MC949/MO446 — Introduction to Computer Vision
INSTITUTO DE COMPUTAÇÃO — UNICAMP
2nd Semester, 2017

Project 2

Prof. Adín Ramírez Rivera
`adin@ic.unicamp.br`

1 Description

Video stabilization is a process in which a given video that has disturbances is processed in order to remove such problems. Video stabilization is an interesting task that helps to remove shakiness from videos that were captured with a free hand or in presence of movement from the camera.

In this project you will work with video processing and will implement an stabilization algorithm that depends on feature matching and model fitting through RANSAC.

2 Data

Before starting the project you need to capture some data. Get your cellphone or camera, and go take a few seconds of video to something interesting with your hand (if your device has video stabilization turn it off).

Experiment, and take several samples at different speeds and shakiness. If you like, share your videos in the forums of the course too!

3 Video Stabilization Algorithm

Our stabilization algorithm is quite simple but many variations can arise. The general algorithm for the implementation is the following. For every pair of images:

1. Find interest points in your image
2. Create feature descriptors for each of them
3. Hypothesize matches
4. Perform RANSAC to find matches consistent with an affine transformation
5. Use the inlier set to find a better estimate of the transformation
6. Transform the image

3.1 Interest Points and Descriptors

The first step in the stabilization process is to find the interest points in your images and describing them. For this task you need to select an algorithm (or set of algorithms) to find your image descriptors. Some of the descriptors you can look at and implement include: SIFT [1], SURF [2], BRIEF [3], or ORB [4]. Note that there are many other type of descriptors that you can read and implement.

You need to explain the descriptor you selected and briefly explain your implementation. Put more emphasis on the explanation on the selection of parameters (like thresholds) of your algorithm and how did you selected. I recommend you to perform several experiments to find good parameters, and to understand what the parameters are doing (see § 4).

3.2 Match Hypothesis

For each pair of images, use your interest point extractor and describe all the points. You need to find the candidate matches between these images. For this operation you need to use a metric function (commonly an ℓ^2 norm, or a metric define for your descriptor) and find the closest match for every point. Prune your candidates using a threshold to have a higher confidence for the matches.

Note that if your metric is not a distance you need to perform the process both ways. That is, compare the features of I_1 with the features of I_2, and compare the features of I_2 with the ones in I_1. Then, you keep the ones that are consistent.

At the end of this step you should end with a set of pairings for the interest points between your two images. Lets try to recover the transformation.

📝 Explain your algorithm for the matching, and your metric to compare the interest points. Show some examples of the matches in your experiments, and explain your results.

3.3 Affine Transformation Fitting

Our problem is to learn the transformation between our two images. For simplicity, we will assume an affine transformation (but in general you need a projection). We will consider the transformation

$$x' = ax + by + c, \tag{1}$$
$$y' = dx + ey + f, \tag{2}$$

where (x', y') are our transformed coordinates and (x, y) are our original coordinates. Our objective is to learn the parameters a, b, c, d, e, and f for our pair of images. Since our affine transformation has 6 degrees of freedom we need 3 point matches to solve the equations (since each point gives two equations).

We will use RANSAC to find the best transformation for our images. To do so, we need to select three point matches and solve the transformation they define. We will use least squares[1] to solve our transformation. That is, we need to change the shape of our transformations for x' (1) and y' (2) to a matrix form. Our system is

$$\begin{bmatrix} x_1 & y_1 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & x_1 & y_1 & 1 \\ x_2 & y_2 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & x_2 & y_2 & 1 \\ x_3 & y_3 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & x_3 & y_3 & 1 \end{bmatrix} \begin{bmatrix} a \\ b \\ c \\ d \\ e \\ f \end{bmatrix} = \begin{bmatrix} x'_1 \\ y'_1 \\ x'_2 \\ y'_2 \\ x'_3 \\ y'_3 \end{bmatrix}, \tag{3}$$
$$XA = Y, \tag{4}$$

where X, A, and Y represent the corresponding matrix. Hence the affine transformation A can be computed by

$$A = \left(X^T X\right)^{-1} \left(X^T Y\right). \tag{5}$$

Note that you need to implement this operation instead of using a solver.

Once you have the transformation A from a set of three points, you need to transform all the points from the first image using A, and see how many matches confirm the hypothesis on the transformed image. Repeat this step N times (📝 explain how do you compute the number of iterations N needed in your RANSAC implementation).

📝 In your report you need to explain your implementation of RANSAC (in particular focus on explaining your assumptions and the way you select the iterations and other parameters). Also, explain how you compute (or solve) the affine parameters (5).

3.4 Transform

After all the iterations on RANSAC you will select the affine transformation A that has the most amount of matches. Now compute a final transformation, $\hat{A}$, by solving (5) with all the matches (instead of three). Note that the matrix X can have a general form of $2M \times 6$ and Y can be $2M \times 1$, where M is the number of points you have (in this

[1] You can't use a solver. You will need to perform the matrix operations to solve the optimization problem.

case your number of final matches). Explain your generalization of 5. You can explain it together with the explanation on how you solve the parameters for brevity.

Using the final transformation $\hat{A}$ transform your image. You can, for example, transform the second frame into the reference frame of the first one. Then for the following frame (the third one) you want to align it with the aligned version of the second one.

Repeat this process until all the frames in the video are aligned. Note that you may need to increase the spatial size of the video and fill the gaps in the frames with zeros to accommodate the variations on the video. (Explain your technique to deal with the resizing problem.)

Explain your approach to align the images using the final transformation, and your assumptions to make it work.

4 Experiments

Evaluate your method on the data you gathered at the beginning (see § 2). Perform several experiments tweaking your parameters and evaluating different configurations. You can show plots that show evaluation metrics of your results to summarize the different executions (e.g., you can show MSE of your aligned images vs. the parameter variations, if you find other metrics cite your sources).

You need to show your work (that is show frames of the stabilized video, and superimpose markers where the features are from each pair of frames, consider using different colors to show and distinguish the markers).

Additionally, include the videos of the successful stabilization. I recommend to create a small video to showcase your algorithm, where you show side by side the original video and your stabilization results. You can put several videos to show your results (successful or not). **Consider a 2 minute maximum run time.**

5 Extras

If you are eager to do more and bring your method to a next level try the following:

- Use a projective transformation instead of an affine one (+10%).
- Use other variations of RANSAC to improve your results (+5%).
- Optimize your code to stabilize the video on real time (+15%).

6 Evaluation

Your grade will be defined by the following aspects:

1. Interest point detection and description	20%
2. Hypothesis matching and search	15%
3. Affine transformation solver	15%
4. RANSAC	15%
5. Final transformation and alignment	15%
6. Video showing your results	10%
7. Overall report	10%

Each item corresponds to the questions and requirements defined in the previous sections. The overall report, point 7, refers to the evaluation of the details presented in your report, your way of presenting results, explanation and use of concepts and theory, references, etc. **Your English usage won't be graded**, but your ability to present your results, ideas, and how the are supported will be. Each other point will be evaluated according to the completeness and correctness of the requested items.

You will be eligible for extra points, defined in § 5, if you completed the required assignment, and you have an explanation of your extra development in your report.

7 Submission

You need to create a folder named `p2-XX-YY` where `XX` and `YY` are the RAs of the members of your team. Note that the RAs **must** be sorted.

Your submission must have the following subfolders:

- `input`: a directory containing the input assets (images, videos or other data) supplied with the project. **Store the videos used to produce the experiments here.** For any other asset, setup your `Makefile` to automatically download them from a public server or repository. **Your videos shouldn't be too big.** If you want to show your results on high definition videos, host them somewhere else, and use your script to download them from there.
- `output`: a directory where your application should produce all the generated files (otherwise stated in the problem). This directory should be empty.
- `my-output`: use this directory to show your previous results (in case you want to show intermediary work). At least, this folder must contain the output of your created video to showcase your results (original and stabilized)—see point 6 in § 6.
- `src`: a directory containing all your source code. You only need to submit files that are not derived from other files or through compilation. In case some processing is needed, prefer to submit a script that does that instead of submitting the files.
- `Makefile`: a makefile that executes your code through the docker image. An image is already built and available for use (`adnrv/opencv` at the docker hub registry). The code will be executed through a standard call to `make`, so other dependencies must be provided by you under that constraints.
- `report.pdf`: a PDF file that shows all your work for the given project, including images (labeled appropriately, that is, in accordance to the convention given) and other outputs needed to explain and convey your work. When needed iclude explanations to the questions given in the project.

The principal folder must be zipped into `p2-XX-YY.zip` and submitted through the Moodle website. No other files will be accepted. **Note that upon unzipping your file, the original folder** `p2-XX-YY.zip` **must be created.** That is, do not zip the contents, but rather the folder.

8 Notes

- For the size of your videos you can experiment with different sizes
- Note that there are several implementations that you can find in the internet. This project is for **you to implement** the algorithms. Thus, do not submit code from others. And if you re-use code from someone for a non-restricted part, disclose it in your report and code.
- You are not allowed to use any prebuilt function for image transformation or warping, for computing solutions to the least squares problem, RANSAC, interest point detector, or feature extractors. You are allow, however, to use the operations for matrices supplied by OpenCV or other matrix library (e.g., multiplication, transpose, inverse, etc.).
- All the submissions must be self contained and must be executable in a Linux environment. Specifically, your code must execute in the docker image `adnrv/opencv`, available at docker hub (`https://hub.docker.com/r/adnrv/opencv/`).
- It is your responsibility to make sure your code compiles and executes correctly. No effort will be made to run your code besides executing `make` inside a docker.
- You must program in Python 3.6 (or higher) or in C/C++ with `gcc` version 6.2.0, using OpenCV 3.2.0. All available within the image. If you need to install other packages you must do so within your `Makefile` as automatic prerequisites.
- Check the `Makefile` provided with the project zero. It will be used to automatically execute your code. You must pass the full path of your project folder: `make SOURCE_DIR=$(realpath ./pA-XX-YY)`.

References

[1] D. G. Lowe, "Object recognition from local scale-invariant features," in *Computer vision, 1999. The proceedings of the seventh IEEE international conference on*, IEEE, vol. 2, 1999, pp. 1150–1157.

[2] H. Bay, T. Tuytelaars, and L. Van Gool, "Surf: Speeded up robust features," *Computer vision—ECCV 2006*, pp. 404–417, 2006.

[3] M. Calonder, V. Lepetit, C. Strecha, and P. Fua, "Brief: Binary robust independent elementary features," *Computer Vision—ECCV 2010*, pp. 778–792, 2010.

[4] E. Rublee, V. Rabaud, K. Konolige, and G. Bradski, "Orb: An efficient alternative to sift or surf," in *Computer Vision (ICCV), 2011 IEEE international conference on*, IEEE, 2011, pp. 2564–2571.