

Computer Vision

Course Assignment

Pedro José Pérez García, 756642
20-07-2024

Contents

- Problem overview and main objectives
 - Chosen pictures
- Old camera pose computation
 - Camera calibration
 - Keypoint detection and matching
 - Removing spurious data
 - Computing the location of the old camera
- Image differences
 - Our approach
 - Results

Main problem overview

- Take 2 or more pictures of a historical building or monument
- Obtain an additional picture of that place, must date before 1990
- Compute the pose of the camera that took the old photo
 - Position
 - Orientation
- After that, compute the differences in the pictures
 - Many possible approaches

Our chosen old picture

- Torre de Belém, Lisboa
- Taken between 1930 and 1980
- Unknown author

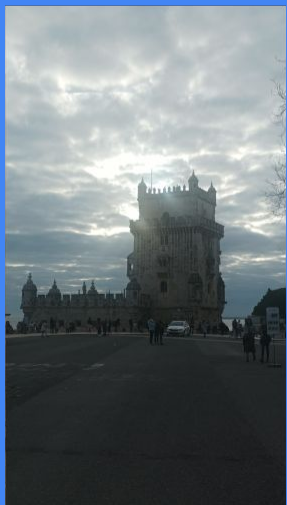
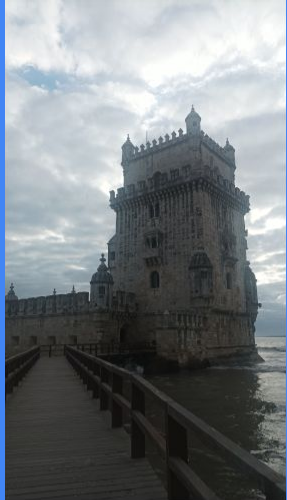
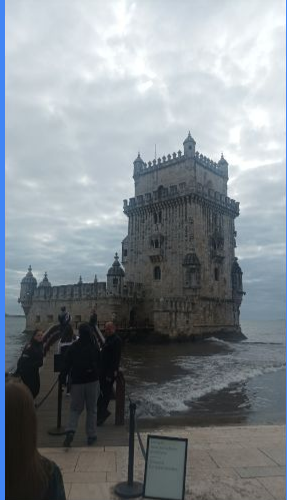


Biblioteca de Arte da Fundação Calouste Gulbenkian

Our pictures

- Torre de Belém, Lisboa
- Taken during my Erasmus stay there
 - Not purposefully taken for Computer Vision
 - Artifacts like lens flare
- Different aspect ratio





Our pictures

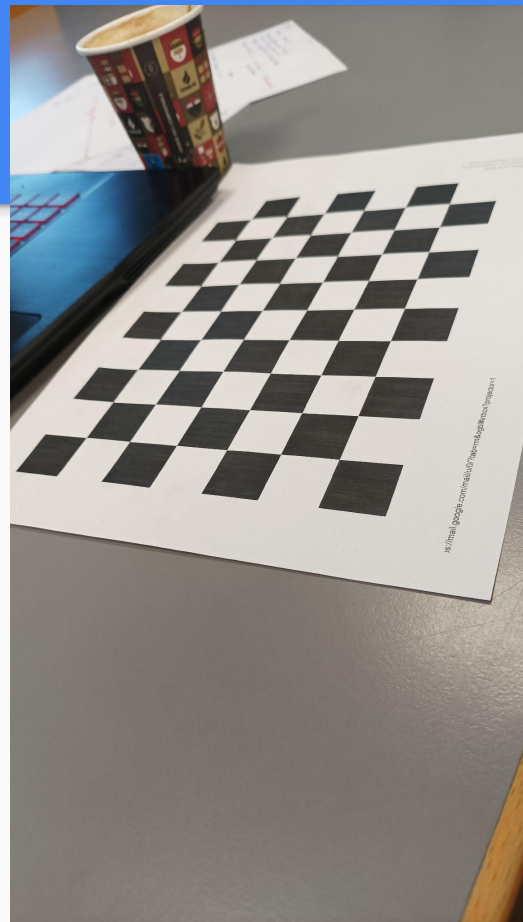
- Torre de Belém, Lisboa
- Taken during my Erasmus stay there
 - Not purposefully taken for Computer Vision
- Different aspect ratio
- 10 different pictures
 - Choose 2?
 - Try them all?



Old camera pose computation

First step: Camera calibration

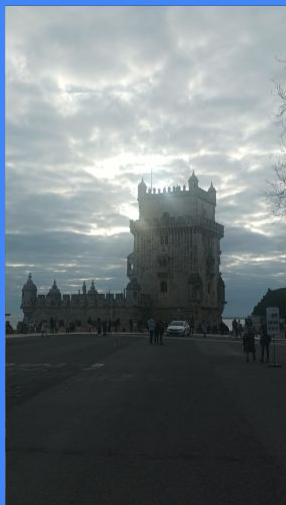
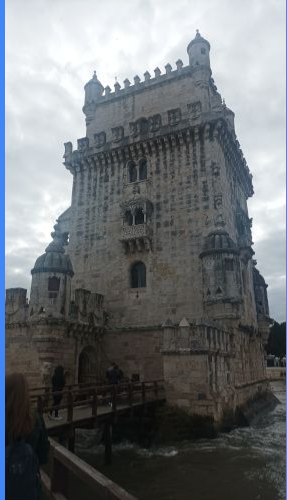
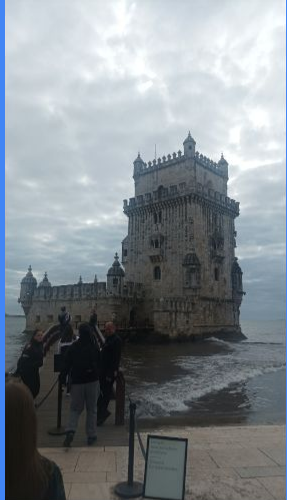
- 8 different pictures, in vertical orientation
- 10x7 checkerboard pattern
- Taken in different orientations
- We follow the example calibration code
 - `cv.findChessboardCorners`
 - `cv.calibrateCamera`
- Store the obtained K_c in text files
 - We also store the radial distortion coefficients



Next step: Keypoint detection and matching

- We have 10 pictures, but we don't want to choose 2 (yet)
 - Bruteforce them!
- We match every possible pair among our n new pictures
 - Any number (quadratic cost, be careful!)
- The process is controlled by using config files
 - That's why we save in .txt files after every step
 - We can execute steps independently!
 - Faster debugging of specific steps

```
[stage2]
matches_file_folder =      ../new_pictures
matches_file_name =       matches.txt
new_images_folder =       ../new_pictures
old_images_folder =       ../old_pictures
old_image_name =          torre_de_belem_1930_1980.jpg
matches_output_folder =   ../superglue_output/
```



Next step: Keypoint detection and matching

- How do we do it?: SuperGlue
- We also considered LightGlue
 - Newer version, real-time performance
 - We stuck with SuperGlue for familiarity
- `match_pairs.py --superglue outdoor --max_keypoints 2048 --resize -1 -1`



SuperGlue

Keypoints: 2048:1528

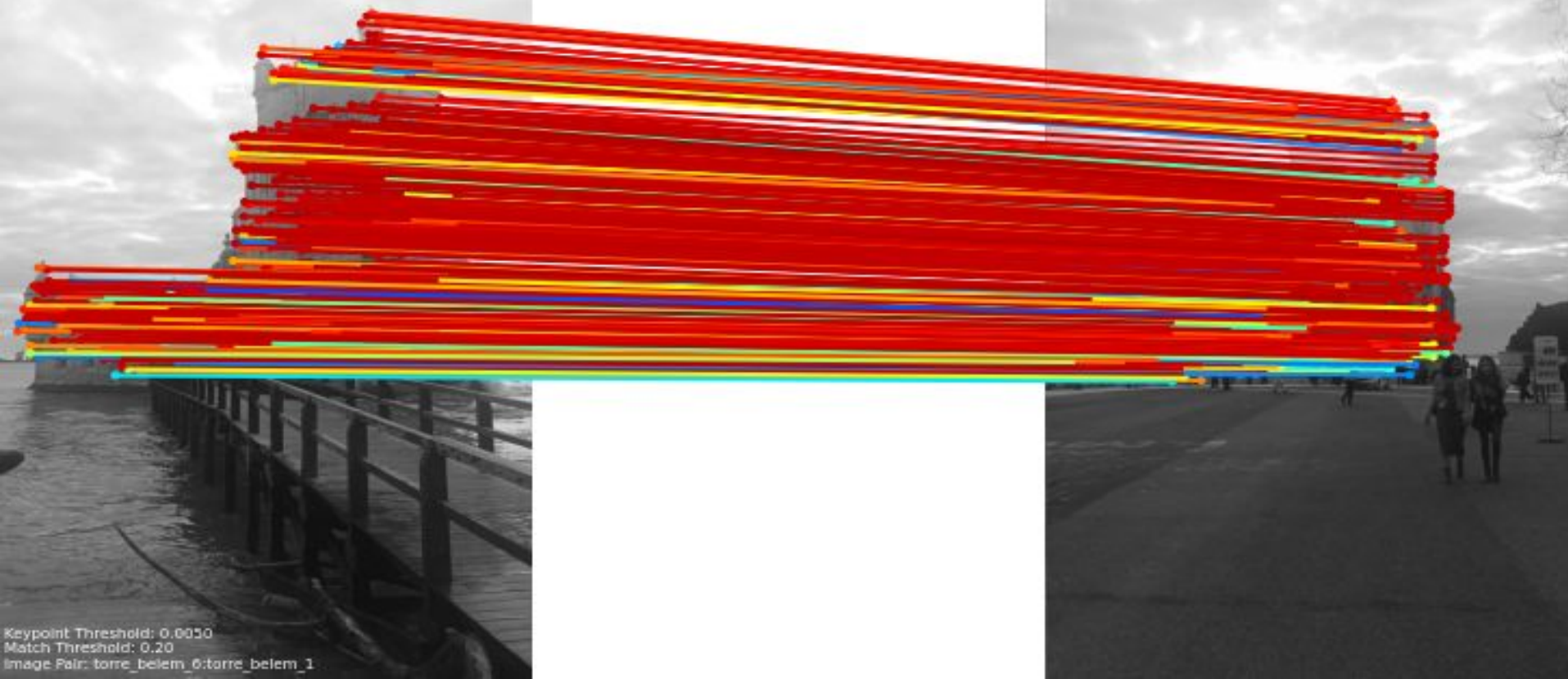
Matches: 355



SuperGlue

Keypoints: 2048:2048

Matches: 474



Keypoint Threshold: 0.0050

Match Threshold: 0.20

Image Pair: torre_belem_0:torre_belem_1

Third step: Remove Spurious Data

- We want to get the best 2 images from the 10 we have
- We also want to get rid of spurious inliers
- RANSAC (RANDOM SAMPLING CONSENSUS)
 - For a certain number of iterations:
 - Sample 8 points at random
 - Compute Fundamental matrix F with 8-point algorithm
 - Use F to compute distance to epipolar line (transfer error)
 - For the rest of matches, if distance < threshold, add to possible inliers
 - Get the biggest set of possible inliers as our final inliers

Given a match **in pixels** $\{(x_0, y_0, w_0), (x_1, y_1, w_1)\}, x_1^T F x_0 = 0$

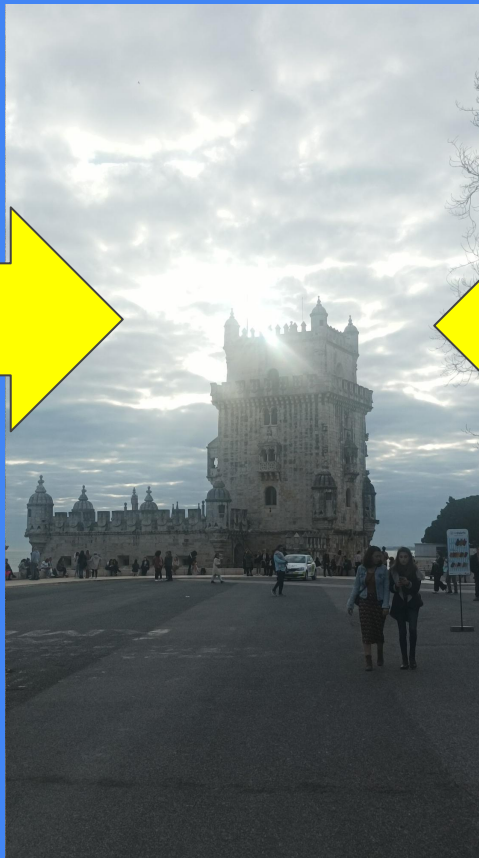
$$\begin{bmatrix} x_0 x_1 & y_0 x_1 & w_0 x_1 & x_0 y_1 & y_0 y_1 & w_0 y_1 & x_0 w_1 & y_0 w_1 & w_0 w_1 \end{bmatrix} \begin{bmatrix} f_{11} \\ f_{12} \\ f_{13} \\ f_{21} \\ f_{22} \\ f_{23} \\ f_{31} \\ f_{32} \\ f_{33} \end{bmatrix} = 0$$

Third step: Remove Spurious Data

- RANSAC gives us the inliers for one image pair **im1-im2**
- We also compute the inliers for **im1-imOLD**
- Get the common inliers:
 - Those that have the “same” points in im1 for their matches
 - Tolerance threshold of 2 pixels, works well with up to 5 pixels
- Search for the best im1-im2-imOLD combination
 - If for any im1-im2, `obtained_inliers < current_best inliers`, skip the rest of computations



400 inliers



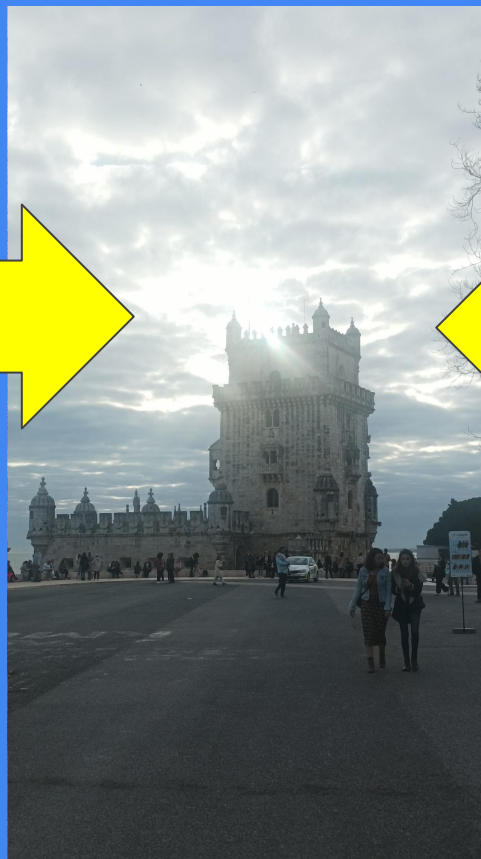
Only 100 in common

300 inliers





300 inliers



175 in common: best!

300 inliers



Image 1, all the inliers with OLD image

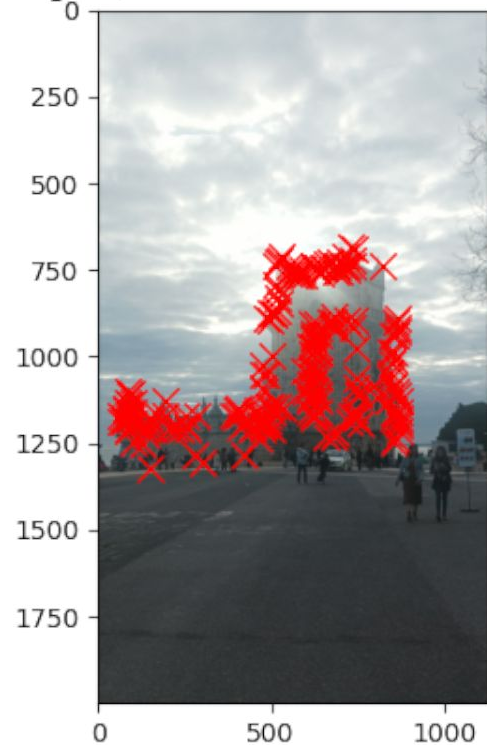


Image 1, all the inliers with image 2

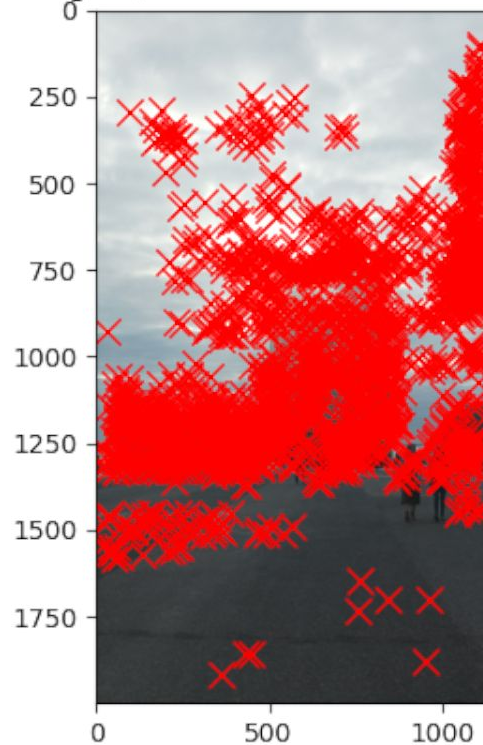
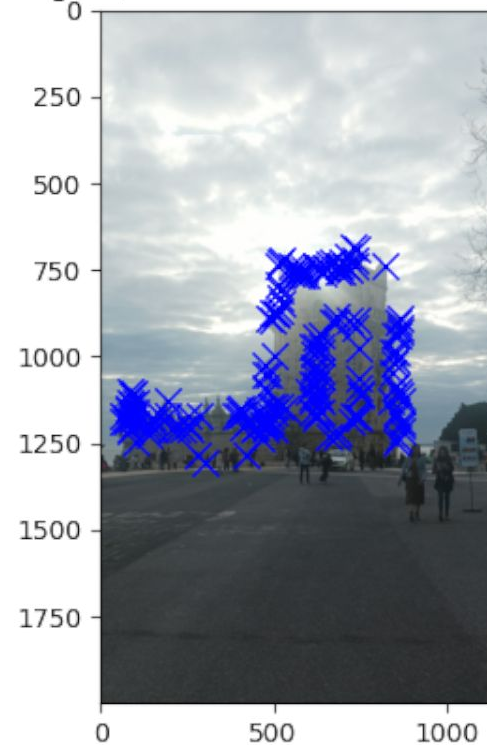


Image 1, all the inliers with OLD image



Fourth step: Camera pose computation

- Load our saved inliers and K_c
- Compute fundamental matrix F from inliers
- Essential matrix $E = K_c^T F K_c$
- With E , perform 3D triangulation
 - Chirality

3. Motion from E

$$[U, \Sigma, V] = \text{svd}(E)$$

$$E \cong \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 0 \end{bmatrix} V^T$$

$$E \cong [u_1 \quad u_2 \quad u_3] \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 0 \end{bmatrix} \begin{bmatrix} v_1^T \\ v_2^T \\ v_3^T \end{bmatrix}$$

4 solutions

$$R_{+90}, t; R_{+90}, -t; R_{-90}, t; R_{-90}, -t$$

Where:

$$R_{+90} = \pm U W V^T \quad R_{-90} = \pm U W^T V^T$$

Sign such as $|R_{+90}| > 0, |R_{-90}| > 0$

$$t = u_3 \quad W = \begin{bmatrix} 0 & -1 & 0 \\ 1 & 0 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

11.3.1 Eight, seven, and five-point algorithms. Recovering t and R . Szeliski 2022, 2nd Edition.

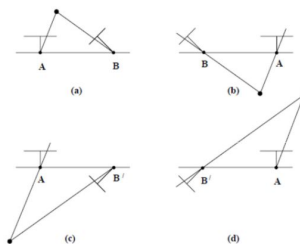


Fig. 9.12. The four possible solutions for calibrated reconstruction from E . Between the left and right slides there is a baseline reversal. Between the top and bottom rows camera B rotates 180° about the baseline. Note, only in (a) is the reconstructed point in front of both cameras.

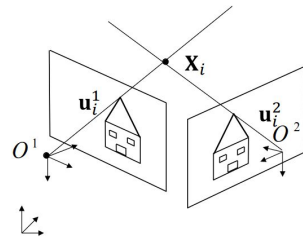
4 motion solutions yield the same E , only one of them yields 3D points in front of **both** cameras (chirality)

Fourth step: Camera pose computation

- Now we have X_{3D}
- Also t_{c2c1} , and R_{c2c1} (camera 2 relative to camera 1)
- Bundle Adjustment for 2 views
 - Feed our newly calculated data as initial guess
 - Rotation encoded
 - Translation encoded too, as spherical coords
 - Our residual is the reprojection error

2. Triangulation

If a point is observed in two calibrated projective cameras, with known camera poses, its 3D pose can be recovered

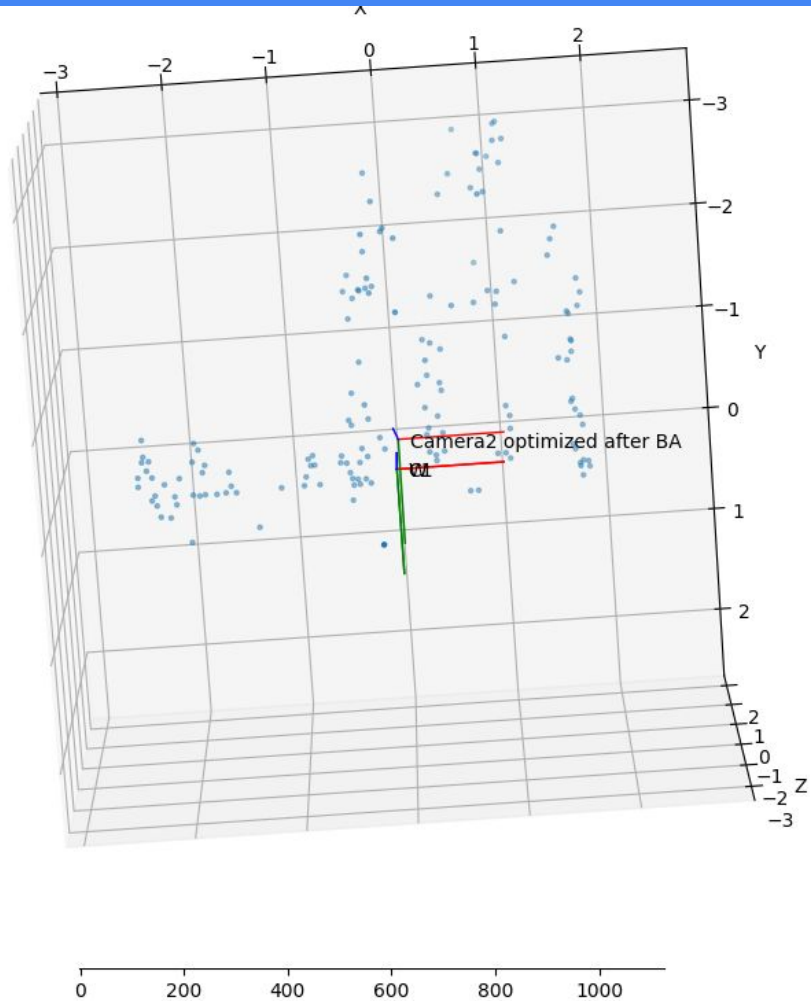


$u_i^1, u_i^2, \theta_{int}^1, \theta_{ext}^1, \theta_{int}^2, \theta_{ext}^2$

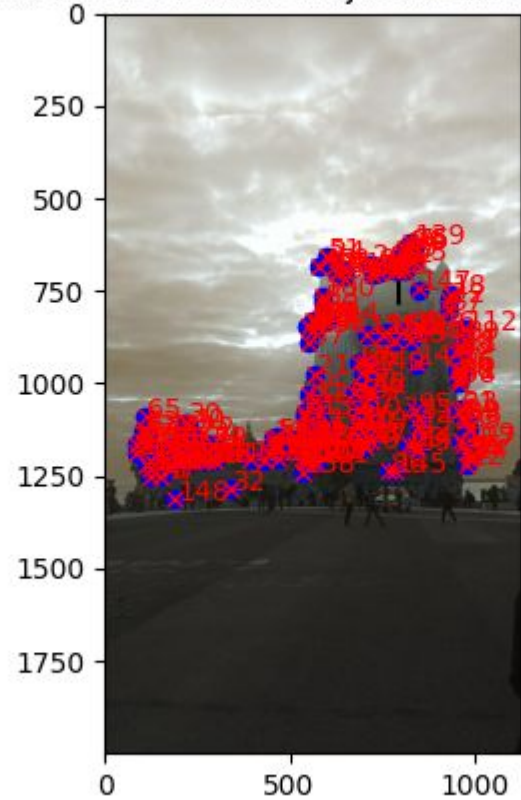


X_i

Intersection of two incoming rays



Residuals for Bundle Adjustment in Image2



Fourth step: Camera pose computation

- Why did we use only common inliers among our 3 images?
 - **DLT: Direct Linear Transformation**
 - With ≥ 6 2D-3D matches, we can compute P , projection matrix
 - If our 2D matches are common we can use the triangulated 3D points X_{3D} !

MRGCV Computer Vision 15

5. DLT camera matrix P

We can consider 2D-3D matches including points at infinity:

$$\{x_i \ y_i \ w_i \ X_i \ Y_i \ Z_i \ W_i\}$$

If 6 or more 2D-3D matches available, P can be computed.

$$\begin{bmatrix} -w_i X_i & -w_i Y_i & -w_i Z_i & -w_i W_i & 0 & 0 & 0 & 0 & x_i X_i & x_i Y_i & x_i Z_i & x_i W_i \\ 0 & 0 & 0 & 0 & -w_i X_i & -w_i Y_i & -w_i Z_i & -w_i W_i & y_i X_i & y_i Y_i & y_i Z_i & y_i W_i \end{bmatrix} \begin{bmatrix} p_{00} \\ p_{01} \\ p_{02} \\ p_{03} \\ p_{10} \\ p_{11} \\ p_{12} \\ p_{13} \\ p_{20} \\ p_{21} \\ p_{22} \\ p_{23} \end{bmatrix}$$

Assembling the $2n$ equations we obtain the matrix $A_{2n \times 12}$

$$[U, S, V] = \text{svd}(A)$$

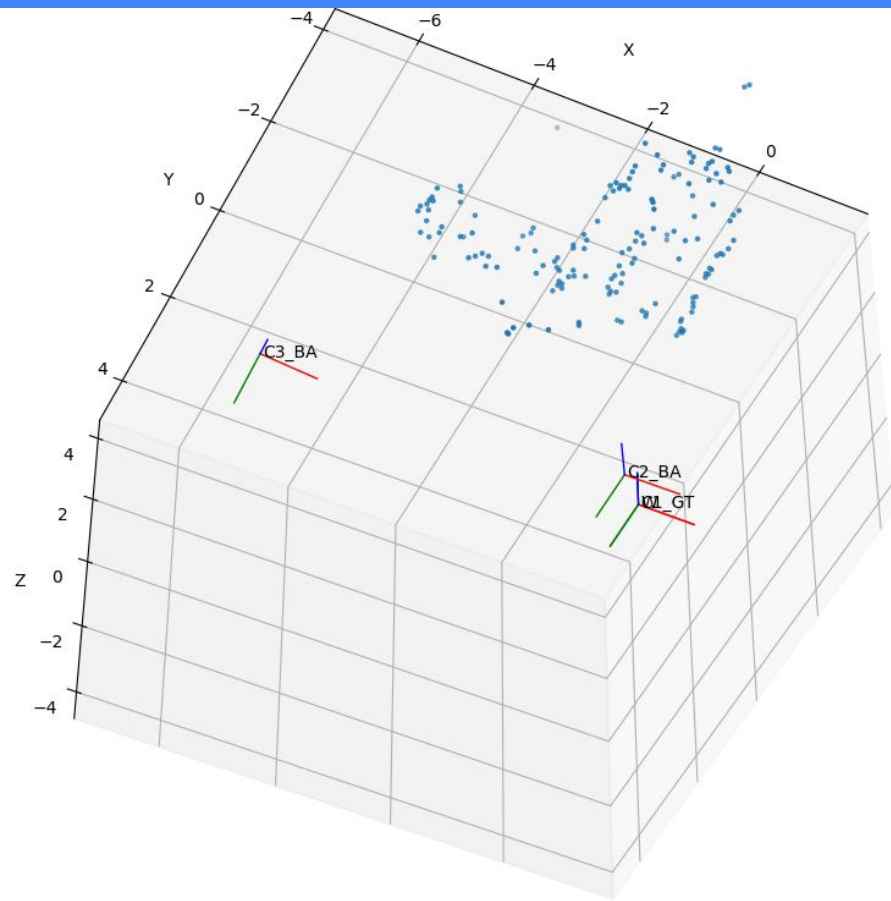
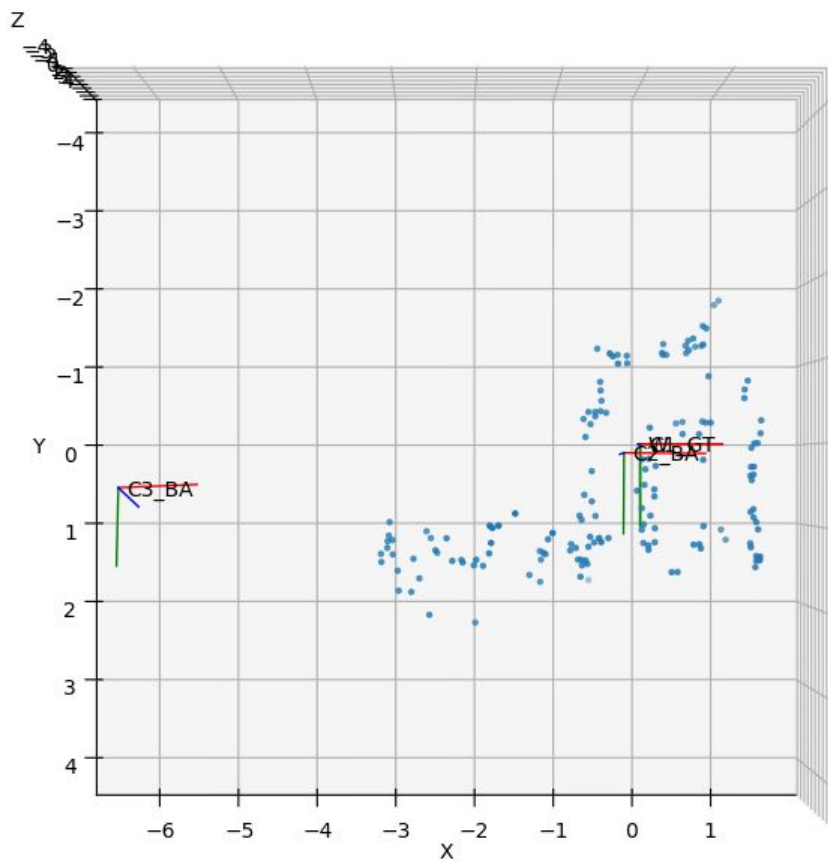
The solution is the 12th column of V

Fourth step: Camera pose computation

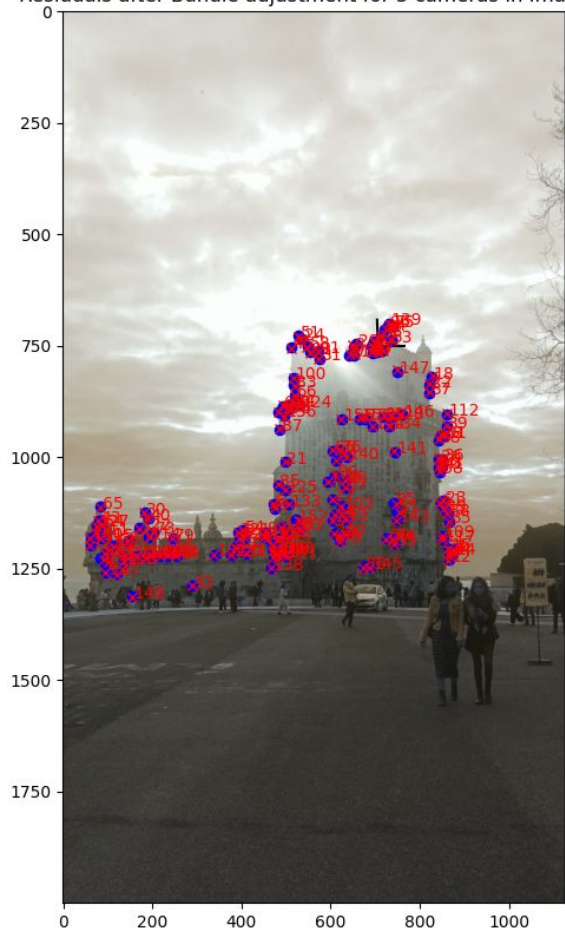
- We get a first estimation for P_3 , projection matrix for camera 3
 - Decompose it
 - `cv.decomposeProjectionMatrix` didn't work for us
 - <https://stackoverflow.com/questions/55814640/decomposeprojectionmatrix-gives-unexpected-result> fixed the issue
 - We obtain a K_{c3} matrix and ignore R_{c3c1} , t_{c3c1}

Fourth step: Camera pose computation

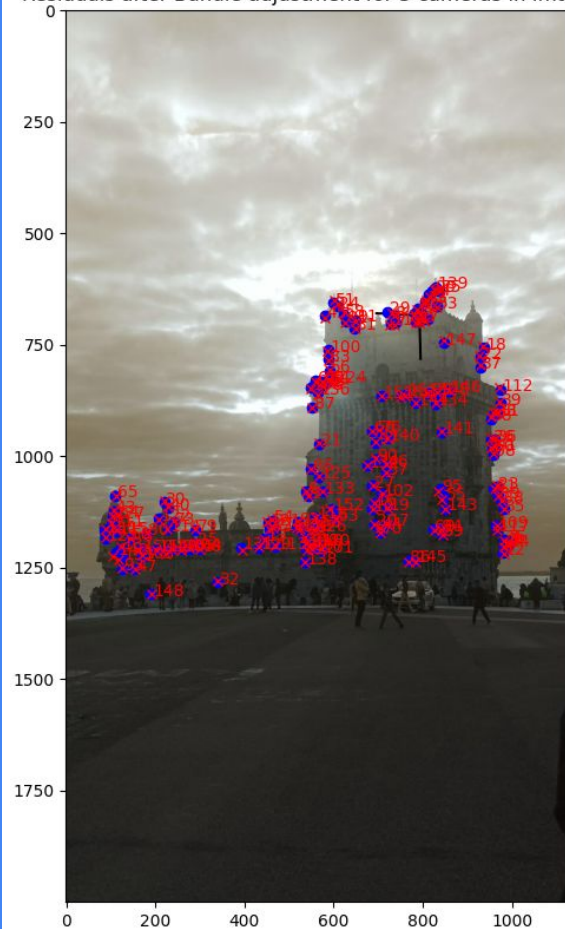
- Feed K_{c3} into a PnP algorithm to refine our rotation and translation
 - This rotation and translation will be our initial guess for the final optimization
- 3 view Bundle Adjustment
 - Same as before, but optimize one more camera
 - Updated residual function to support N cameras
 - Final result!



Residuals after Bundle adjustment for 3 cameras in Image 1



Residuals after Bundle adjustment for 3 cameras in Image 2



Residuals after Bundle adjustment for 3 cameras in Image 3



Image differences

Image differences: our approach

- Compute homography
 - OpenCV's method instead of ours
 - We wanted to go for accuracy
- Warp the new image into old



Image differences: our approach

- Compute homography
 - OpenCV's method instead of ours
 - We wanted to go for accuracy
- Warp the new image into old
- Crop non-overlapping area
- Apply grayscale
- Equalize histograms



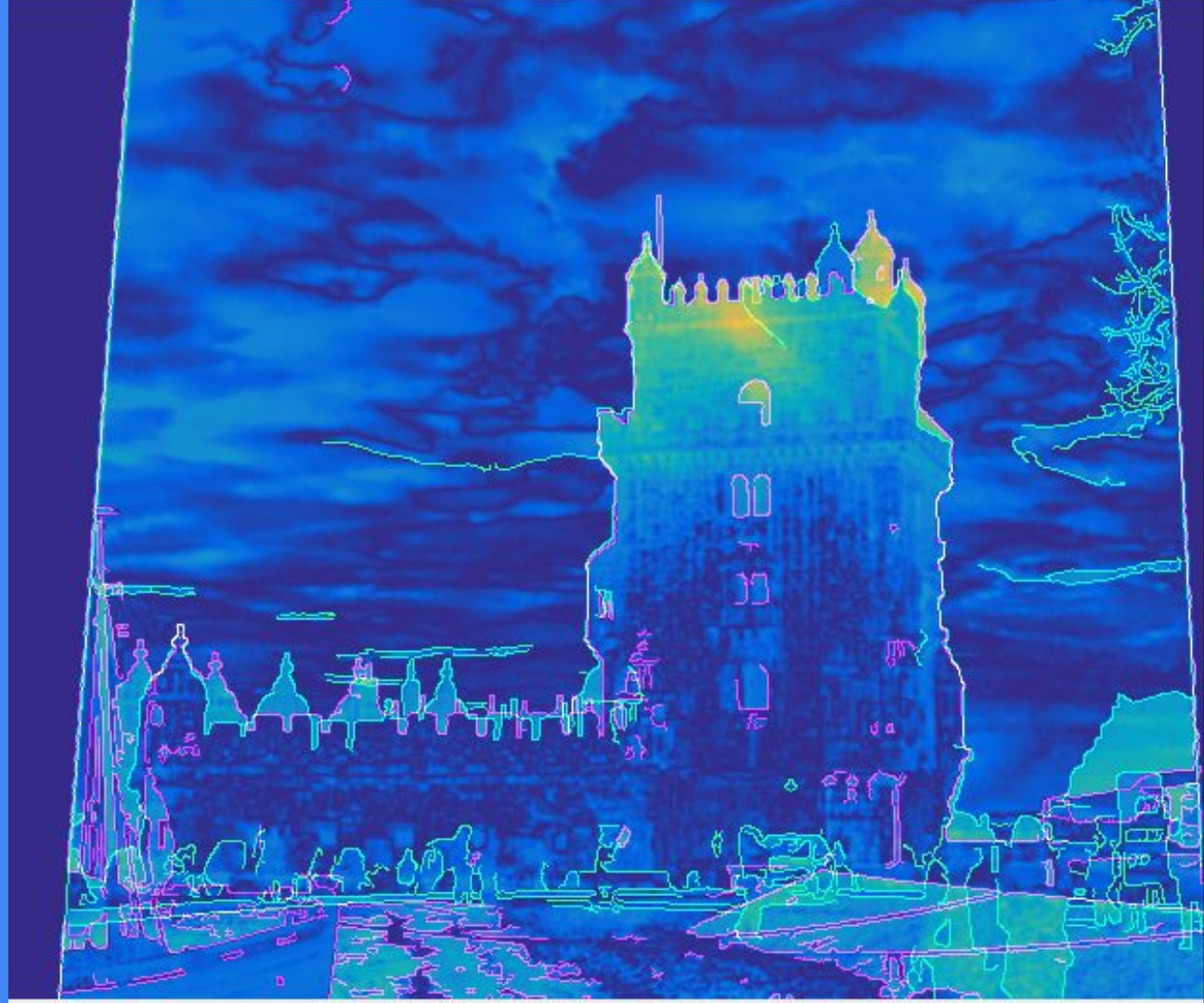
Image differences: our approach

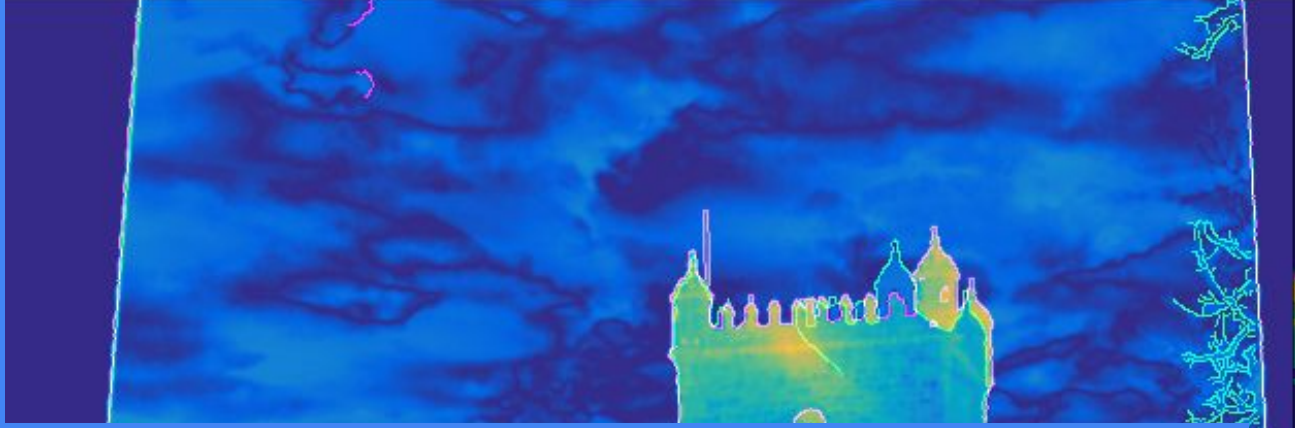
- Compute homography
 - OpenCV's method instead of ours
 - We wanted to go for accuracy
- Warp the new image into old
- Crop non-overlapping area
- Apply grayscale
- Equalize histograms
- Apply small Gaussian blur
- Apply absdiff



Image differences: our approach

- Apply a colour gradient
- Overlay Canny edges
 - 1 colour for each image
- We experimented with thresholding, with and without Otsu
 - Bit operations like and, xor, etc..
 - Did work sometimes but only for specific images
 - Too ad-hoc to be considered a solution





Thank you for your attention

Any questions?

