\mathbf{R}, \mathbf{t}) = \sum_{n=1}^{N} d^2_{\mathrm{PP}}(\mathbf{y}_n, [\mathbf{R}|\mathbf{t}]\,\mathbf{x}_n)$$

- If we minimize over the elements of **R** we will in general not get an $\mathbf{R} \in \mathrm{SO}(3)$
- By choosing an appropriate representation of **R**, SO(3) can automatically be enforced, more on this later.
- Note also that J(**R**,**t**) is not a true ML estimator (**why**?)

Finding outliers with robust PnP

- Use a minimal PnP solver inside RANSAC
- Incorrect correspondences are easier to find with robust PnP than with a robust essential matrix solver:

  - PnP uses a point-to-point error:

$$\boldsymbol{\epsilon} = \mathbf{y}_n - \mathrm{n}_{\mathrm{P}}([\mathbf{R}|\mathbf{t}]\mathbf{x}_n)$$

  - The five point algorithm uses a point-to-line error

$$\epsilon = \mathbf{y}_n^T \mathrm{n}_{\mathrm{L}}(\mathbf{E}\mathbf{y}'_n)$$

  - Any point on the epipolar line will be accepted!

Minimal solvers for robust PnP

- E.g. Kneip's method from CVPR11:

https://github.com/urbste/MLPnP_matlab_toolbox/tree/master/p3p_code_final

- More recent CVL solver from ECCV18:

https://github.com/midjji/lambdatwist-p3p

Calibrated Structure from Motion(SfM)

Given:

- N views I_n (images) of a static scene
- In each view: a set of interest points
  - Tentative (likely) correspondences between views
- The internal camera calibration

Sought:

- The 3D positions of the interest points
- The camera poses of each view

  For a moving camera this is the **ego-motion**

These two are determined simultaneously!

Reconstruction ambiguity

- The choice of **world coordinate system is arbitrary** we can put it in camera 1, …,N or elsewhere.
- The reconstruction also has a **scale ambiguity**:
  - Scale the scene by a factor *s*
  - Scale the camera positions by a factor *s*
  - All images will look the same!

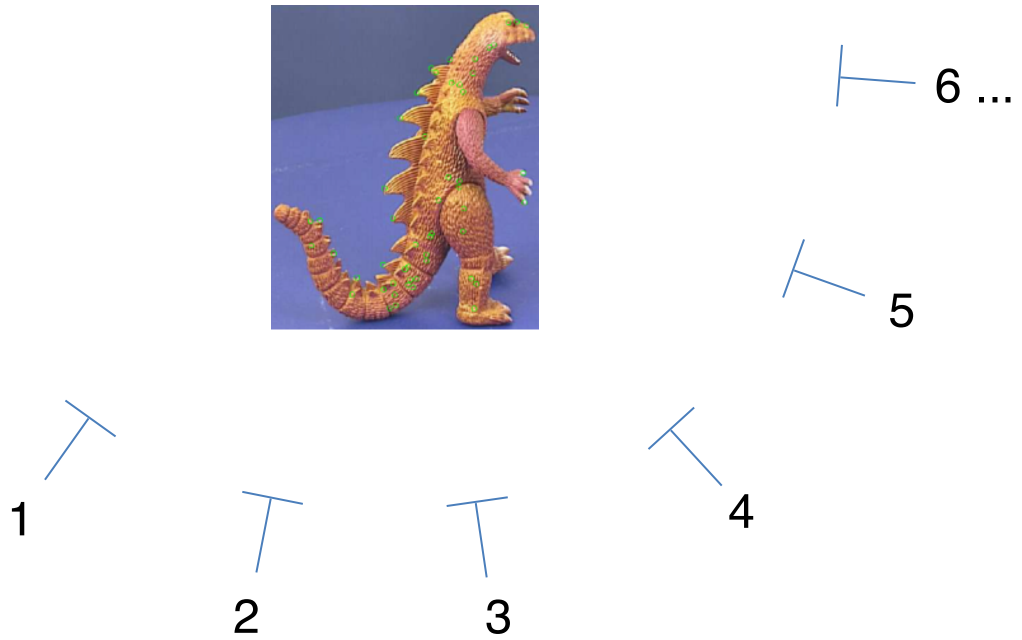
Resolving the ambiguities:

- The scale can be resolved using known distances e.g:
  - the distance between cameras 1 and 2 is $|\mathbf{t}_{12}|=0.43$m
  - the modelled object is known to be 2.5dm high
- The WCS can be set using georeferencing
  - three or more known WCS locations are needed

An SfM pipeline

- A program that solves the **structure from motion** problem is called an **SfM pipeline**; a graph of processing blocks
- The pipeline can be formulated in many different ways, depending on, for example,
  - computational architecture (PC or cluster)
  - robustness and/or, required accuracy in the result
  - …
- Main variants: incremental and parallel/global.

Multiple cameras in a static scene



Simple incremental SfM pipeline

1. Select a pair of views
2. Find correspondences
3. Estimate **E**, then extract $\mathbf{R}_2$, $\mathbf{t}_2$

For each new view *j*:

4. Add new correspondences between new view and the old ones
5. Find new camera pose $[\mathbf{R}_j,\mathbf{t}_j]$ with PnP

Repeat 4,5 until all views have been used.

Simple SfM pipeline

- This approach is simple
  - Based on solving small *local* problems
  - Using standard techniques
- However, no *global* consistency of the resulting 3D points is enforced
- In practice, small errors will accumulate to large errors ⇒ the pipeline is brittle.

Bundle adjustment

To reduce drift, and to obtain a more robust pipeline we need to periodically make a **global refinement** of **all camera poses** and **all 3D points**

This is called **bundle adjustment** (BA)

Issue: The 3D points are visible only in a limited number of views

Bundle adjustment

- Bundle adjustment implies that we have two sets:
  - *P* is a set of reconstructed 3D points
  - *Q* is a set of images or camera views
    - For each view in *Q* there exists an estimate of the corresponding camera pose, and visible image points
- All 3D positions in *P* and poses in *Q* are expressed in a global coordinate system, e.g., one of the camera coordinate systems

Bundle adjustment

- The matching between P and Q can be measured in terms of **the total squared re-projection error** ε, defined as

$$\epsilon = \underbrace{\sum_{i \in \mathcal{P}}}_{\text{3D points}} \underbrace{\sum_{j \in \mathcal{Q}}}_{\text{views}} v_{ij} d^2_{\mathrm{PP}} \left(\mathbf{y}_{ij}, [\mathbf{R}_j, \mathbf{t}_j] \, \mathbf{x}_i \right)$$

$v_{ij} \in \{0, 1\}$ - visibility index

Bundle Adjustment

v_{ij} is a binary mask that describes if point $i \in \mathcal{P}$ is visible in view $j \in \mathcal{Q}$ (v_{ij}=1) or not (v_{ij}=0)

aka. **missing data pattern**

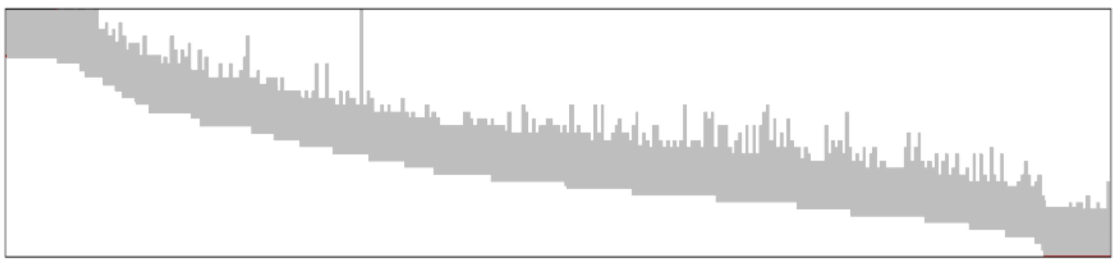


Fig. 14 Missing data pattern for a subset of the Oxford dinosaur sequence. The gray elements correspond to observed data.

Viktor Larsson, Carl Olsson. "Convex Low Rank Approximation", IJCV 2016

Bundle Adjustment

Given initial estimates of the 3D points $\mathbf{x}_i \in \mathcal{P}$ and the poses $[\mathbf{R}_j \ \mathbf{t}_j] \in \mathcal{Q}$:

$$\epsilon(\{\mathbf{R}_j, \mathbf{t}_j\}_{j\in\mathcal{Q}}, \{\mathbf{x}_i\}_{i\in\mathcal{P}}) = \sum_{i\in\mathcal{P}} \sum_{j\in\mathcal{Q}} v_{ij} d^2_{\mathrm{PP}} \left(\mathbf{y}_{ij}, [\mathbf{R}_j \ \mathbf{t}_j]\, \mathbf{x}_i\right)$$

In summary: **bundle adjustment** minimizes the re-projection error by adjusting both **3D points** and **camera poses**

Bundle adjustment

- The bundle adjustment step optimizes a highly non-linear function ε

  - Must be done with iterative minimization procedures

  - Good initial solutions are critical for success!

Proposed: Incremental SfM Pipeline

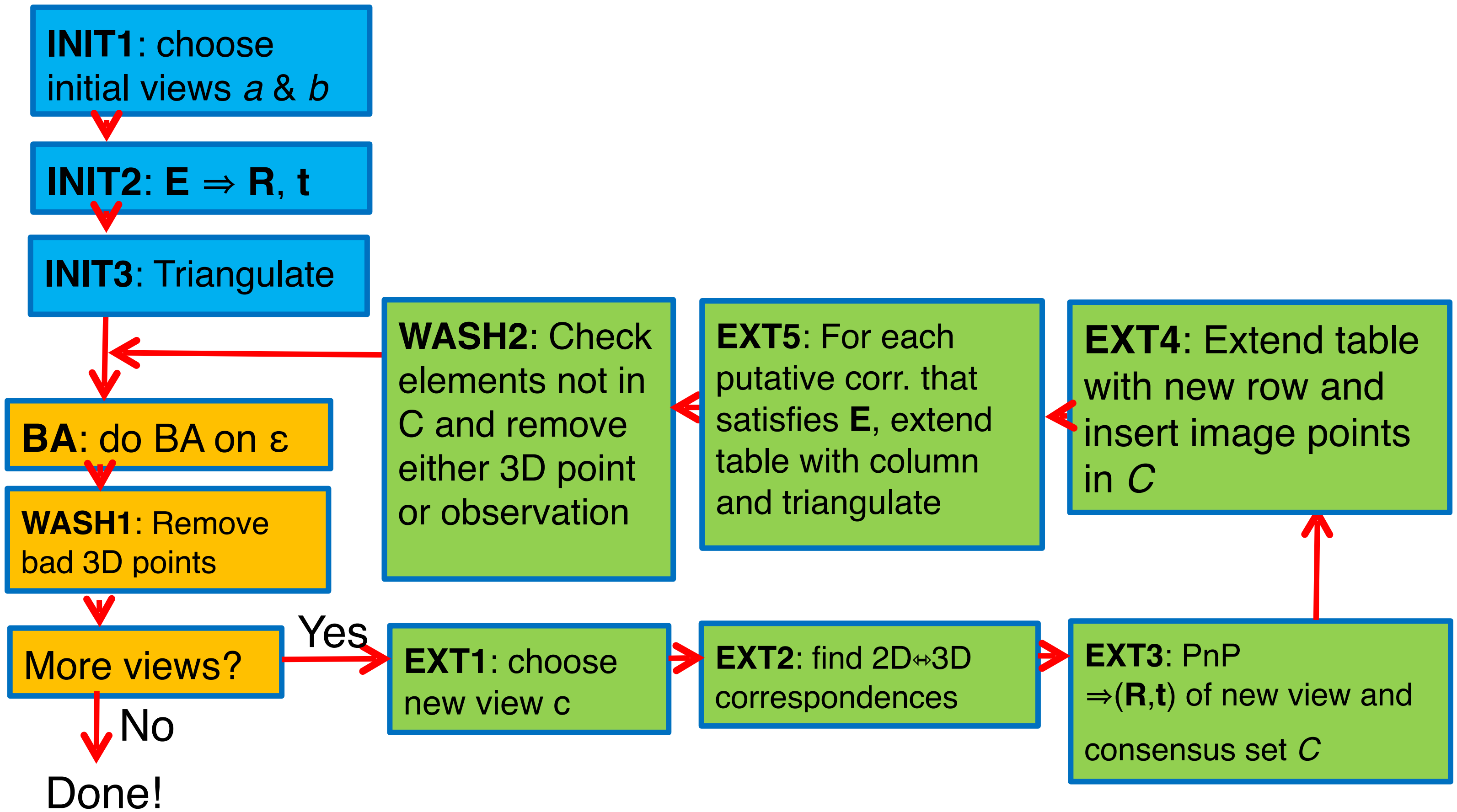
Start with minimal *P* and *Q*

1. The initial *Q* contains only two views, e.g. I_1 and I_2
2. The initial *P* contains only the 3D points that can be triangulated from these two views

Iterate:

3. Bundle adjust over *P* and *Q*
4. Add a new view to *Q*
5. Extend *P* with new 3D points that can be triangulated from the extended *Q*
6. Repeat until all views are added to *Q*

Proposed: Incremental SfM Pipeline



Initialization of *P* and *Q*

Initially, *P* and *Q* are empty

STEP: INIT1:

- Pick two views, *a* and *b* (*e.g.* I_1 *and* I_2)
- Choose *a* and *b* wisely:
  - We want high accuracy in the triangulation step
    - They should have as large baseline as possible
    - Be far apart in the image sequence
  - We want high accuracy in the pose estimation
    - They should have as many common points as possible
    - Be close to each other in the sequence

Initialization of *P* and *Q*

STEP INIT2:

- Solve the correspondence problem and determine $\mathbf{E}_{ab}$
  - Use putative correspondences and RANSAC!
- From $\mathbf{E}_{ab}$, get $\mathbf{R}_b$ and $\mathbf{t}_b$ (relative pose, no scale!)
- Set $\mathbf{R}_a = \mathbf{I}$ and $\mathbf{t}_a = \mathbf{0}$
  - Camera *a* defines the global coordinate system!
- Put $\{I_a, I_b\}$ together with their poses in *Q*

Initialization of P and Q

- **STEP INIT3**:
- With $\mathbf{C}_a = [\ \mathbf{I} \mid \mathbf{0}\ ]$ and $\mathbf{C}_b = [\ \mathbf{R} \mid \mathbf{t}\ ]$
- Triangulate 3D points from point correspondences in views I_a and I_b
  - Use only correspondences consistent with **E**
  - These can still be outliers!
- Put these 3D points in P
- Set $v_{ij} = 1$ for all points in P for views I_a and I_b

Bundle adjustment

Given that *P* and *Q* are defined, the next step is bundle adjustment (BA)

STEP BA:

The bundle adjustment step optimizes the highly non-linear function ε

NOTE: ε depends on *P* and *Q*!

Use a **non-linear least squares minimiser**

Python: `scipy.optmize.least_squares`

Matlab: `lsqnonlin`

C: `levmar, ceres-solver`

Bundle adjustment

- The result of the bundle adjustment step is a set *P* of 3D points and a set *Q* of camera poses that are optimally compatible
  - According to the total squared re-projection error
  - Implies a Maximum-Likelihood estimate of 3D points and poses assuming Gaussian noise on the image coordinates $\mathbf{y}_{ij}$

Wash *P*

- The set of 3D points, *P*, may contain false 3D points
  - They are triangulated from 2D correspondences that satisfy an epipolar constraint, but such a constraint is not sufficient for true correspondence.

STEP WASH1:
Remove bad 3D points:

  - If large re-projection errors
  - Optionally also: If they move a lot during BA

Extending P and Q

Q contains views $I_1 \ldots I_{j-1}$

If |Q| = j-1 = N, done!

STEP EXT1:

Choose a new camera view I_j and add it to Q

Relatively close to at least one view in Q

Extending *P* and *Q*

STEP EXT2:

- We have putative 2D correspondences between views $I_{j\text{-}1}$ and I_j
- Most of the points in $I_{j\text{-}1}$ have corresponding 3D points in *P*
- Combine these to obtain a set of putative 3D-2D correspondences for view I_j

Extending *P* and *Q*

STEP EXT3:

- Use robust PnP and the putative correspondences to determine the camera pose of view I_j
- Produces:
  - a camera pose (**R**, **t**) of the new view
  - a consensus set *C* of 3D-2D correspondences that are consistent with this camera pose
- **Question**: What do we do with the outliers?

Extending *P* and *Q*

STEP EXT4:

- Add a new row to the table
  - *Q* is increased by one new view
- Insert coordinates of the 2D points in *C* into this row
  - Set all other entries to “invisible”

Extending *P* and *Q*

STEP EXT5:

- Determine **E** between views $I_{j\text{-}1}$ and I_j (e.g. $\mathbf{E} = [\mathbf{e}_{12}]_\times \mathbf{C}'_1 \mathbf{C}'^{+}_2$)
- Go through all putative correspondences between views $I_{j\text{-}1}$ and I_j, not already triangulated
- If they are compatible with **E**:
  - Triangulate 3D point and add to table
  - Can still be outliers since epipolar constraint is only necessary (not sufficient)

Extending *P* and *Q*

STEP WASH2:

- There may be existing 2D↔3D corresp. that do not appear in the consensus set *C*
  - If the feature has only been visible in a few views, remove it completely
  - If it has been visible for some time, only the 2D observation should be removed

Iterate

Return to

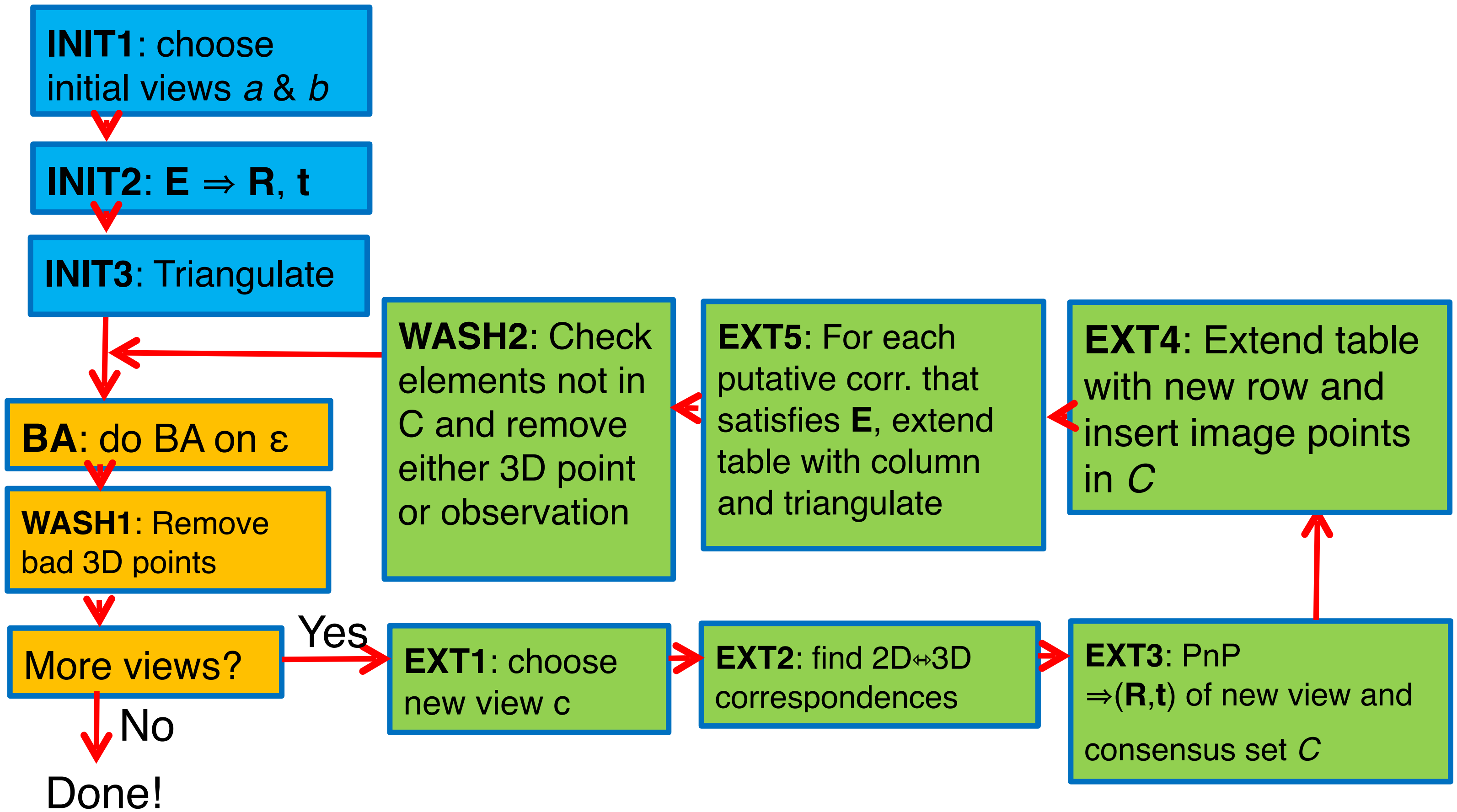
STEP BA:

- Setup ε
- do bundle adjustment (minimise)

If not all views are in *Q* goto **EXT1**

Otherwise: done

Proposed: Incremental SfM Pipeline



BREAK

Bookkeeping

One column for each point in *P*

	$\mathbf{x}_1$	$\mathbf{x}_2$	$\mathbf{x}_3$	…	$\mathbf{x}_P$
$\mathbf{R}_1, \mathbf{t}_1$	$\mathbf{y}_{11}$	$\mathbf{y}_{21}$	$\mathbf{y}_{31}$	…	$\mathbf{y}_{P1}$
$\mathbf{R}_2, \mathbf{t}_2$	$\mathbf{y}_{12}$	$\mathbf{y}_{22}$	$\mathbf{y}_{32}$	…	$\mathbf{y}_{P2}$
⋮	⋮	⋮	⋮	⋱	⋮
$\mathbf{R}_Q, \mathbf{t}_Q$	$\mathbf{y}_{1Q}$	$\mathbf{y}_{2Q}$	$\mathbf{y}_{3Q}$	…	$\mathbf{y}_{PQ}$

One row for each view in *Q*

In addition: each row can contain image points for a view that has not been associated with any 3D points in P

Each frame adds one row and some columns to the table

Same shape as the visibility v_{ij}

Outlier removal

The step **WASH1** may have to be extended to be more strict.

Options: Remove 3D-points that

- Have large re-projection errors after BA
- Have moved far during BA
- Lie far from the 3D centroid

You could also rerun BA after this step

Practical issues (I)

Parameterization of **R**:

- **unit quaternions**, **exponential coordinates** (aka. axis-angle vector) $\boldsymbol{\alpha n}$, ...
- See IREG compendium chapter 11.

- define functions rmatrix_to_expc() and expc_to_rmatrix()
- Set up test cases that verify that these are each other's inverse.

Practical issues (II)

Parameterization of the projection operators [**R t**]:

- The camera center is located at $-\mathbf{R}^T\mathbf{t}$
- Possible problem in the optimization:
- The camera center depends on both **R** and **t**
  - Small variations in **R** means large variations in the camera center if **t** is large
- Alternative parameterization: $\mathbf{R}^T[\ \mathbf{I}\ -\mathbf{t}\ ]$
- **R**,**t** are now in the **world coordinate system** (WCS) e.g. camera center is located at **t** (and is independent of **R**)

Practical issues (III)

`scipy.optimize.least_squares` (and others) minimizes functions like:

$$\epsilon(\mathbf{r}) = \mathbf{r}^T\mathbf{r}$$

Here $\mathbf{r} = (r_1 \;\; r_2 \;\; \ldots \;\; r_P)^T$ is the **residual vector** for our problem. The squaring is done implicitly in the optimizer.

Each r_k is a **residual**, a difference between a prediction from the model and an observation

Practical issues (III)

For best results, each reprojection should thus generate two residuals (why?)

r_{k1} is the signed difference in the *x*-direction

r_{k2} is the signed difference in the *y*-direction

The corresponding ε now becomes

$$\epsilon = r_{11}^2 + r_{12}^2 + r_{21}^2 + r_{22}^2 + \ldots + r_{P1}^2 + r_{P2}^2$$

We need to implement a function returning

$$\mathbf{r} = (r_{11}\ r_{12}\ r_{21}\ r_{22} \ldots\ r_{P1}\ r_{P2})^T$$

Practical issues (IV)

In general, a **non-linear least squares minimizer** operates on a parameter vector $\boldsymbol{\theta}$

$$\epsilon(\boldsymbol{\theta}) = \mathbf{r}(\boldsymbol{\theta})^T \mathbf{r}(\boldsymbol{\theta})$$

It uses a Taylor expansion of the residual vector **r** around $\boldsymbol{\theta}$:

$$\mathbf{r}(\boldsymbol{\theta} + \mathbf{h}) \approx \mathbf{r} + \mathbf{J}\mathbf{h}$$

J is the **Jacobian** of **r** with respect to $\boldsymbol{\theta}$

J contains the derivatives of the elements in **r** w.r.t. the elements in $\boldsymbol{\theta}$

Practical issues (IV)

An approximation of ε is then given as

$$\epsilon(\boldsymbol{\theta} + \mathbf{h}) \approx (\mathbf{r} + \mathbf{J}\mathbf{h})^T (\mathbf{r} + \mathbf{J}\mathbf{h}) =$$

$$= \mathbf{r}^T \mathbf{r} + 2\mathbf{h}^T \mathbf{J}^T \mathbf{r} + \mathbf{h}^T \mathbf{J}^T \mathbf{J}\mathbf{h}$$

Given this approximation, we want to determine **h** such that ε(**Θ** + **h**) is minimized

Determine **h** such that $\nabla_{\mathbf{h}}\varepsilon = 0$:

$$\mathbf{J}^T \mathbf{r} + \mathbf{J}^T \mathbf{J}\mathbf{h} = 0$$

Normal equations of the least squares problem defined by ε(**Θ** + **h**)

Practical issues (IV)

The minimizer will thus first compute or approximate **J** and then solve the normal equation to update the parameters as:

$$\boldsymbol{\theta}_{n+1} = \boldsymbol{\theta}_n + \mathbf{h}$$

Several simplifying tricks can be applied to these computations, see:

Triggs, McLauchlan, Hartley, Fitzgibbon, *Bundle Adjustment – A Modern Synthesis*, Vision Algorithms: Theory and Practice, 2000

Practical issues (IV)

In our case, the parameters are:

- a set of 3D points in P (3 [or 4] parameters each)
- a set of camera poses in Q (3+3=6 [or 7] params each)

- Each element of **r** is a signed distance in x or y-direction between an image point $\mathbf{y}_{ij}$ and the corresponding projected 3D point in view j: $[\mathbf{R}_j\ \mathbf{t}_j]\ \mathbf{x}_i$

Practical issues (IV)

Useful observation:

- The re-projection error for point $\mathbf{y}_{ij}$ depends on
  - the pose of camera j but **not on any other cameras**
  - The position of 3D point $\mathbf{x}_i$ but **not on any other 3D points**
- Each row in $\mathbf{J}$ has, at most, 6 + 3 = 9 non-zero elements, in well defined positions

Practical issues (IV)

- **J** is very sparse (many elements are = 0)
- Implies, we only need to compute the “non-zero” elements of **J**
- A significant improvement in speed (and accuracy) is possible if we can tell the optimizer which the “non-zero” elements in **J** are
- For example, see:

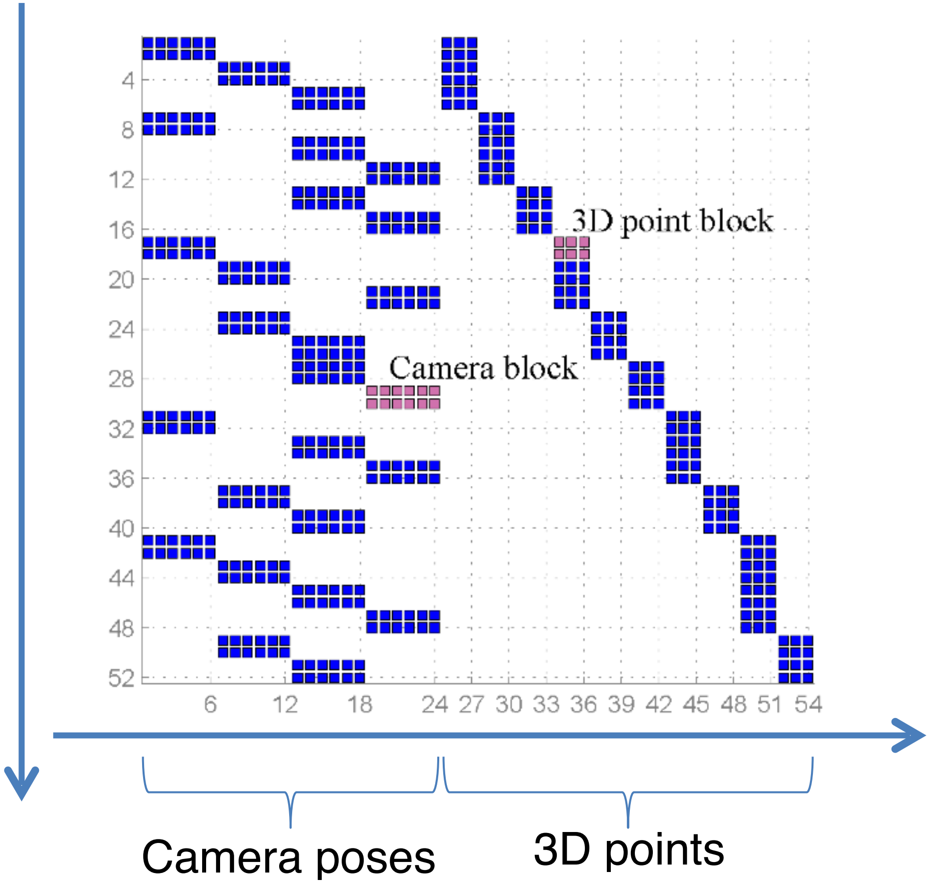
  `jac_sparsity` parameter to `optimize.least_squares`
- This is tested in extra task of CE3

Practical issues (IV)

$\frac{\partial \epsilon}{\partial y_1}$ and $\frac{\partial \epsilon}{\partial y_2}$ for image points in different views

Example of **J**

From PhD thesis of Johan Hedborg 2012



Practical issues (V)

To make things simple, all coordinates and re-projection errors may be expressed in C-normalized coordinates

However, the real errors are in pixel coordinates

- A more correct approach is thus to use pixel coordinates for the points and the re-projection errors
- Include also **K** (and lens distortion) in the calculations of ε
- For zero skew, equal focal lengths, and no lens distortion **the results are the same** though.

Project 2

Project description at:

http://www.cvl.isy.liu.se/education/undergraduate/tsbb15/3d-reconstruction-project

This year: Two groups of 5 people each.

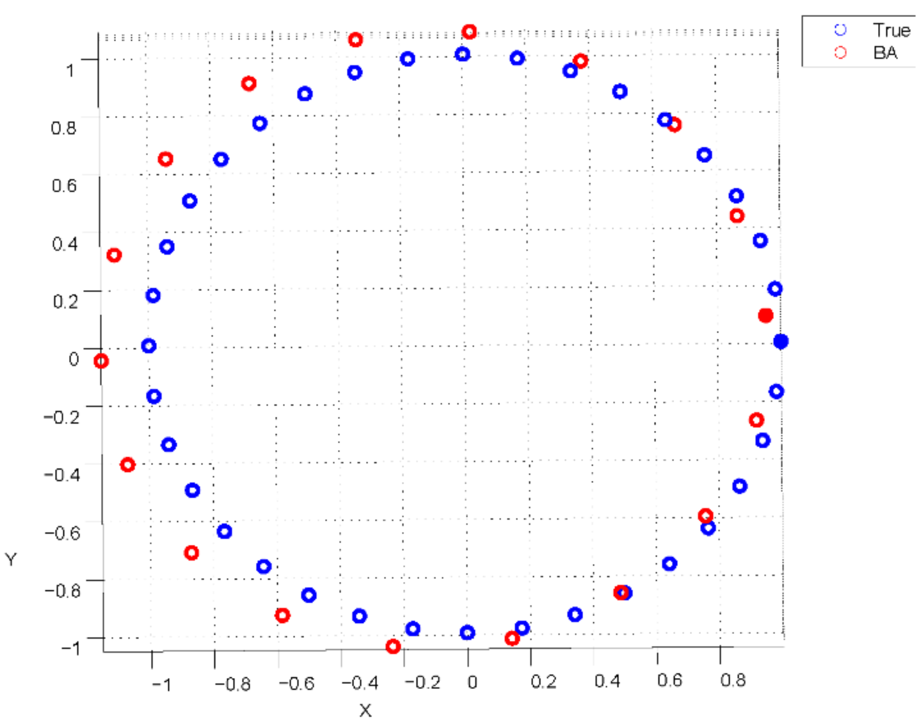
Timeline:

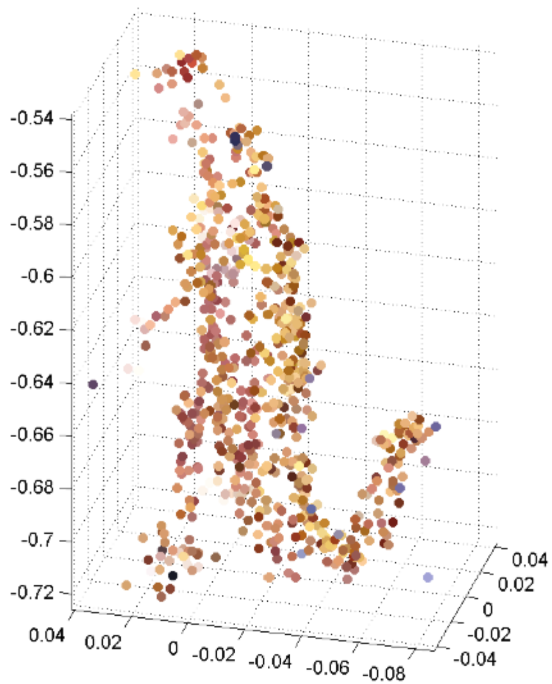
- Introductory lecture on March 31 (today)
- Design plan due April 6
- Report due May 13 (checked by guide before)
- Presentation seminar on May 17

Example: initial data


Two examples of images from the *dinosaur* sequence, with corresponding interest points

Example: result





Results from 2011 project by Bertil Grelsson and Freddie Åström

Project 2

Additional recommendations:

- Develop small modules
- Test each module before integration
- Test on data that gives predictable results
- Verify which coordinate system you are using!
- Start with CE3, it is very useful for project 2.
- Make use of supervision

Good Luck!