Università degli Studi di Padova

– Computer Vision ◎ Final Project –

-Drone Panorama-
A comparions between the rigid translation and the affine model

June 18, 2017

Instructor:
Pietro Zanuttigh

Students:
Guglielmo Camporese
Matteo Ciprian
Ettore Mariotti

Contents

Introduction

In this final project for the course 'computer vision', our objective is to build a piece of software able to take images from a video and construct a panorama, an higher resolution image obtained with the skillful combination of the video frames. Our focus in particular will be on comparing two different way of merging the images together:

- Rigid translation (2 parameters)
- Affine transformation (6 parameters)

The rigid translation account only for vertical and horizontal translation:

$$\begin{bmatrix} u \\ v \end{bmatrix} = \begin{bmatrix} x \\ y \end{bmatrix} + \begin{bmatrix} t_x \\ t_y \end{bmatrix} \tag{1}$$

The affine transformation instead relates to a more general model in the form:

$$\begin{bmatrix} u \\ v \end{bmatrix} = \begin{bmatrix} m_1 & m_2 \\ m_3 & m_4 \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix} + \begin{bmatrix} t_x \\ t_y \end{bmatrix} \tag{2}$$

Where (x, y) represent the coordinate system of the single image, (u, v) represent the coordinate system of the final panorama.

We wanted to test this method on a different setting than the ideal case of the camera mounted on a professional tripod, so we opted for a quadcopter.

In particular, we used the drone 'Bebop 2' with his included camera as the acquisition system and MATLAB as the language for building the software. The following of this report is organized as follows: in 1 explain the the acquisition system setup, in 2 we go in depth about the MATLAB script, in 3 results are presented and finally in 4 conclusions are drawn.

Section 1

Acquisition System

The video was taken by a drone (a quadcopter).

1.1 Drone Characteristics

The quadcopter chosen was the ***Bebop 2***, from *Parrot*. It comes with an excellent 3-axis physical stabilization of the whole body, moreover the camera system have both an hardware and a software proprietary stabilization mechanism.

The camera have a wide-angle optics with a CCD sensor of 14Mega pixels. It takes large wide-angle pictures and post-process them in order to transform them in typical planar 2D images.

This allows the drone to tilt the view in the vertical axis without actually having to move the camera: the upper or lower part of the wide-angle is anti-transformed from a spherical projection and returned in output.

Figure 1.1: Raw wide-angle image

Figure 1.2: Post-processed output

1.2 Flight Setup

The drone comes with a controlling suite called *FreeFlight Pro* that can be installed on any smartphone. Of that suite we used *FlightPlan*, an environment that allows the precise planification of the flight, in particular we could choose *when* to take the clip, *where*, at *what altitude*, the *rotation speed* and *angle*. Of relevance we report:

1. Rotation Angle: 130°
2. Rotation Speed: $30°/s$ $\rightarrow$ 4.3 seconds
3. Vertical Tilt: 0° and 10°

The acquisition were performed in daylight and with such setup we got in output 2 videos in which every frame isn't blurred.

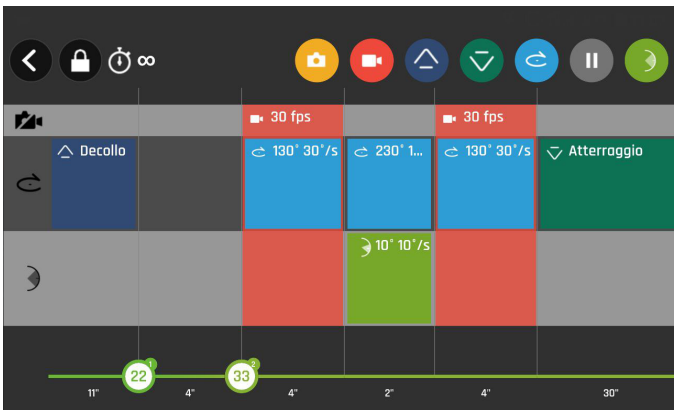

Figure 1.3: *FlightPlan* setup example

Section 2

Matlab Script

This section describes the scope of the various functions we constructed, for a detailed description of the parameters of each m-file we invite the reader to consult the appendix of this report.

Since we wanted to build a script that merged two horizontal panoramas shifted by a vertical angle, we first started with designing the code so that it could merge two adjacent images along the horizontal direction (‘`hor_merge`’). For the vertical merge we built another function (‘`vertical_merge`’) that calls back the former routine in order to unite the panoramas (detail will be described later).

`hor_merge` works as follows:

1. Extract useful frames from video(s);
2. Compute SIFT descriptors and find matches between adjacent images;
3. Use RANSAC for finding a suitable set of consistent points with the transformation and estimate its parameters;
4. Merge and blend the images together;

`vertical_merge` do this:

1. Rotate of 90° the panoramas in output from `hor_merge`;
2. Call `hor_merge` and merge the two rotated panoramas;

2.1 Frame Extraction

In order to automate the frame extraction from the video we built the function `read_video` which takes in input the videos and ask the user the angle spanned from the moving camera. It return in output a set of frames in an appropriate directory.

Another routine[1] `read_images` pick the frames and store them in the MATLAB workspace, it requires in input to specify the directory of the files to load and their extension. This allows the user to work with either a video or directly a set of images.

2.2 Features Matching

For computing the relative position between two object we first have to identify the points of interest. This objective is achieved with SIFT algorithm, that for each point of interest it computes a 128-dimension vector. In order to have a match between two feature point the Euclidean distance of the respective descriptors is computed, after a number of check the matching process returns the points representing the same object in different pictures.

For our code we used the VLFeat library, which comes with handy methods for computing descriptors and matches.

2.3 RANSAC Parameters Estimation

As false positives may appear in the output of SIFT algorithm, a method for identifying a consistent set of points for computing the transformation (rigid or affine) is needed. This problem is solved with a simplified version of RANSAC, a non deterministic algorithm. The key idea is the following:

1. Select correspondences $(x, y) \leftrightarrow (u, v)$ at random and get its transformation $\mathbf{T}$ (following the model of choice)

2. Count how many correspondences are consistent with $\mathbf{T}$ using a simple threshold on the absolute (or squared) error defined as $|\Delta x_n - \Delta x| + |\Delta y_n - \Delta y| < th$

[1] This choice has been done for code-modularity purpose

3. Iterate k times (in this case $k = 1000$) and keep the correspondence with the largest compatible set

4. Solve the system

Once we have a consistent set of points we can proceed to estimate the transformation.

2.3.1 Rigid Translation

For the rigid translation our model will be:

$$\begin{bmatrix} u \\ v \end{bmatrix} = \begin{bmatrix} x \\ y \end{bmatrix} + \begin{bmatrix} t_x \\ t_y \end{bmatrix} \tag{2.1}$$

Given the large set of $\mathbf{t}$ vectors, its mode is returned (the most frequent value).

2.3.2 Affine Transformation

For the affine transformation our model will be:

$$\begin{bmatrix} u \\ v \end{bmatrix} = \begin{bmatrix} m_1 & m_2 \\ m_3 & m_4 \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix} + \begin{bmatrix} t_x \\ t_y \end{bmatrix} \tag{2.2}$$

That can be rewritten without loss of generality as:

$$\begin{bmatrix} x_1 & y_1 & 0 & 0 & 1 & 0 \\ 0 & 0 & x_2 & y_2 & 0 & 1 \\ x_3 & y_3 & 0 & 0 & 1 & 0 \\ 0 & 0 & x_4 & y_4 & 0 & 1 \end{bmatrix} \begin{bmatrix} m_1 \\ m_2 \\ m_3 \\ m_4 \\ t_x \\ t_y \end{bmatrix} = \begin{bmatrix} u_1 \\ v_1 \\ u_2 \\ v_2 \\ u_3 \\ v_3 \end{bmatrix} \tag{2.3}$$

Since in general we will have a overdetermined system in the form $A \cdot x = b$ we are going to compute the solution in the least square sense (the one that minimize the squared error), in other words we want to find:

$$\min_x ||A \cdot x = b||^2 \tag{2.4}$$

This is achieved trough:

$$x = (A^t A)^{-1} A^t b \tag{2.5}$$

2.4 Merge and Blending

Once we get the parameters for the model we need to juxtapose adjacent images and blend the shared overlapping portion of scene. The juxtaposition of the images was done creating one big frame with the dimensions of the final panorama for each image in the dataset (we shall call it *slide*). Then the i-th image is put in the appropriate position with respect to the estimated transformation. Once this have been done for all the images in the dataset, a simple blending technique is performed: the overlapping scene between frame i and $i + 1$ is replaced with a weighted average of the two using a linear ramp.

Since the underlying models are different we created two different algorithms, one for the rigid translation and one for the affine model. The key steps after the estimate of the model parameters are:

1. Initialize the dimension of the final image
2. Create one slide (layer) for each image with the dimension of the final panorama all padded to 0 except the portion mapped by the transformation of the frame
3. The blending is different for each model:

   Rigid Translation: The overlap region is replaced with a weighted average done with a linear ramp;

   Affine Transformation: The overlap region is replaced with a weighted average using a convex combination of the distances of the pixels from the centers of the images;

2.4.1 Pseudocode

We now present the pseudocode for merging and blending in the following pages.

Pseudocode - Merge (Translation Model)

Input: Set of color images $\mathcal{T}$ thas has to be merged.
Ouput: Merged image Im_final.

$N \leftarrow |\mathcal{T}|$;
// compute the cilindrical projection:
$fov \leftarrow 70$;
$\mathcal{T}_p \leftarrow project(\mathcal{T}, fov)$;
// compute the frames and the descriptor using SIFT:
$\mathcal{T}_p^g \leftarrow rgb2gray(\mathcal{T}_p)$;
$[F_i, D_i] \leftarrow vl\_sift(\mathcal{T}_p^g),\ i = 1, \ldots, N$;
// find the matches:
$[M_i, S_i] \leftarrow vl\_ubmatch(D_i, D_{i+1}),\ i = 1, \ldots, N-1$;
// estimate the translation:
$T_i \leftarrow RANSAC(F_i, F_{i+1}, M_i),\ i = 1, \ldots, N-1$;
// initialize the dimensions of the final image:
$dim\_y\_final \leftarrow 2 \cdot size\_y(\mathcal{T}(1))$;
$dim\_x\_final \leftarrow N \cdot size\_x(\mathcal{T}(1))$;
// compute one slide for each image:
$S_i \leftarrow get\_slide(T_i, \mathcal{T}_p(i), \mathcal{T}_p(i+1)),\ i = 1, \ldots, N$;
$O_i \leftarrow get\_overlap\_region(S_i, S_{i+1}),\ i = 1, \ldots, N-1$;
$overlap\_size\_x_i \leftarrow get\_overlap\_region\_size\_x(O_i),\ i = 1, \ldots, N-1$;
// linear blending:
$Im\_final \leftarrow empty\_color\_image(dim\_y\_final, dim\_x\_final)$;
$Im\_final \leftarrow \sum_i^N S_i$;
for $y = 1 : dim\_y\_final$ **do**
  $j \leftarrow 1$;
  for $x = 1 : dim\_x\_final$ **do**
    if $(x, y) \in O_i$ **then**
      $\lambda \leftarrow \frac{j}{overlap\_size\_x_i}$;
      $Im\_final(y, x) \leftarrow S_i \cdot \lambda + S_{i+1} \cdot (1 - \lambda)$;
      $j \leftarrow j + 1$;
    else
      $j \leftarrow 1$;
    end
  end
end

Pseudocode - Merge (Affine Model)

Input: Set of color images $\mathcal{T}$ thas has to be merged.
Ouput: Merged image Im_final.

$N \leftarrow |\mathcal{T}|$;
// compute the cilindrical projection:
$fov \leftarrow 70$;
$\mathcal{T}_p \leftarrow project(\mathcal{T}, fov)$;
// compute the frames and the descriptor using SIFT:
$\mathcal{T}_p^g \leftarrow rgb2gray(\mathcal{T}_p)$;
$[F_i, D_i] \leftarrow vl\_sift(\mathcal{T}_p^g),\ i = 1, \ldots, N$;
// find the matches:
$[M_i, S_i] \leftarrow vl\_ubmatch(D_i, D_{i+1}),\ i = 1, \ldots, N-1$;
// estimate the affine transform:
$[R_i, T_i] \leftarrow RANSAC(F_i, F_{i+1}, M_i),\ i = 1, \ldots, N-1$;
// initialize the dimensions of the final image:
$dim\_y\_final \leftarrow 2 \cdot size\_y(\mathcal{T}(1))$;
$dim\_x\_final \leftarrow N \cdot size\_x(\mathcal{T}(1))$;
// compute one slide for each image:
$S_i \leftarrow get\_slide(T_i, \mathcal{T}_p(i), \mathcal{T}_p(i+1)),\ i = 1, \ldots, N$;
$O_i \leftarrow get\_overlap\_region(S_i, S_{i+1}),\ i = 1, \ldots, N-1$;
$c_i \leftarrow get\_center\_of\_image(S_i),\ i = 1, \ldots, N$;
// linear blending:
$Im\_final \leftarrow empty\_color\_image(dim\_y\_final, dim\_x\_final)$;
$Im\_final \leftarrow \sum_i^N S_i$;
for $y = 1 : dim\_y\_final$ **do**
  for $x = 1 : dim\_x\_final$ **do**
    if $(x, y) \in O_i$ **then**
      $point \leftarrow (x, y)$;
      $d_1 \leftarrow dist(point, c_i)$;
      $d_2 \leftarrow dist(point, c_{i+1})$;
      $Im\_final(y, x) \leftarrow S_i \cdot \frac{d_2}{d_1+d_2} + S_{i+1} \cdot \frac{d_1}{d_1+d_2}$;
    end
  end
end

Section 3

Results

We used different datasets for testing our software and results are shown below:

3.1 Rigid translation

With the rigid translation model we get nice results as long as the vertical angle of the camera from the horizon is small. When we merge the pictures together ghosting appears as expected, since it is an intrinsic problem of linear blending.

3.1.1 Arcella

The first dataset comes from an aerial video taken in Arcella. We took a video spanning 130° horizontally, with a vertical axis inclination of around 0° the first and 20° the second[1].

This dataset is quite challenging since the richness of details in the depth field makes the various objects look different from a point of view to another. The geometry of the objects in the scene changes from angle to angle.

The results for the non tilted version (see Figure 3.1) are quite good even though in the center ghosting artifacts are subtle but evident. [t] When applying the algorithm to the 20° tilted video the ghosting is much more evident (see Figure 3.2). This is probably due to the change of the geometry of the objects caused by the change of the perspective. This highlights one

[1]It is not perfectly 0° as it is not trivial to keep it stable with wind

Figure 3.1: Arcella - vertical angle near 0

Figure 3.2: Arcella - Vertical angle around 20

of the limitation of the translation model: it can deal only with non-tilted views

The final merge (Figure 3.3) finally is not really satisfying as it carry over the errors from the past steps.

3.1.2 Piazzetta

This dataset named 'Piazzetta' was taken at a lower altitude. Here the artifacts due to the depth perspective are fewer and the results are better. Notice for example the evident different position between the face and the corner of the building, that is due to the perspective change of filming with

Figure 3.3: Arcella - Final merge

Figure 3.4: Piazzetta - angle 0

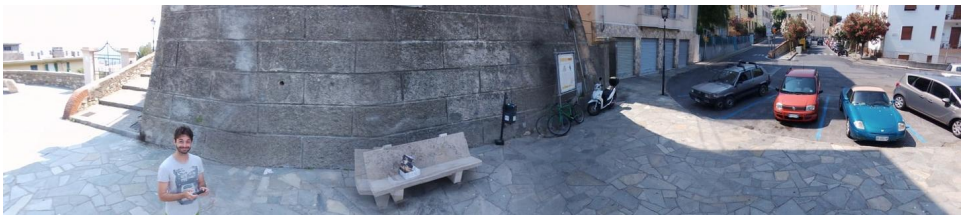

Figure 3.5: Piazzetta - Tilted

a different vertical angle.

Given the two panoramas the vertical merge is quite satisfying, probably because the angle was quite small.

3.1.3 Porto Maurizio

This dataset is another aerial view taken from high altitude. We can see that the translation model suffer in the same way as the Arcella dataset.

This model clearly can't handle the change in the object geometry due to perspective changes. Notice however the left side of image 3.7: being the sea all flat we got a nice merge.

Figure 3.6: Piazzetta - Final Merge

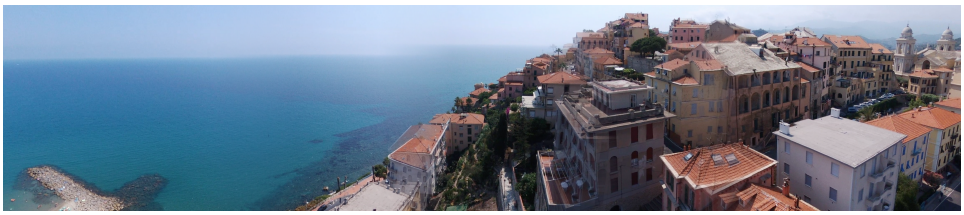

Figure 3.7: Porto Maurizio - Flat

Figure 3.8: Porto Maurizio - Tilted

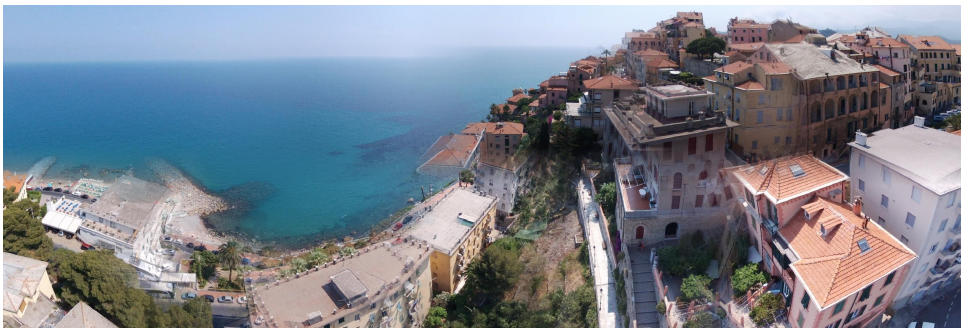

Figure 3.9: Porto Maurizio - Final Merge

Figure 3.10: Arcella - Flat (affine)

3.2 Affine Model

The same datasets were tested with the affine model. It shows better robustness to the change of perspective, but it is not an optimal solution that solve the problem completely. Moreover since the output values are fractionary, an interpolation is needed. We reserved this for a future work. As we can see the images are tilted in a circular fashion, this is probably due to the non-0 vertical tilt of the camera (in fact, the more tilted the takes, the more circular the output). This tilted output makes impossible to utilize the function `vertical_merge` as done for the rigid translation, so only the distinct panoramas are presented.

3.2.1 Arcella

We can see that here the ghost instances are fewer and less evident, as the change in geometry due to perspective is in part accounted by the more general and complex transformation.

3.2.2 Piazzetta

Analogous considerations can be done with the Piazzetta dataset. It gives us the insight that the affine model partially solve the problems of objects in different depths.

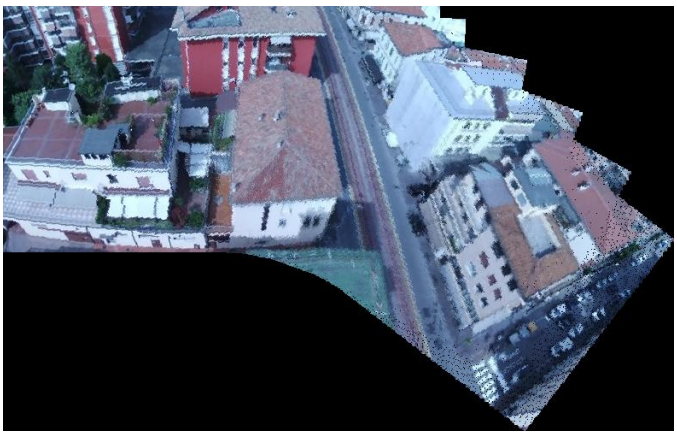

Figure 3.11: Arcella - Tilted (affine)

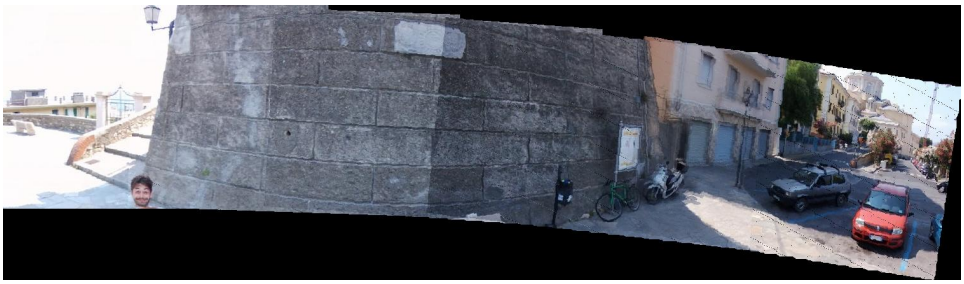

Figure 3.12: Piazzetta - Flat (affine)

Figure 3.13: Piazzetta - Tilted (affine)

Section 4

Conclusions

We tested and implemented two different approach to image stiching: rigid translation and affine transformation. The former is faster than the latter because of its reduced complexity, but its applications scenarios are limited. The affine on the other hand gives better performances in terms of constructing a continous image, but give space to other problems: interpolation and circular disposition of the images. As a future work one could think about fixing the interpolation issue and reversing the geometric circular disposition.

We can acknowledge the fact that if we move ourself from the ideal situation of an high performance camera on a tripod things get more complex and solving the problem of panorama in its general form require a non-trivial effort. In particular the change of depth in the scene requires more elaborate models in order to solve the issues arose from the change in the perspective.

Nonetheless in a simplified environment, where the change in depth are limited and the acquisition system is stable and perfectly flat, the translation and affine model achieve reasonable performances.

Appendix A

Matlab script sintax

A.1 Loading dataset functions

`read_video.m`:

- INPUT:

  dir_video_input: directory of videos

- OUTPUT:

  sampled frames saved in the same directory of the input

- Videos need to be named '`angle_`⊗', where ⊗ represent the vertical axis tilt of the camera

- The script ask the user to specify the horizontal angle spanned by the camera so it can automatically compute what frame capture next

`read_images.m`:

- INPUT:

  sdirectory: directory of images,

  ext: their extension

- OUTPUT:

  images: cell of images

  n_images: number of images

  fov: the field of view[1]

[1] the returned value is always 66

A.2 Merging function

`merge.m`:

- INPUT:

  images: cell containing the set of images that has to be merged

  n_im : (optional) select how many images have to be merged

  str_project: (optional) select the projection of the images to use.

- OUTPUT:

  image_fi: horizontal merge of pictures

`verticale_merge.m`:

- INPUT:

  img_struct: struct containing the set of images that has to be vertically merged

- OUTPUT:

  im_final: vertical merge of pictures

A.3 Main

`test_merge.m`: read data from two folders: '/input_video' (where videos are labeled as described in read_video), '/input' where it expect to find the images. Perform the merge and return the output 'final_merge.jpg' in the main directory.