Computer Vision Course Assignment

Pedro José Pérez García, 756642

June 20, 2024

1 Introduction

For the final project of our Computer Vision course within the Master's Degree in Robotics, Graphics and Computer Vision (MRGCV), our main task required to take a photograph that was captured before the year 1990, and compare it with a set of new images taken recently by ourselves, so that we could have access to the camera that took them. With those images, we are supposed to obtain a decent approximation of the tridimensional geometry of the scene, including the position of the camera that took the original photograph. After that, we should conduct a study on the main visual differences between one of the recent images and the original, older one. We have successfully achieved both goals by applying the knowledge we gained during the course, which we used to create an almost fully-automated Python program that handled most of the work.

We decided to use a set of photographs taken last year in Lisbon, more concretely, in the Torre de Belém monument. We found this appropiate, since the images were not taken purposefully for this project, and thus, no special measures were taken, so if our approach works for regular images taken in the wild, it will work even better for images that have been prepared previously. Our old image dates at some point between 1930 and 1980, according to the archive we found it in. There's also the additional challenge presented by having different layouts, since the old image is horizontal and our pictures were taken in portrait mode, holding the phone vertically.

Figure 1: Our chosen old photograph, which dates from an unknown point between 1930 and 1980 in Lisbon.

Figure 2: All the pictures we are considering for computing the location of the camera that took the old picture. Only two will be chosen for the final steps. Second and fifth images in the bottom row usually produced good results together, as well as bottom right with top right.

2 Overview of our solution

2.1 Computing camera localization

We have aimed to implement a general solution for the old camera location problem. This means that our algorithm can work with an unspecified number of n_{photos} photographs and still return a valid solution. We have also modularized the process, splitting it in 4 independent steps, where the solutions for each intermediate stage are stored and can be recovered later, removing the need to rerun previous steps:

- **Stage 1:** Obtaining the calibration matrix K_{c12} for the camera that took our "new" pictures, $im1$ and $im2$.
- **Stage 2:** Matching all our n_{photos} images to get the pair with most chances of producing a good reconstruction.
- **Stage 3:** Applying RANSAC to get the inlier points among the 3 images, our best pair of 2 images and the old one.
- **Stage 4:** The actual reconstruction step for the position of the old camera, obtained via Bundle Adjustment. Our methods for obtaining the initial guesses are discussed in the following paragraphs.

2.1.1 Camera calibration

For the first step, we simply take several pictures of a checkerboard pattern, we run the SIFT (Lowe 1999) detector on them and later we apply the *cv.calibrateCamera* function. We save the resulting calibration matrices and the computed radial distortion coefficients in text files, even though we don't use the latter, since we tested the full camera pose computation program with both distorted and undistorted images, and undistorted images performed slightly worse.

2.1.2 Image matching with SuperGlue

For our second step we compute matches among all the possible image pairings that can be produced with our n_{photos} images (10 in our case). We decided to use SuperGlue for that effect (P. Sarlin et al. 2019), a neural approach for image matching that works extremely well in both indoor and outdoor scenes. We considered using its updated, more capable version, LightGlue (Lindenberger, P.-E. Sarlin, and Pollefeys 2023), but in some cases it produced less matches and we already had some familiarity with SuperGlue, so we stuck with that. It is also worth mentioning that in order for SuperGlue to work properly, we had to set its running parameters to `resize -1 -1` to aovid weird image size and orientation issues, and we also set the `--superglue outdoor`, since our images were taken in the outside, assuming that this option would use slightly different weights tuned for outdoor scenes.

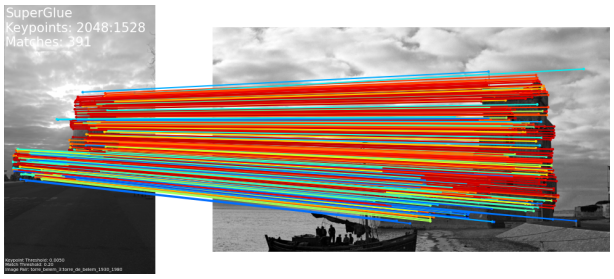

Figure 3: Matches computed by SuperGlue for one of our images and the old picture.

2.1.3 Filtering inlier matches with RANSAC

The third step required loading the previously computed matches, which we later passed through a RANSAC (RANdom SAmpling Consensus, Fischler and Bolles 1981) algorithm we ourselves implemented (we modified our implementation from past lab sessions) in a manner that allows changing its parameters from a configuration *.ini* file, the same as with most of the steps we mention here.

With this, we compute the inlier matches among every possible pair of images, and then with our old image, to get the best images $im1$ and $im2$ such that the number of common inlier points among $im1$, $im2$ and im_{old} is maximal. We consider two inlier matches to be common between 2 image pairs if their 2D points in one of the images (in this case, the points from both inliers in $im1$) are lower than a threshold euclidean distance that we can modify. Our default value for this threshold is quite restrictive for high resolution images, 2.0 pixels.

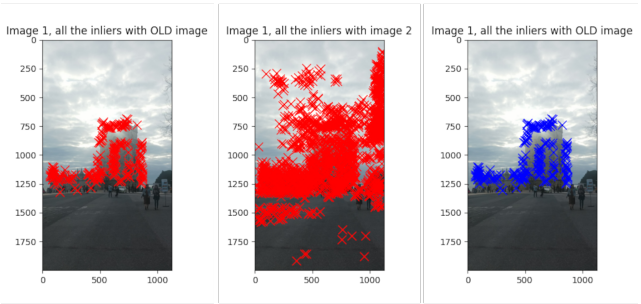

Figure 4: Best set of inliers obtained after RANSAC. On the left, inliers between new image and old image, drawn over the new one. In the middle, inlier matches between the two new images, drawn over the second image. On the right, inliers matches that appear both in image 1 and image 2, and in image 1 and the old image, drawn in blue on top of the first "new" image.

Once we identify our pair of images, we save their paths, as well as the common inliers data in a text file that will be read by the next step once it gets executed.

2.1.4 Bundle Adjustment for camera pose estimation

The fourth and last step of the process is the longest one. It performs the actual reconstruction of the old camera position and rotation, using our two chosen images, which produced the most inlier matches among all combinations, and the old photograph. First, we need to compute the 3x3 fundamental matrix that relates the points in our pair of new images. To that effect, we apply the normalized 8 point algorithm (Hartley 1997) by performing a SVD decomposition on a matrix conformed by our matches, expressed in pixels according to equation 1:

$$x_1^T F x_0 = 0 \tag{1}$$

Where:

$$x_1^T x_o = \begin{bmatrix} x_o x_1 \\ y_0 x_1 \\ w_0 x_1 \\ x_0 y_1 \\ y_0 y_1 \\ w_0 y_1 \\ x_0 w_1 \\ y_0 w_1 \\ w_0 w_1 \end{bmatrix}^T , F = \begin{bmatrix} f_{11} \\ f_{12} \\ f_{13} \\ f_{21} \\ f_{22} \\ f_{23} \\ f_{31} \\ f_{32} \\ f_{33} \end{bmatrix} \tag{2}$$

After the SVD decomposition, we obtain three matrices, U, Σ and V. The result we're after is the last

column of the transposed V matrix. After that we need to enforce rank 2, as in equation 3. Note that if the previous stage of execution didn't return at least 8 inlier matches, the position of the old camera cannot be computed accurately.

$$F_{rank2} = U \begin{bmatrix} \sigma_1 & 0 & 0 \\ 0 & \sigma_2 & 0 \\ 0 & 0 & 0 \end{bmatrix} V^T \tag{3}$$

With:

$$U, \Sigma, V = svd(F), \Sigma = \begin{bmatrix} \sigma_1 & 0 & 0 \\ 0 & \sigma_2 & 0 \\ 0 & 0 & \sigma_3 \end{bmatrix} \tag{4}$$

Once we have our fundamental matrix F, we can compute the 3x3 essential matrix E, which describes the geometric relationship between two cameras that compose a system. In other words, we can use this matrix to compute the 3D structure of the scene, i.e the relative position for the points in our inlier matches, and the relative position for both our camera poses, C_1 and C_2. This matrix can be easily computed from the camera calibration matrix K_{c12} and the fundamental matrix F, as we explain in equation 5.

$$E = K_{c12}^T F K_{c12} \tag{5}$$

With it, we can perform a triangulation to obtain the 3D position of the points in our matches, as well as the translation and rotation of camera 2 with respect to camera 1, t_{c2c1} and R_{c2c1}. Part of this triangulation involves retrieving the correct translation and rotation, since the same E can be obtained by four different combinations of translations and rotations. This is what we know as chirality. To solve it, we have to check which of the four possible solutions has the biggest amount of points in front of both cameras, as we illustrate in Algorithm 1.

The 3D position of the points (with no scale), along with this rotation and translation, will be our initial guess, which we refine by least squares optimization, with the Levenberg-Marquardt algorithm (Levenberg 1944) in a Bundle Adjustment process. For this optimization, we aim to minimize the reprojection error of the points seen from one camera, when reprojected into the other one, which is the same residual function as we already used in past lab sessions. This optimization allows us to get a better camera translation and rotation for camera 2, t_{c2c1} and R_{c2c1} (we are assuming camera 1 to be in the origin of the world and with no rotation, so that everything is defined with respect to it). The result can be seen in Figure 5

In order to compute the pose for our third, old camera, we need to provide an estimation for its position and rotation. Since this algorithm is only getting points that are common among the 3 cameras, we can establish a correspondence between the 2D points and our triangulated, optimized ones. This is exactly what the DLT algorithm (Sutherland 1974) needs in order to compute

Data: Essential matrix E, Projection matrix for images 1 and 2 K_{c12}, 2D match points in image 1 x_1 and image 2 x_2
Result: 3D triangulated points X_{3D}, translation t_{c2c1}, rotation R_{c2c1} for camera 2 wrt. camera 1
U, Σ, V= svd(E);
$W = \begin{bmatrix} 0 & -1 & 0 \\ 1 & 0 & 0 \\ 0 & 0 & 1 \end{bmatrix} V^T$;
$R_{+90} = UWV$;
$R_{-90} = UW^TV$;
if $det(R_{+90}) < 0$ **then**
  swap R_{+90} and R_{-90};
end
t = last column of U;
$P1$ = identityProjectionMatrix(K_{c12});
$P2_1, P2_2, P2_3, P2_4$=projMats(t,$-t$,R_{+90}, R_{-90});
X_1, X_2, X_3, X_4=
triangulate3D(x_1,x_2,$P1$,$P2_1$,$P2_2$,$P2_3$,$P2_4$);
$X_{1c2}, X_{2c2}, X_{3c2}, X_{4c2}$=$X_1, X_2, X_3, X_4$ in cam2;
for $i = 1$ **to** *4* **do**
  for $j = 1$ **to** *npoints* **do**
    if $X_i[j].z$ *= 0* ***and*** $X_{ic2}[j].z$ *= 0* **then**
      $pointsInFrontOfBothCams_i$++;
    end
  end
end
return X_i such that
$pointsInFrontOfBothCams_i$ is highest, with corresponding t, R;

Algorithm 1: Pseudo-Algorithm for triangulating 3D points, and relative pose for camera 2 with respect to camera 1. In our solution, we obtained $-t$ and R_{-90}.

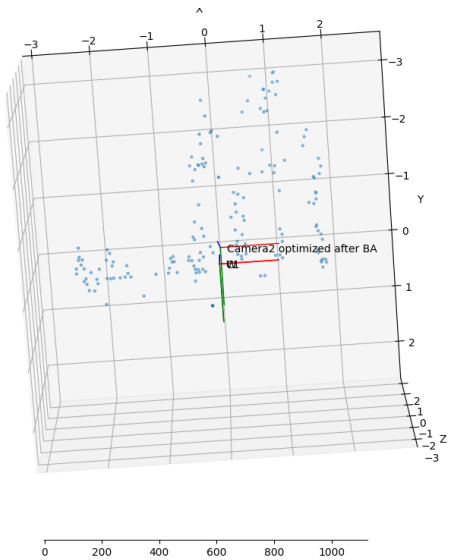

Figure 5: Our optimization for the pose of the second camera and the 3D points. Camera 2 is closer to the monument than camera 1, and slightly tilted to the left.

the projection matrix for the camera in the old picture, P_{old}. We decompose this matrix P_{old} to get the rotation, translation and calibration matrix for the camera (Check Section2.2.2 for a note on projection matrix decomposition). But, instead of using those as initial guess, we get the calibration matrix K_{c3} and feed it to a PnP (Perspective n-Points, Lepetit, Moreno, and Fua 2009) algorithm that returns a more refined guess for the rotation and translation of the third camera, t_{c3c1} and R_{c3c1}.

Now, we can perform the final Bundle Adjustment optimization, for all the 3D points and the camera parameters at once. We use the same Levenberg-Marquardt least squares as before, adding the necessary parameters for the third camera, and using a redefined cost function that can handle up to n cameras. Luckily, we already did the necessary modifications for the fourth lab session of the course. Our results can be seen in Figure 6.

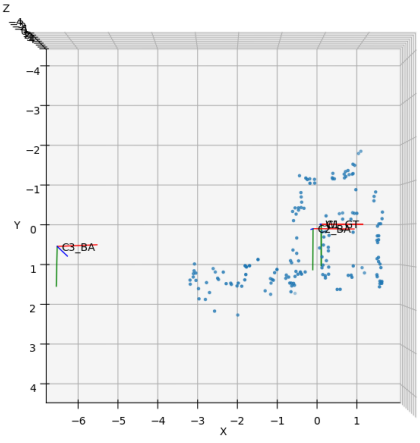

Figure 6: Our final computation for the position of the camera that took the old photo. The camera is located lower, more to the left, and farther away from the monument than the other two. Also pointing to the right.

Thus, we have optimized the position for the three cameras, as well as the 3D points.

2.2 Image differences

In order to compute the visual differences between the old image and a new one, we implemented a different execution mode within the same script. This second part of the program is simpler in comparison with its counterpart and needs fewer parameters.

2.2.1 Image alignment and homography

First, we need to allign both images. Since it's critical to allign them with as much precision as possible, we relied on some of openCV's algorithms, even though we had implemented our own versions for the first part of this project. We find keypoints with the help of the `sift.detectAndCompute` method, which we feed to a FLANN matcher. We filter those matches as per Lowe's ratio test, using a ratio of 0.7 as our thresholding value. Once we have our matches, we compute the homography to align the images with the help of `cv.findHomography`.

After the image alignment, we perform some extra processing to warp the new image over the old one, and then we crop the areas that are not overlaying in both (see top image in figure 7).

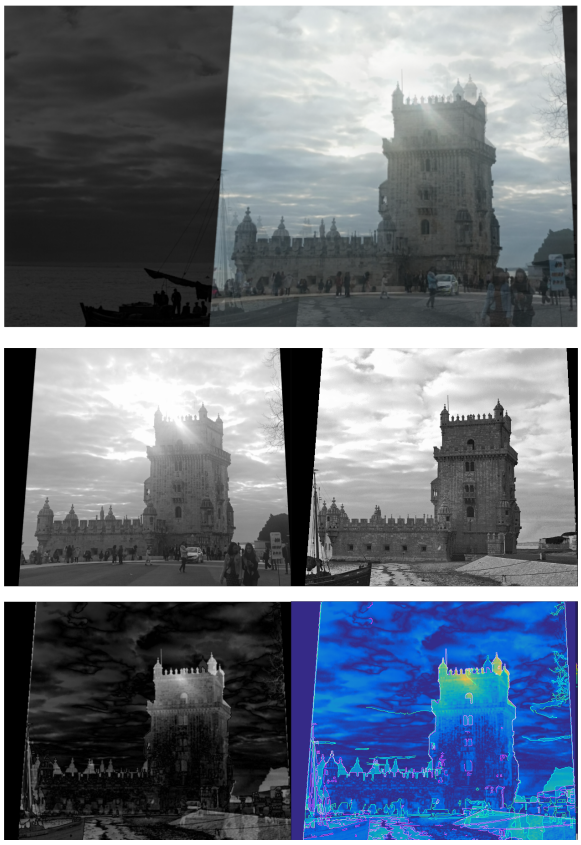

Figure 7: Part of the intermediate results of the steps in our image difference computation approach. At the top, the two original images overlayed on top of each other with the homography we computed. In the middle row there's the two images, cut to fit the overlaying area after performing histogram equalization. The last row shows the absolute difference between the images in the second row on the left, and our final result after applying the gradient and the Canny edges on the right.

2.2.2 Image difference

We checked several methods to compute our image differences, but decided to go with our own approach, refined by experimenting and iterating over different techniques and parameter values.

First, we transform our warped and cropped images into grayscale, and perform a histogram equalization in order to achieve a better final result, which result can be seen in the middle row in Figure 7. After this, we compute the absolute difference between both images (See

bottom left in Figure 7). We then perform Canny edge detection over both images, and give a different colour to each set of edges. We finetuned the parameters to fit our chosen images.

Once we have done all this, we apply a gradient over the grayscale absolute difference image, and overlay the Canny edges of both images over it. This way, slightly brighter areas hold some differences, and in case of doubt we can check what the original images looked like with the Canny edges. Our final result is presented in figure 8.

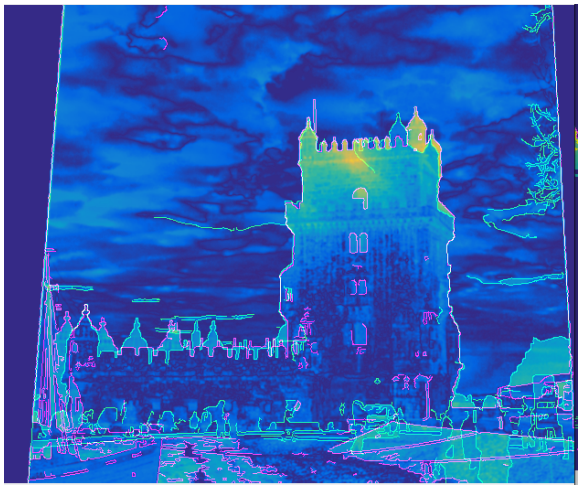

Figure 8: The final result for our image difference computation process. Edges in green correspond to our new image, while the pink ones correspond to the old one.

The most noticeable differences are in the lower part of the image corresponding to the road, which in the old picture used to be a beach. There's also some highlights in the waves and the silhouette of the boat. In the lower right side, we can notice a couple trees, people walking, and a parked car. The tower itself doesn't present many differences. The silhouettes in its left side are a consequence of a bad alignment, since the homography aligned better the plane represented by the main tower. There's a last barely noticeable difference, a second antenna that only appears in the new image in the right side of the top of the main tower.

Projection matrix decomposition

As a quick note, `cv.decomposeProjectionMatrix` wasn't working for us. Instead, we tried the decomposition explained in this link. It's based on performing a Cholesky decomposition. With this, we got correct results.

3 Final conclusions

We have developed a program that can complete both tasks of the assignment successfully, and can be used with many different images. Our program is also easily configurable from the user interface and its `config.ini` file. We have computed the location of the third camera for an extremely aged picture, taken at a low resolution, and compared against pictures taken at a different aspect ratio. We think this is a good proof that our methods are robust and can be used with different sets of images. Another strength is that we can feed the program several images and it will process them to get the most promising pair to compute the old camera location.

Our image difference algorithm allows us to quickly visualize changes between the two images, where we can easily see bright areas where changes occur. If the differences aren't clear enough, our Canny edges make spotting them more easy.

References

Fischler, Martin A. and Robert C. Bolles (June 1981). "Random sample consensus: a paradigm for model fitting with applications to image analysis and automated cartography". In: *Commun. ACM* 24.6, pp. 381–395. ISSN: 0001-0782. DOI: 10.1145/358669.358692. URL: https://doi.org/10.1145/358669.358692.

Hartley, R.I. (1997). "In defense of the eight-point algorithm". In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* 19.6, pp. 580–593. DOI: 10.1109/34.601246.

Lepetit, V., F. Moreno, and P. Fua (Feb. 2009). "EPnP: an accurate o(n) solution to the PnP problem". In: *International journal of computer vision* 81.2, pp. 155–166. DOI: 10.1007/s11263-008-0152-6. URL: http://hdl.handle.net/2117/10327.

Levenberg, Kenneth (1944). "A METHOD FOR THE SOLUTION OF CERTAIN NON – LINEAR PROBLEMS IN LEAST SQUARES". In: *Quarterly of Applied Mathematics* 2, pp. 164–168. URL: https://api.semanticscholar.org/CorpusID:124308544.

Lindenberger, Philipp, Paul-Edouard Sarlin, and Marc Pollefeys (2023). *LightGlue: Local Feature Matching at Light Speed*. arXiv: 2306.13643.

Lowe, D.G. (1999). "Object recognition from local scale-invariant features". In: *Proceedings of the Seventh IEEE International Conference on Computer Vision*. Vol. 2, 1150–1157 vol.2. DOI: 10.1109/ICCV.1999.790410.

Sarlin, Paul-Edouard et al. (2019). "SuperGlue: Learning Feature Matching with Graph Neural Networks". In: *CoRR* abs/1911.11763. arXiv: 1911.11763. URL: http://arxiv.org/abs/1911.11763.

Sutherland, I.E. (1974). "Three-dimensional data input by tablet". In: *Proceedings of the IEEE* 62.4, pp. 453–461. DOI: 10.1109/PROC.1974.9449.

Computer Vision course slides by the professors.