Structure from Motion

TSBB15, group 4

Students: Hoang Tran, Matheus Bernat, Viktor Ivarsson, Yunhee Kim

Supervisors: Pavlo Melnyk, Felix Järemo-Lawin

Main responsibilities

Initialization - Yunhee

Bundle Adjustment - Hoang

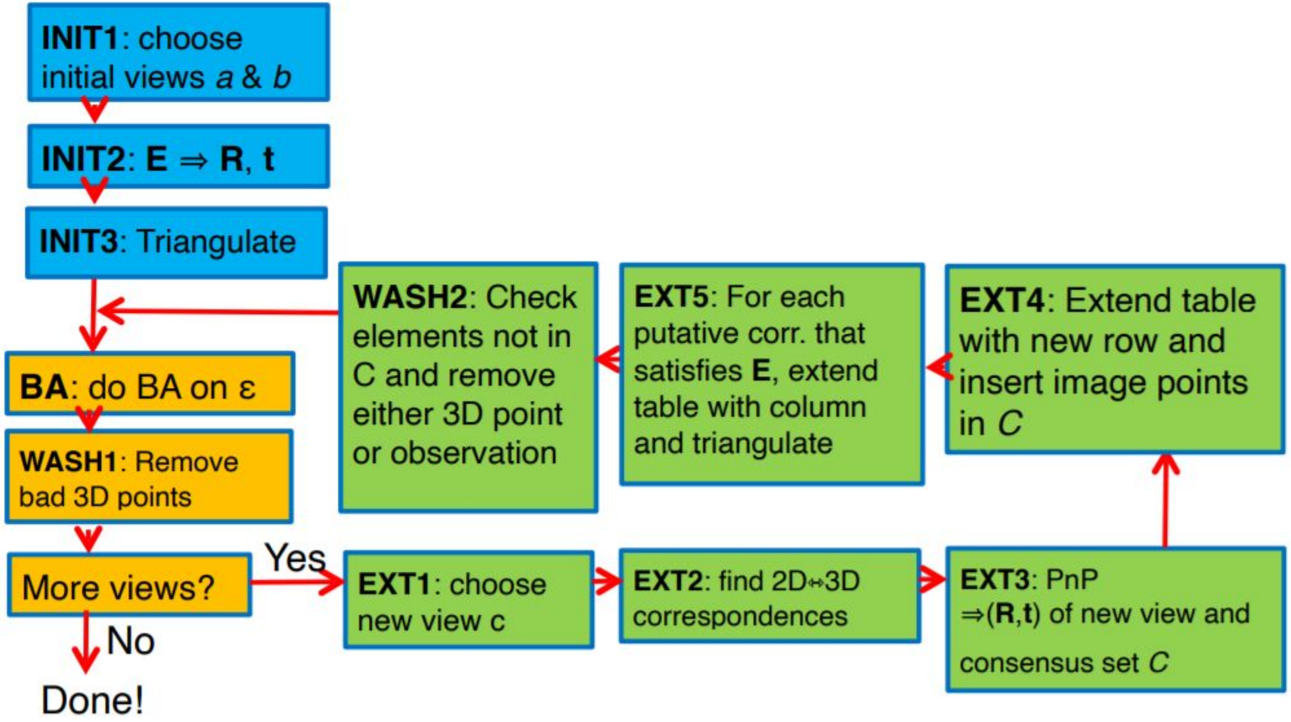
Camera pose estimation - Matheus and Viktor

Meshing - Viktor

Evaluation - Matheus and Yunhee

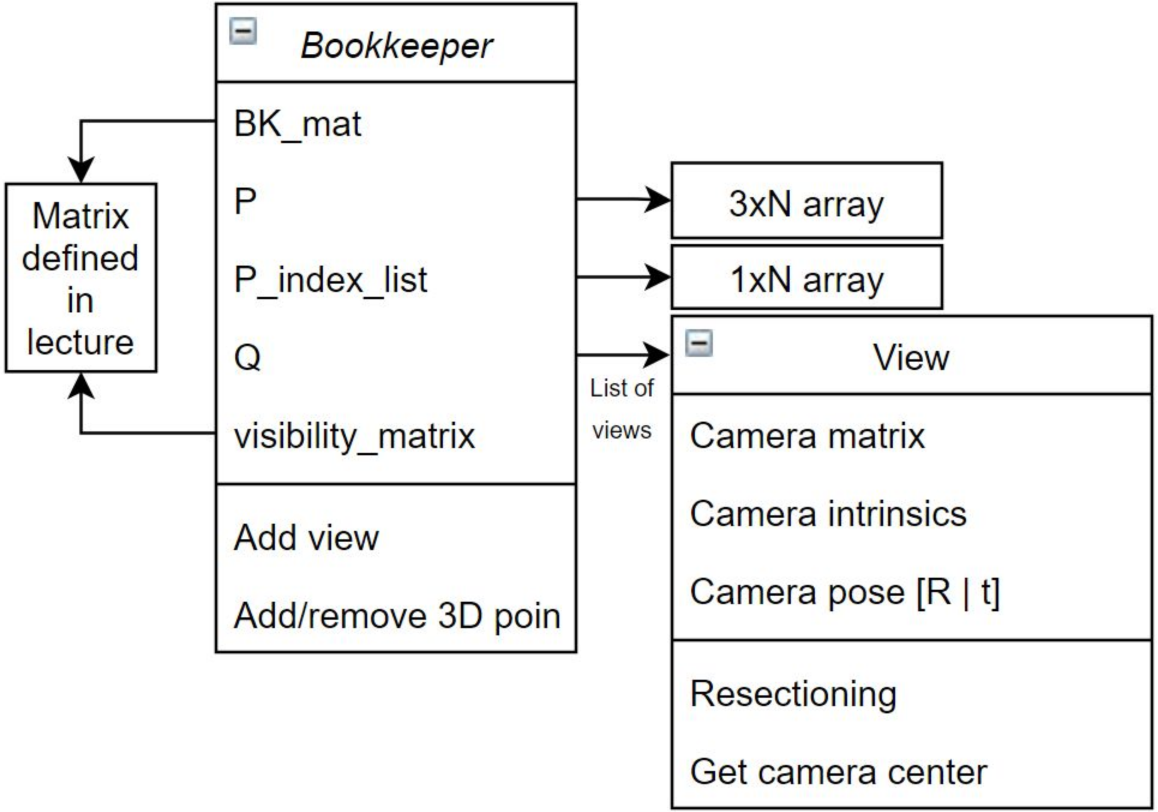
Pipeline

Same as everyone else



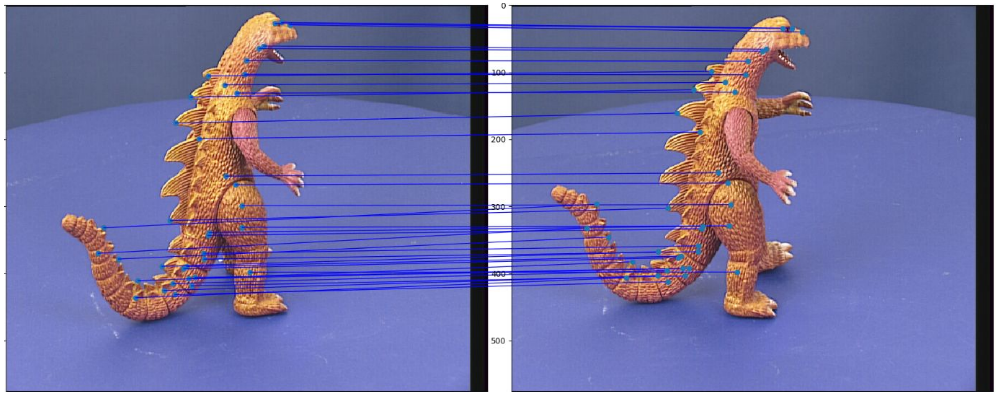
Data structure

- Considered objects for points in P
- Used P_index_list instead for indexing
- Risk of getting lost in an index jungle
- Prioritize speed



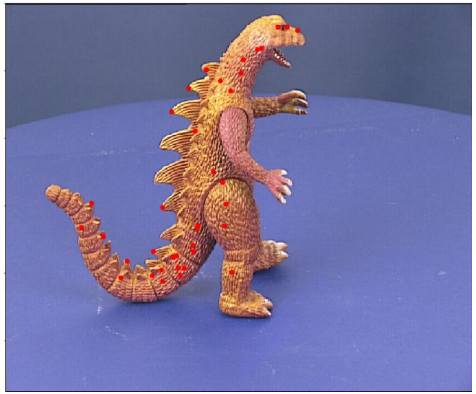
Initialization

- Hardcoded initial views - first and third view.
- Essential matrix - *findEssentialMat* from the OpenCV library.



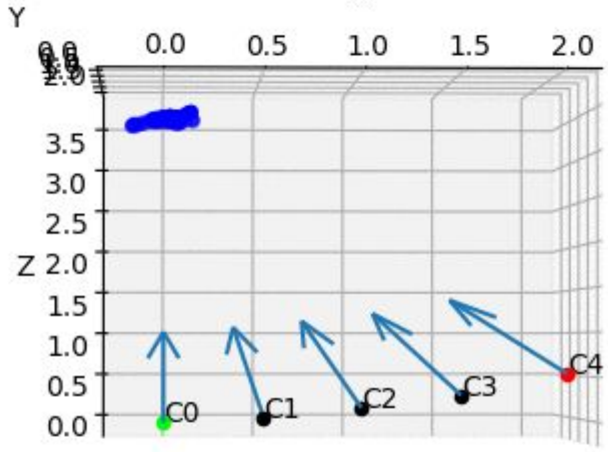
First view

Third view

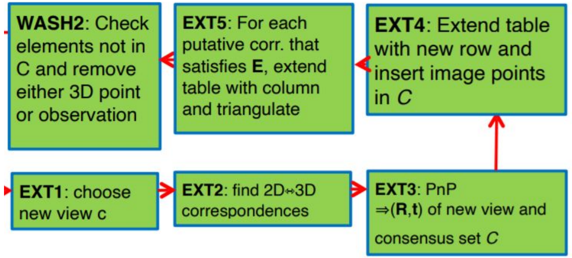


Third view with reprojected triangulated 3D points

Adding new views



- Simply followed the suggested pipeline:
  - Choose **new view** and find 2d-3d correspondences
  - Get **camera matrix** C of the new view with PnP
  - Determine **fundamental matrix** F between new view and nearby view
  - **Triangulate points** and add to list of triangulated 3D points
  - **Wash** bad triangulations
- New view is chosen as the one closest to an already used view
- OpenCV's *solvePnPRansac()* used, needs at least four 2d-3d correspondences



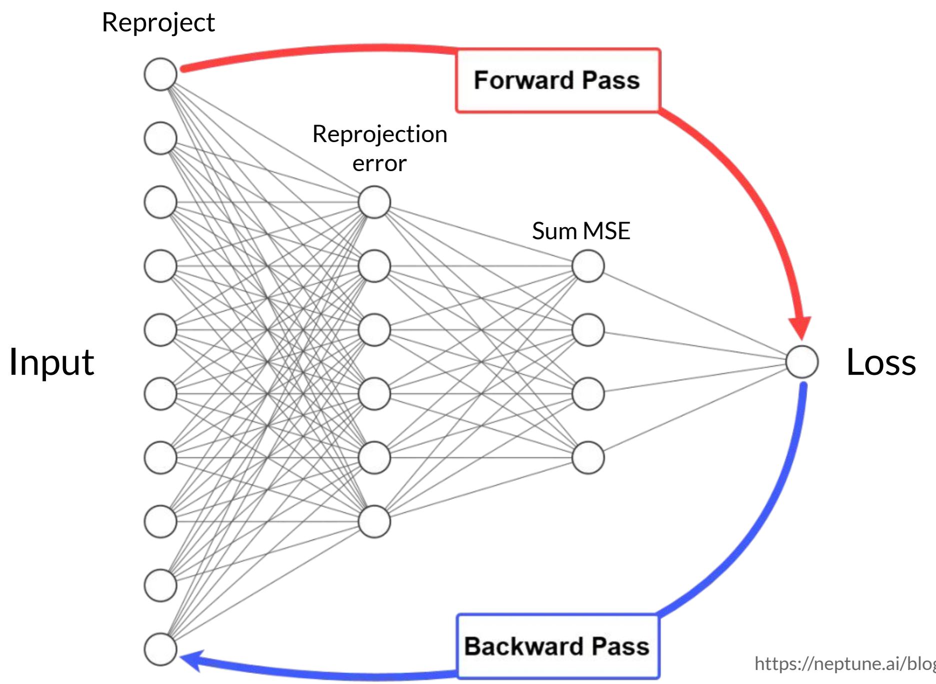
Adding new views - improvements

- Instead of choosing closest unused view, choose unused view that sees the most triangulated points (as in Schönberger et al.).
    - Note that for this dataset, where the camera moves in a circular path, the view closest to an unused view often is logically the one that sees the most triangulated points. So such an improvement isn't interesting when working with the Dino-dataset, but would be necessary in more complex datasets.
- Washing of points needs parameter tuning as the reconstruction worked just as well without it.

Bundle Adjustment

- Pytorch
  - Tool mostly used for training weights in neural networks
  - Works with Tensors
  - Calculate gradient through backpropagation
  - No need for sparsity masks etc.
- Adaptive Moment Estimation as an optimizer
  - Combines Adaptive Gradients and Root Mean Square Propagation
  - Uses randomly chosen data to make an approximation of the gradient
  - Simple, fast and efficient

Bundle Adjustment - Pytorch



https://neptune.ai/blog/backpropagation-algorithm-in-neural-networks-guide

Bundle Adjustment - Pytorch

1. Structure which parameters to optimize:
    - Make sure **rotations** stay as rotations by switching to **axis-angle representations**
    - Cast inputs to tensors
        - **Rotation** parameters: 3x(len(Q)-1) tensor
        - **Translation** parameters: 3x(len(Q)-1) tensor
        - **3D point** parameters: 3xN tensor
    - Enable gradient calculation for them
    - Send them to device (CPU in our case)
2. Calculate the loss through a forward pass:
    - Cannot use functions designed for numpy arrays to calculate. Including numpy and scipy.
    - Used PyTorch3D (Copyright © **2020 Facebook Inc**) for axis-angle and rotation matrix conversion for tensors.
    - Project points and calculate error
    - Results in a 1-element tensor with all the information for gradient estimation
3. Backpropagation and optimization step
4. Iterate step 2 and 3 desired amount of times
5. Detach gradient, send back to CPU and cast back to numpy array

Bundle Adjustment - Error calculation

- Used sum of mean squared reprojection errors
- Final error around 5.0-6.0 meaning each view had mean square error of around 0.15 pixels
- Washed points with high mean reprojection error and points that moved around too much in BA

$$\epsilon(R, t, x) = \sum_{j \in Q} \frac{\sum_{i \in P} v_{ij} d_{PP}^2(y_{ij}, K[R_j | t_j] x_i)}{N_j}$$

Bundle Adjustment - Improvements

- More parameter tuning
- Implement adaptive learning rate
- Implement ability to use with GPU

Results: Live Demo

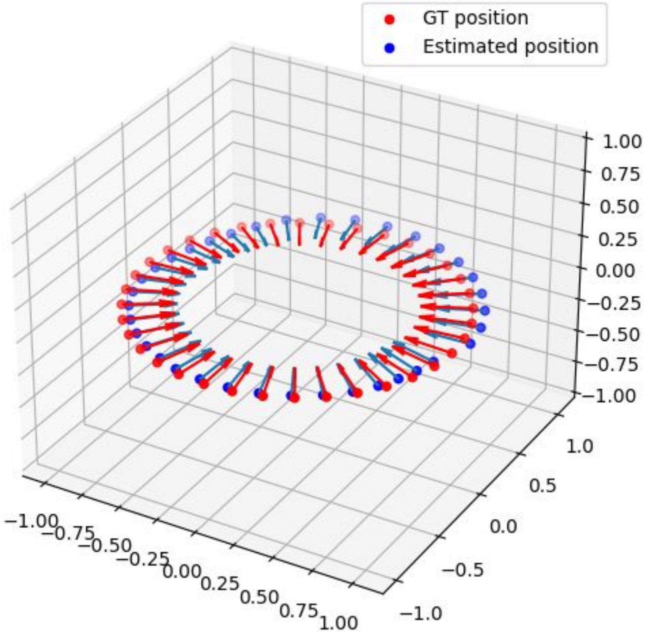
Results

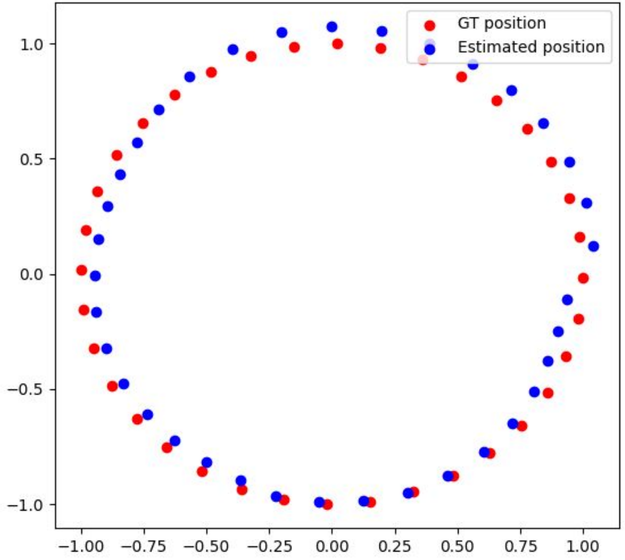
Translational MSE :

0.06297

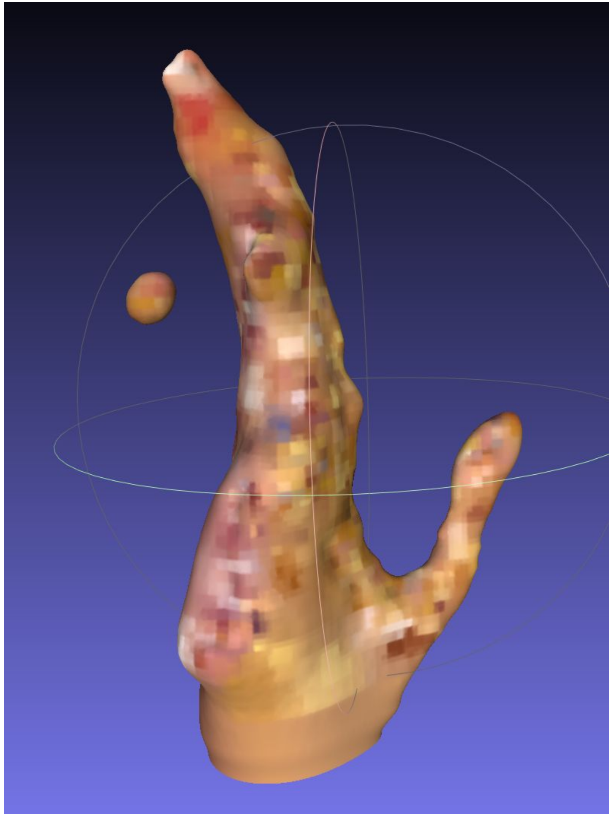
Rotational MSE:

0.06546



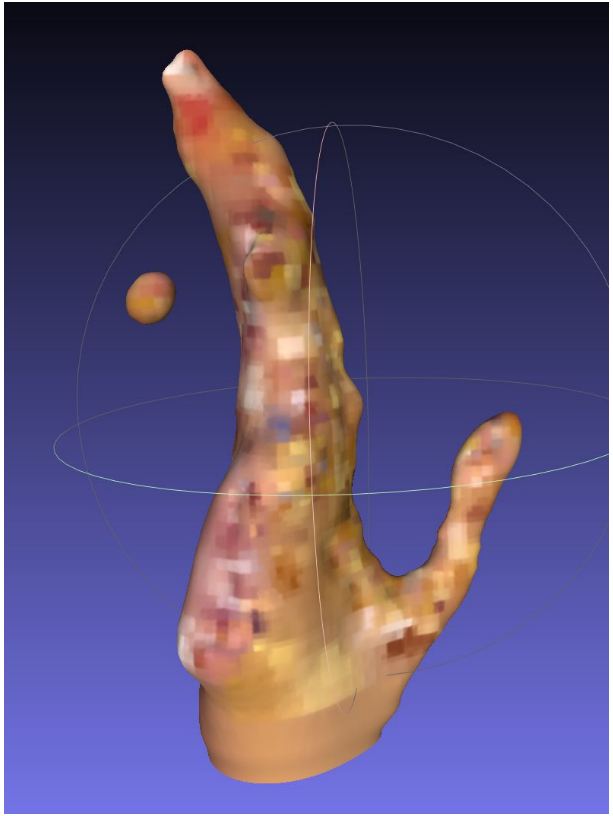


Results



Meshing

- Color estimation
- Estimate normal vectors
- Refine normal vectors
- Poisson surface reconstruction



Thanks for listening! Questions?