Master's degree in
Computer Engineering for Robotics and Smart Industry

UNIVERSITÀ di VERONA
Dipartimento di INFORMATICA

Computer Vision Project

AUTOMATIC 2D MOSAICING:

Development of a software application aimed at automatically generating a 2D mosaic, given a set of images related by an homography

Prof. Umberto Castellani

Fabio Castellini

Index

- Introductory overview about the ***mosaicing*** problem
- ***Salient points*** detection, description and matching (SIFT)
- Robust estimation of the ***homography matrix*** (RANSAC)
- ***Image warping***, alignment and stitching
- Trivial approach to ***color blending***
- Project ***objectives*** and ***developed features***
- Brief discussion about ***Photoshop*** and MATLAB's ***Computer Vision Toolbox*** results
- Mosaicing ***examples*** and ***comparisons***
- ***Conclusions***

Introductory overview about the mosaicing problem

- **Mosacing** *is the process of stitching together several images with the objective of increasing the field of view*. It is a widely used technique in phography because it allows to produce high resolution panorama images.
- In a few words, mosaicing consists in finding *point-to-point corrispondences* between the overlapping images, *stitching them together* warping one image, with respect to the other and dealing with the *color blending*. By iteratively repeating the process for all the images, the final mosaic is obtained.
- To have a better end result, all the photographs should be taken with a steady tripod rotating by a certain degree its head. According to Photoshop, the overlapping part should be approximately 40%, too much or too little wouldn't benefit the mosaicing.
- During this work, some of the guidelines weren't respected on purpose, to test the robustness of the algorithms.

Salient points detection, description and matching (SIFT)

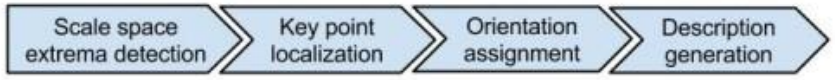
Salient points

- The starting step of the project is *point to point corrispondences detection* between images. It can be done automatically using the **S**cale **I**nvariant **F**eature **T**ransform (SIFT) algorithm.
- Salient points are pixels that can be recognized from the scene and they're fundamental to then estimate the homography matrix.
- This points of interest tipically belong to a region that differs from its neighborhood, contrasting the nearby pixels.
- Salient points must be detected with high accuracy, robustly and independently from illumination, rotation, scale, noise, perspective, viewing conditions...

Salient points detection, description and matching (SIFT)

Scale Invariant Feature Transform (SIFT)

- SIFT is the algorithm that has been studied during the course and that will be used in the project's implementation. This method was published by David Lowe in 1999 and became very popular thanks to its reliability.
- It consists in 3 main steps:
  - *Salient points* ***detection***
  - *Salient points* ***description***
  - *Salient points* ***matching***



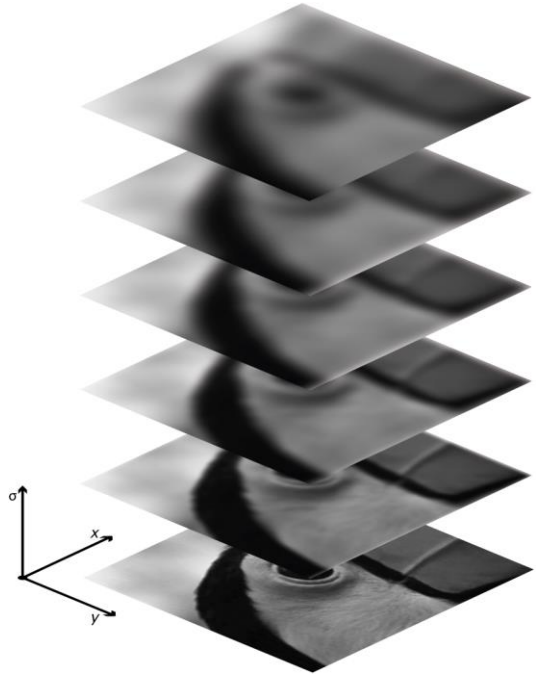
SIFT's pipeline

Feature points extracted using SIFT (only 20 corrispondences were shown)

Salient points detection, description and matching (SIFT)

Salient points detection – *Scale Space & Image Pyramid*

- The ***Scale Space*** of an image is a function $L(x,y,\sigma)$ that is obtained by a convolution of a Gaussian kernel (at different scales) with the input image: $L(x,y,\sigma) = G(x,y,\sigma) * I(x,y)$

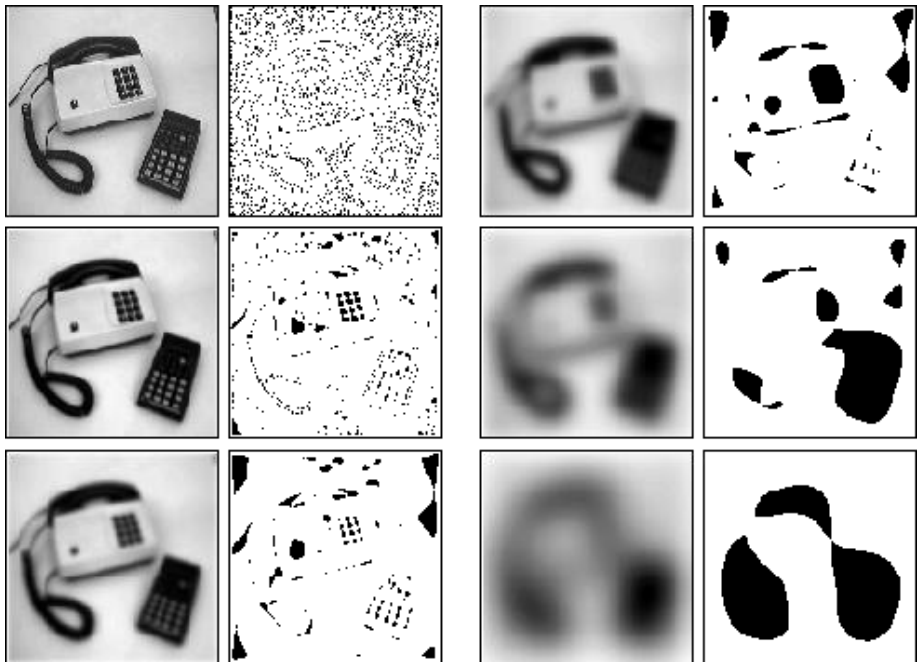


- Basically, the starting image is blurred by a Gaussian filter G and σ proportionally defines the level of blur.
- Considering *different scale intensities* of σ, a Scale Space is obtained.

Salient points detection, description and matching (SIFT)

Salient points detection – *Scale Space & Image Pyramid*

- The benefit of using different scale images is to extract more information about the scene.

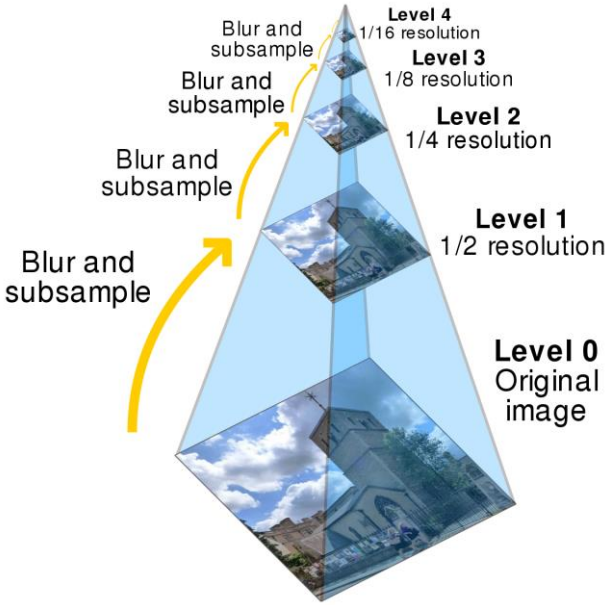


- As the figure shows, the scene contains several details. Those can be detected from some of the filtered images and not from others.
- In particular, all the filtered images are shown and next to those, their respective *binarization*.
- Looking at the binary images we can observe certain elements more in detail, depending on the used Gaussian filter (for example the phone's numpad, the calculator shape...).

Salient points detection, description and matching (SIFT)

Salient points detection – *Scale Space & Image Pyramid*

- The concept of ***Image Pyramid*** is similar to *Scale Space* but it takes into consideration computational cost. In particular, rather than enlarging the Gaussian filter's bandwidth, the starting image can be resampled at different levels.

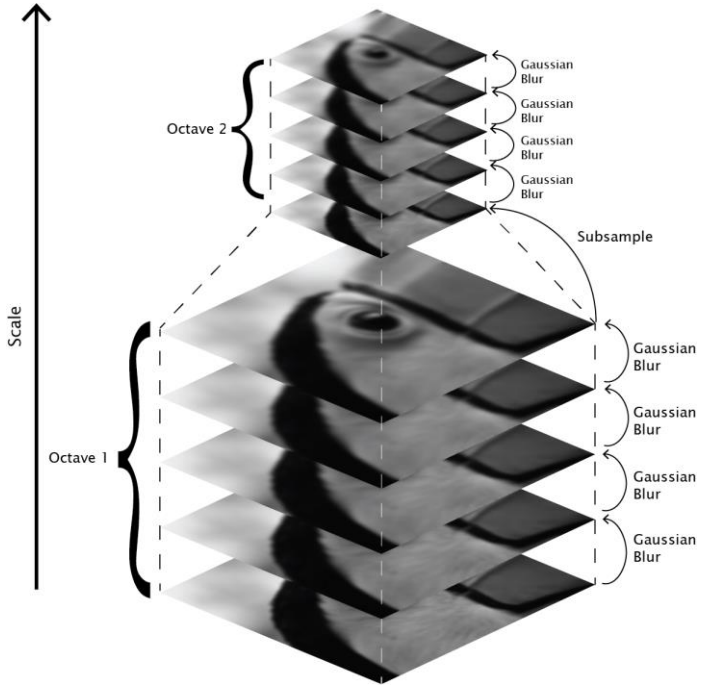


- This way, instead of changing the filter's size, the overall image can be downsampled reaching the same result in a much efficient manner.
- A small window on *Level 1* is larger than the same image window on *Level 0*.

Salient points detection, description and matching (SIFT)

Salient points detection – *Scale Space & Image Pyramid*

- ***Image Pyramid*** and ***Scale Space*** can be combined in order to fasten the process and retrieve the maximum information from the starting scene.



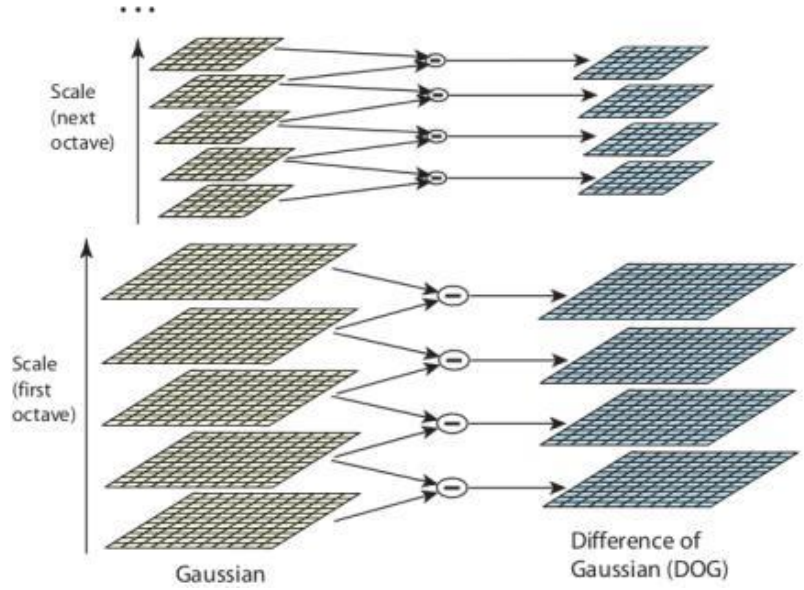
- By considering same regions at different resolutions, the scaling factor is intrinsically considered during the process.
- Corresponding points are detected at *different* image scales. Once they're normalized salient points can be compared.

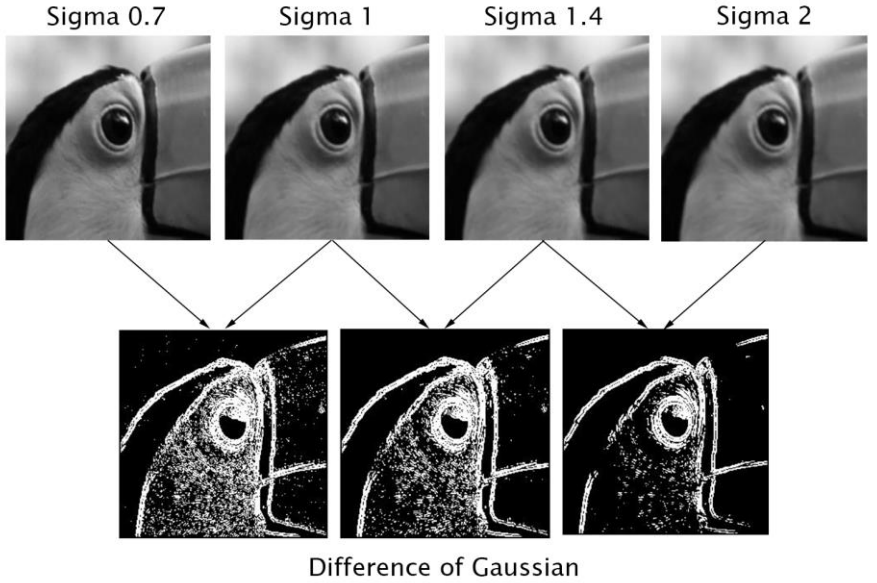
Salient points detection, description and matching (SIFT)

Salient points detection – *Difference of Gaussian*

- The ***difference of Gaussian*** filter can be used to approximate the **Laplacian of Gaussian** filter, to make the process faster, according to the following formulation:

$$D(x, y, \sigma) = \big(G(x, y, k\sigma) - G(x, y, \sigma)\big) * I(x, y) = L(x, y, k\sigma) - L(x, y, \sigma)$$





Salient points detection, description and matching (SIFT)

Salient points detection – *Difference of Gaussian*

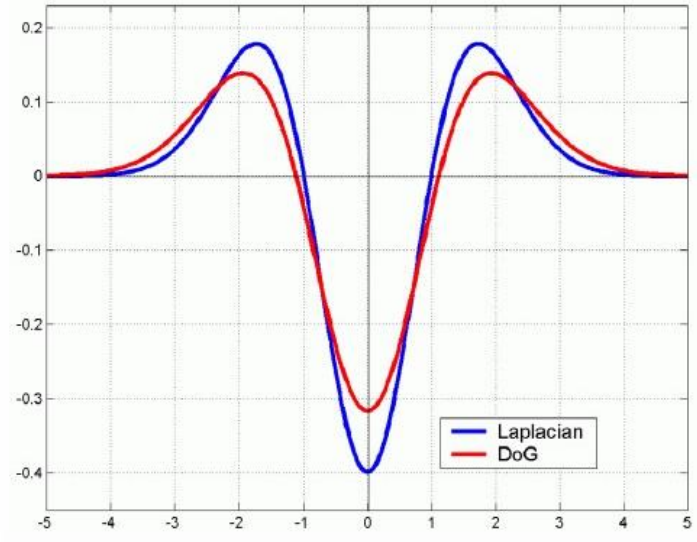
- Given the Difference of Gaussian Space, **salient points** (or regions) **are the extremants** (minimum or maximum) of this volumetric representation (each layer is composed by one image), in which both Space and Scale are considered.

$$L = \sigma^2 \left(G_{xx}(x, y, \sigma) + G_{yy}(x, y, \sigma) \right)$$

(Laplacian)

$$DoG = G(x, y, k\sigma) - G(x, y, \sigma)$$

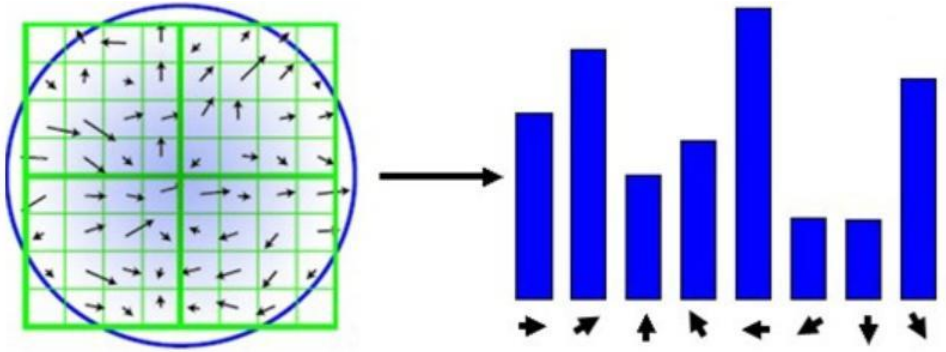
(Difference of Gaussians)



Salient points detection, description and matching (SIFT)

Salient points description

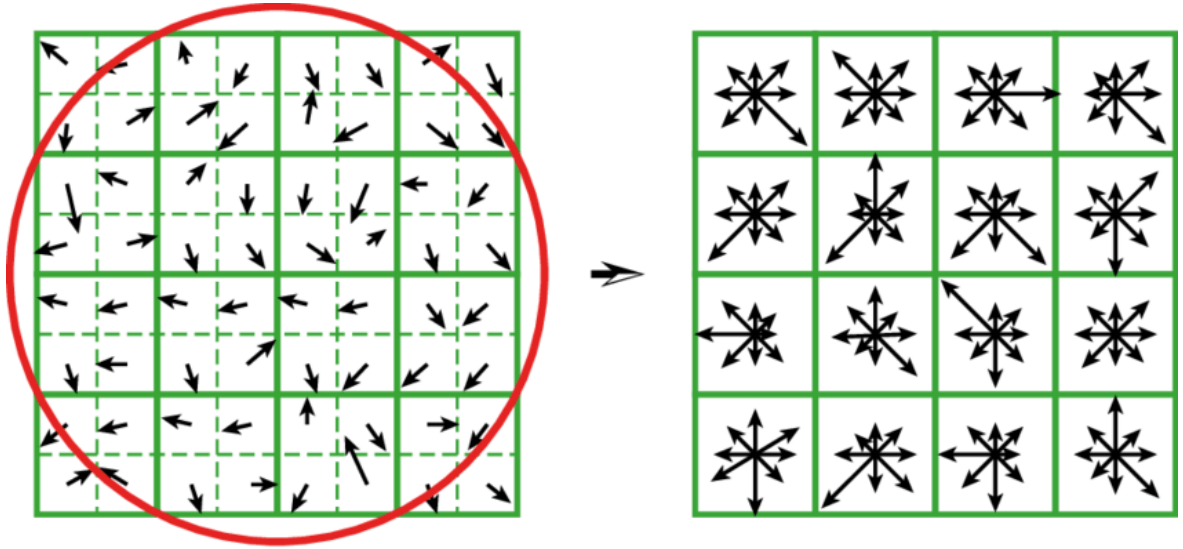
- Other than ***points*** and ***scale***, ***orientation*** has to be computed.
- Starting from the neighborhood (16x16 pixels) of the detected point, gradients of all the pixels in that region can be measured. **Gradient** is a vector ("arrow") that defines intensity and direction.
- So, a histogram of orientations can be drawn for a salient point. The maximum value on the histogram represent the global orientation of the point.



Salient points detection, description and matching (SIFT)

Salient points description

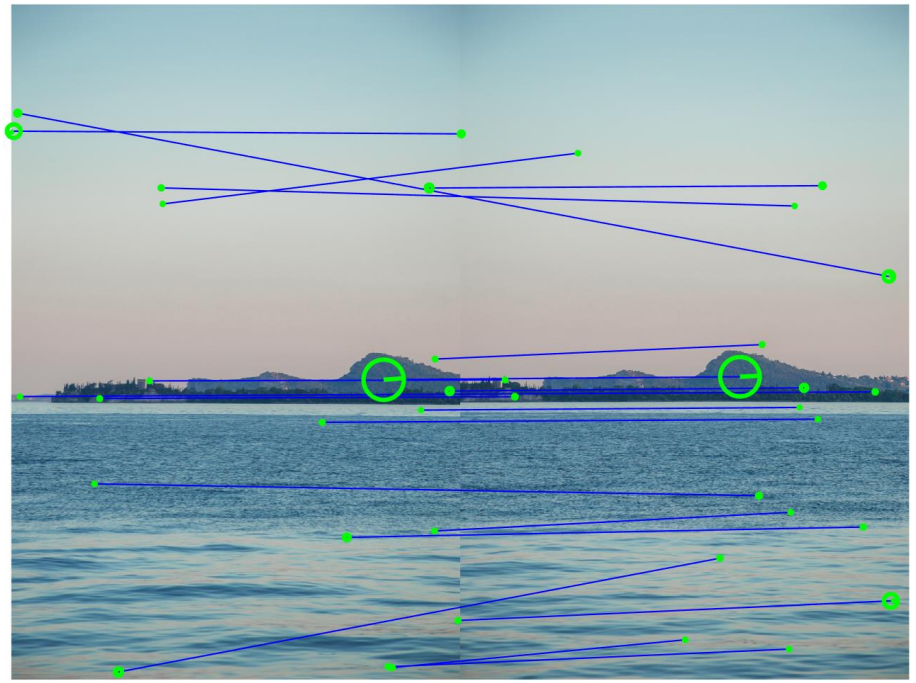
- The point's **descriptor** is a 128-length vector obtained concatenating 16 histograms with 8 bins each (*local* descriptors). The computed descriptions will be used afterwards to compare salient points between the two images.



Salient points detection, description and matching (SIFT)

Salient points matching

- By comparing the salient points descriptors, *matching features* can be detected between the two images.



Feature points extracted using SIFT (only 20 corrispondences were shown)

Salient points detection, description and matching (SIFT)

SIFT – *Robustness*

- SIFT isn't affected by:
  - *Scale:* because salient points are collected at different scales;
  - *Rotation:* because gradients are local characteristics and if the image is rotated, then also the gradient will rotate;
  - *Illumination changes:* because gradients of the image and not pixels' intensity were encoded inside the histograms ;
  - *Perspective deformation:* because local regions are encoded, it depends on the entity of the deformation.

(a) Rotation (b) Slight deformation and scale changes (d) illumination changes (d) Noise disturbance

Robust estimation of the homography matrix (RANSAC)

Homography matrix – *Simple planar scene*

- We choose a world reference frame such that $z = 0$, so that all the points inside the image, lie on the same plane.
- An homography matrix describes the relationship between points on the 3D plane and 2D pixels.

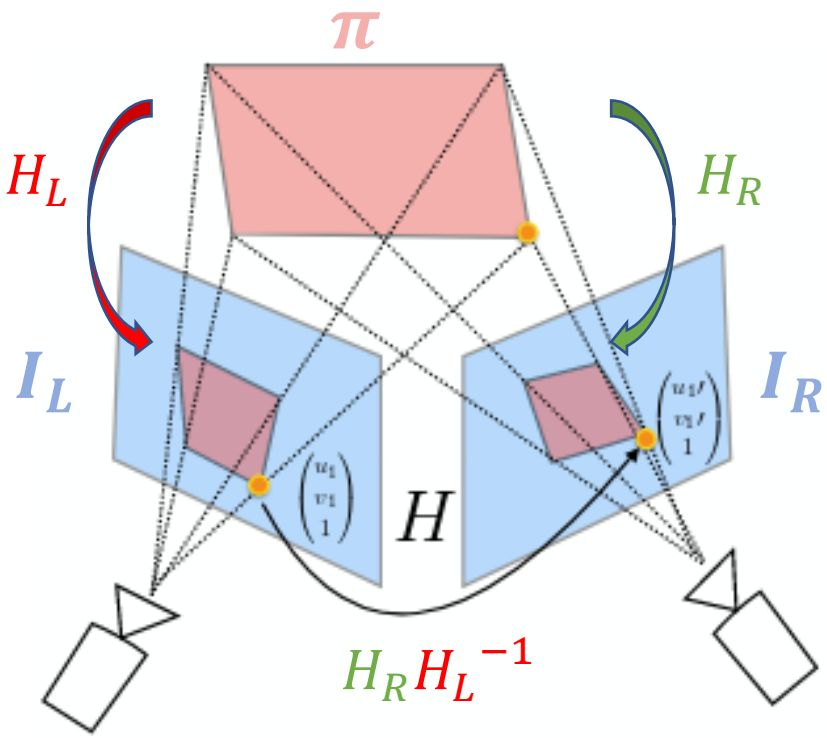
$$\begin{pmatrix} u \\ v \\ 1 \end{pmatrix} = \begin{pmatrix} p_{11} & p_{12} & p_{13} & p_{14} \\ p_{21} & p_{22} & p_{23} & p_{24} \\ p_{31} & p_{32} & p_{33} & p_{34} \end{pmatrix} \begin{pmatrix} x \\ y \\ 0 \\ 1 \end{pmatrix} \rightarrow \begin{pmatrix} u \\ v \\ 1 \end{pmatrix} = \begin{pmatrix} p_{11} & p_{12} & p_{14} \\ p_{21} & p_{22} & p_{24} \\ p_{31} & p_{32} & p_{34} \end{pmatrix} \begin{pmatrix} x \\ y \\ 1 \end{pmatrix}$$

General Perspective matrix (3x4)

Homography matrix

Robust estimation of the homography matrix (RANSAC)

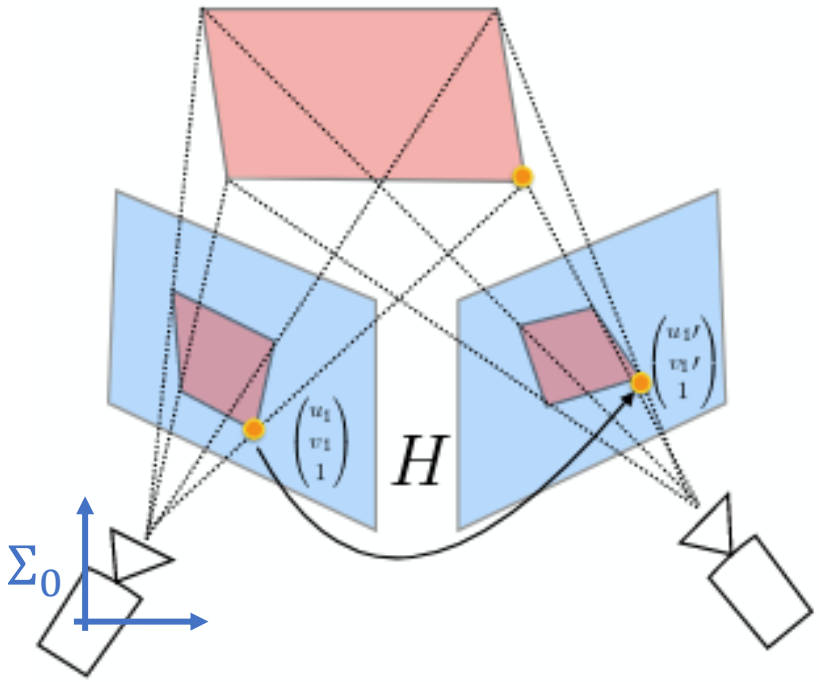
Homography matrix – *Simple planar scene*



- Given the previous equation, pixels among two different images (I_L and I_R) observing the same subject are related by an homography matrix.
- In particular: $m' = H_R H_L^{-1} m$, where m and m' are the 2D pixels of the left and right image respectively.
- Also, $H_\pi = H_R H_L^{-1}$ is the homography of the plane π.

Robust estimation of the homography matrix (RANSAC)

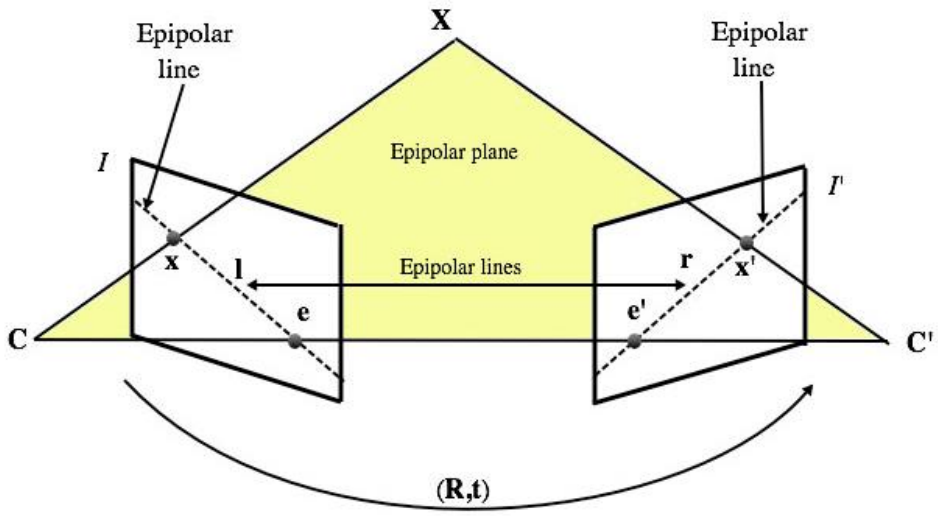
From the Fundamental matrix to the Homography matrix



- Starting from a two cameras stereo setup, let's place the world's reference frame on the left camera (to simplify the computation of P).
- The two Perspective matrices are given by the following equations (left and right cameras): $P = K\,[I|0] = [K|0]$; $P' = K'[R|t]$, where K and K' are the matrices of the internal parameters; R and t are the rotation matrix and the translation vector, that together define the right camera's pose, with respect to the left one.

Robust estimation of the homography matrix (RANSAC)

From the Fundamental matrix to the Homography matrix

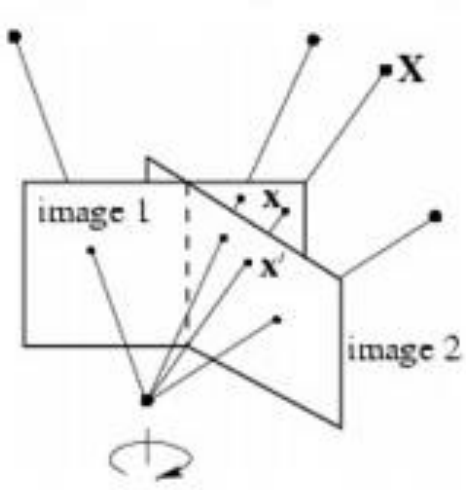


- Now, the epipolar line of m can be computed, using the following definition: $m' = \lambda Q' Q^{-1} m + e'$.
- The above equation can be rewritten as: $\boldsymbol{m' = \lambda K' R K^{-1} m + K' t}$, given that the right epipole $e' = q' - Q' Q^{-1} q$; $q' = K' t$ and $q = \vec{0}$.
- So, two situations can be considered:
  - ***Pure rotational motion***: the translation vector t is null;
  - ***Planar scene:*** generic motion but all the objects lie on the same 3D plane.

Robust estimation of the homography matrix (RANSAC)

Homography matrix – *Pure rotational motion*

- If the motion is a pure rotation, there is no translation, so the pixel to pixel corrispondence can be computed as follows: $\boldsymbol{m' = \lambda K' R K^{-1} m}$. In this case, $\boldsymbol{K' R K^{-1}} = \boldsymbol{H_\infty}$ is called *homography at infinity*.



Such a result can be practically obtained mounting the camera on a tripod and rotating its head.

Robust estimation of the homography matrix (RANSAC)

Homography matrix – *Planar scene*

- Given a set of 3D points M on the world's reference frame (centered on the left camera);
- Given the M' 3D points, representation of M on the right camera's reference frame;
- Given $n^t\widetilde{M} = d$ the equation of the plane, where d is the distance fromthe origin and n^t is the normal to the plane;
- We get that $m \cong K\widetilde{M}$ and $m' \cong K'\widetilde{M}'$.
- To move M from the left to the right camera's reference frame: $\widetilde{M}' = R\widetilde{M} + t$.
- Also, $\widetilde{M}$ belongs to the plane, in fact $n^t\widetilde{M} = d \rightarrow \frac{n^t\widetilde{M}}{d} = 1$.
- After some algebraic manipulation: $\boldsymbol{m' = \boxed{K'\left(R + \frac{t \cdot n^t}{d}\right)K^{-1}}m}$

H_π: *homography of the plane*

Robust estimation of the homography matrix (RANSAC)

Homography matrix – *Planar* / *Rotation*

- To sum up, the relation between *Planar* and *Rotation* homographies is the following:

$$\textit{Planar: } \boldsymbol{m'} = \underbrace{\boldsymbol{K'}\left(\boldsymbol{R} + \frac{\boldsymbol{t}\cdot\boldsymbol{n}^{\boldsymbol{t}}}{\boldsymbol{d}}\right)\boldsymbol{K}^{-\boldsymbol{1}}}_{\boldsymbol{H}_{\pi}}\boldsymbol{m} \qquad \textit{Rotation: } \boldsymbol{m'} = \underbrace{\boldsymbol{K'RK}^{-\boldsymbol{1}}}_{\boldsymbol{H}_{\infty}}\boldsymbol{m}$$

Robust estimation of the homography matrix (RANSAC)

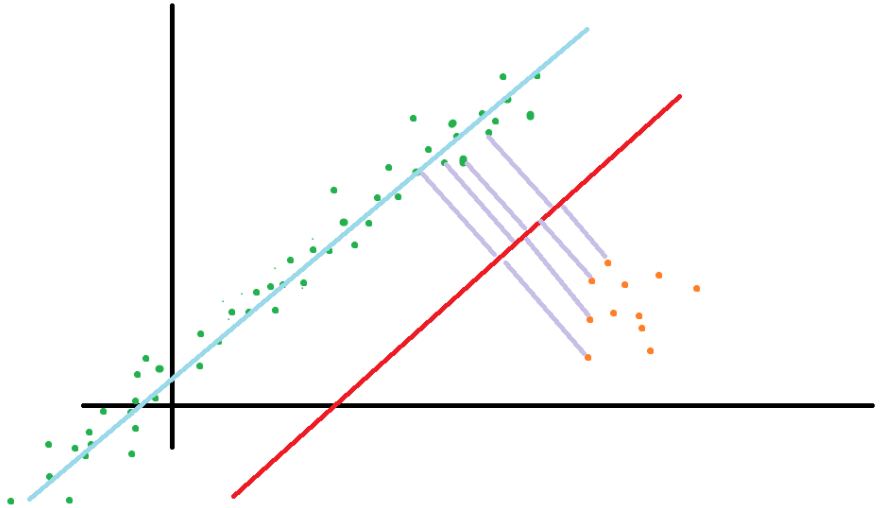
Homography matrix computation

- Given n corresponding points (m_i, m_i'), with $i = 1 \dots n$;
- $m_i' = \boldsymbol{H} m_i$
- $m_i' \times \boldsymbol{H} m_i = 0 \rightarrow [m_i']_\times \boldsymbol{H} m_i = 0$, where $[m_i']_\times$ is the external product matrix
- $vec([m_i']_\times \boldsymbol{H} m_i) = 0 \rightarrow ({m_i}^t \otimes [m_i']_\times) vec(\boldsymbol{H}) = 0$
- This final equation, obtained using the Kronecker product's properties, can be solved using Singular Value Decomposition (SVD).
- If the number of points n is greater than 4, the homography matrix $\boldsymbol{H}$ can be estimated by least squares method.

Robust estimation of the homography matrix (RANSAC)

The RANSAC method – *Line fitting*

- **RANSAC** stands for **RAN**dom **SA**mple **C**onsensus and is used to perform a *robust regression task*. For example, the line fitting problem constists in finding the best fitting line through a set of points including *inliers* and *outliers*. Having a «noisy» dataset a wrong model estimation could be performed, as shown in the picture.



The green dots represent inlier points (or observations), while the orange points represent outliers (noise).
In this case, least squares method was applied and the red line represent the «not so good» fitting line.

Robust estimation of the homography matrix (RANSAC)

The RANSAC method – *Line fitting*

- To make the line fitting robust, RANSAC can be implemented:
  - INPUT:
    - Set of observations, in this case points (x_i, y_i)
    - Error threshold ε
    - Number of iterations
  - OUTPUT:
    - Model's parameters $\hat{\vartheta}$

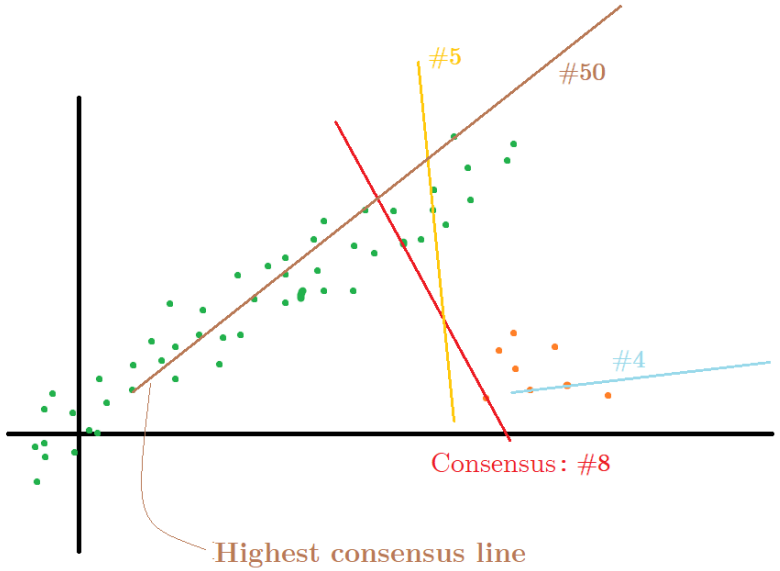
1. Take a random sample of with p elements from the observation set;
2. From this subset estimate the parameters vector $\hat{\vartheta}_j$ with $j = 1 \dots p$;
3. Compute the consensus of the current model $\hat{\vartheta}_j$ with the following formula:

$$\#\left\{y_i : \left(y_i - f\left(x_i, \widehat{\vartheta_j}\right)\right) < \varepsilon\right\}$$

Robust estimation of the homography matrix (RANSAC)

The RANSAC method – *Line fitting*

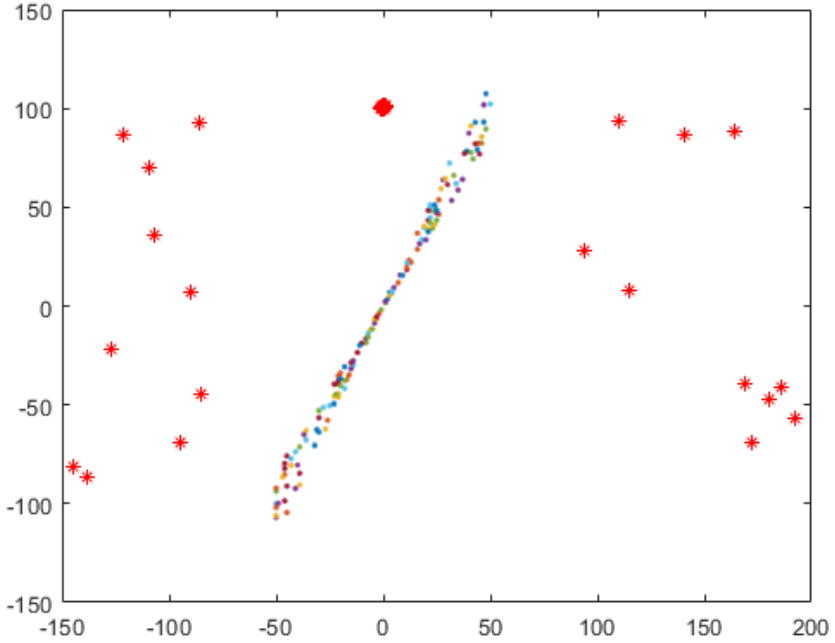
4. Repeat the previous steps until the choosen number of iterations is reached;
5. To obtain the final model's parameters, estimate the regression only using the inlier observations of the largest consensus computed model (using Least Squares method).

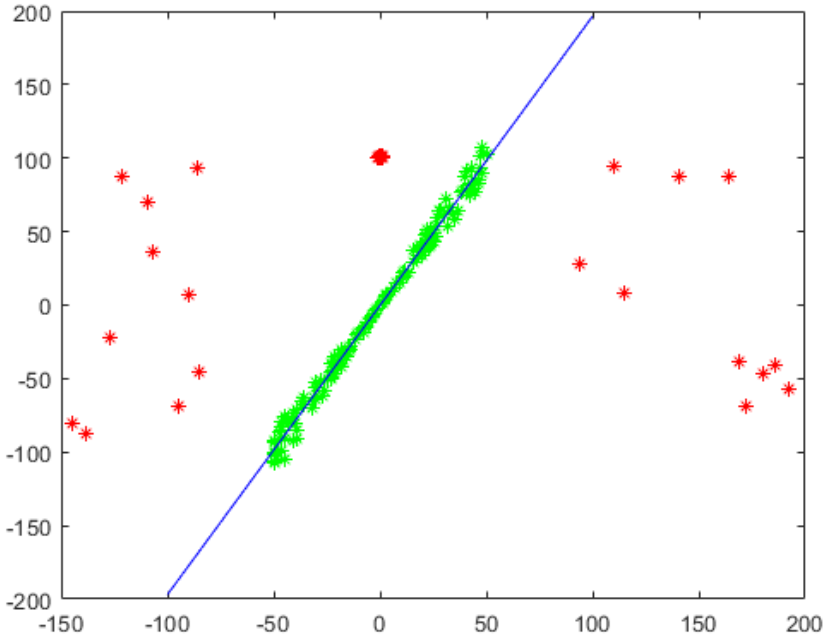


Robust estimation of the homography matrix (RANSAC)

The RANSAC method – *Line fitting*

- This is the result of an implemented MATLAB script that clearly shows the good achievable performances.





On the ***left*** the set of randomly generated points;
On the ***right*** the fitting line (blue), the outliers (red) and inliers (green) points.

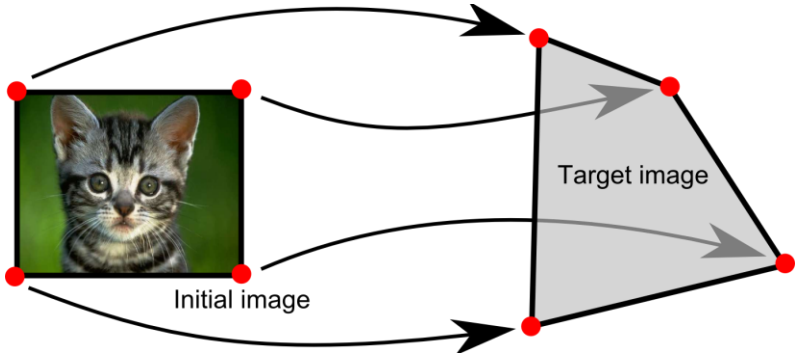
Robust estimation of the homography matrix (RANSAC)

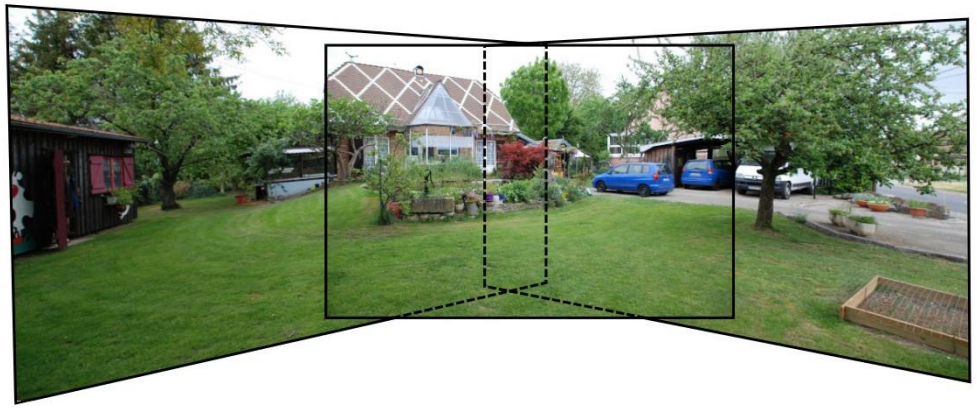
The RANSAC method – *Robust homography estimation*

- The same algorithm can be applied for estimating a robust homography matrix, not heavily influenced by outlier corrispondences.
- The only difference is the type of estimated model: the f function now depends on m_i' and H. Of course, the homography matrix H is the model that is beeing estimated.
- The model's error will be measured by this formula: $|m_i - Hm_i'| < \varepsilon$.
- Moreover, if the number of inliers is greater than 50% the method is fully robust.

Image warping, alignment and stitching

- At this point of the pipeline, supposing that we're creating a 2 images mosaic, the right image should be distorted according to the estimated homography matrix. The homography is used to bring all the right image's pixels on the left image's reference frame (`MATLAB function: imwarp_prof`).





- Then, the images should be the same size, so *bounding boxes* for the warped image are computed (`MATLAB function: adjustSizes`). Both images will include black borders in order to have the same size.

Image warping, alignment and stitching

- Now, the *images can be aligned* making sure that the two reference frames have the same origin (`MATLAB function: getMosaic`).
- Afterwards a color blending strategy has to be implemented, to decide the overlapping pixels' colors (`MATLAB function: colorBlendingEasy`).
- To notice that, the operations perfomed on the images have to be done *three times if color images (RGB) are used*. For clarity reasons, inside the developed script the functions are called once for every channel and the mosaic is then merged into RGB.
- Moreover, at the stitching, if there are leftover black borders that are not useful (and make the image bigger), they're removed (`MATLAB function: minimizeBlackSpace`).

Trivial approach to color blending

- As previously mentioned, overlapping pixels between the two images must be somehow colored.
- The implemented algorithm is very simple:
  - if the two considered pixels are black, they are kept black (they belong to the added borders);
  - if one of the two pixels is colored and the other is black, the final pixel will take that color, ignoring the black;
  - if both pixels aren't black the final color will be the mean of the two.
- Despite the approach being so simple, it shows good results if the image resolution isn't too low and if there aren't strong exposure changes between the images.

Project objectives and developed features

- Automatic estimation of ***correspondences*** (using the SIFT library)
- Robust estimation of the ***homography*** between every two pictures (using RANSAC)
- Trivial ***color blending*** strategy (if only one of the two pixels is colored, use that color, if both aren't black, blend them 50/50)
- ***Image warping*** (using the imwarp function and trying also the SIFT prewritten pipeline) in order to adapt and align the two images
- RANSAC and SIFT ***parameters***' setting (number of inliers, iterations, threshold...)
- Three ***image stitching strategies*** (mosaicing performed with different orders)
- ***Splitting*** option of the mosacing procedure
- ***Cropping*** option of the final mosaic

Project objectives and developed features

Three image stitching strategies

- As mentioned, mosaicing is an iterative process where pair of images are stitched together in order to enlarge the field of view. Depending on the mosaicing order, the final resulting image can have a distortion towards left, right, up or down.
- The user can choose among three different orders, to achieve the most pleasent result:
  - **first2last:** images are stitched starting from the first picture that has been taken to the last one;
  - **last2first:** images are stitched starting from the last picture that has been taken to the first one;
  - **fromCenter:** images are stitched starting from central picture, enlarging the field of view, step by step, going from center to borders (first and final pictures). In this case, the resulting image will be more similar to the first2last or to the last2first, depending on the number of images (odd or even).
- In the next slides, practical examples will be shown, to better explain the concept.

Project objectives and developed features

Three image stitching strategies

In this case, **first2last** option was selected. As can be seen, most of the distortion is on the left side.

Project objectives and developed features

Three image stitching strategies

In this case, **last2first** option was selected. As can be seen, most of the distortion is on the right side.

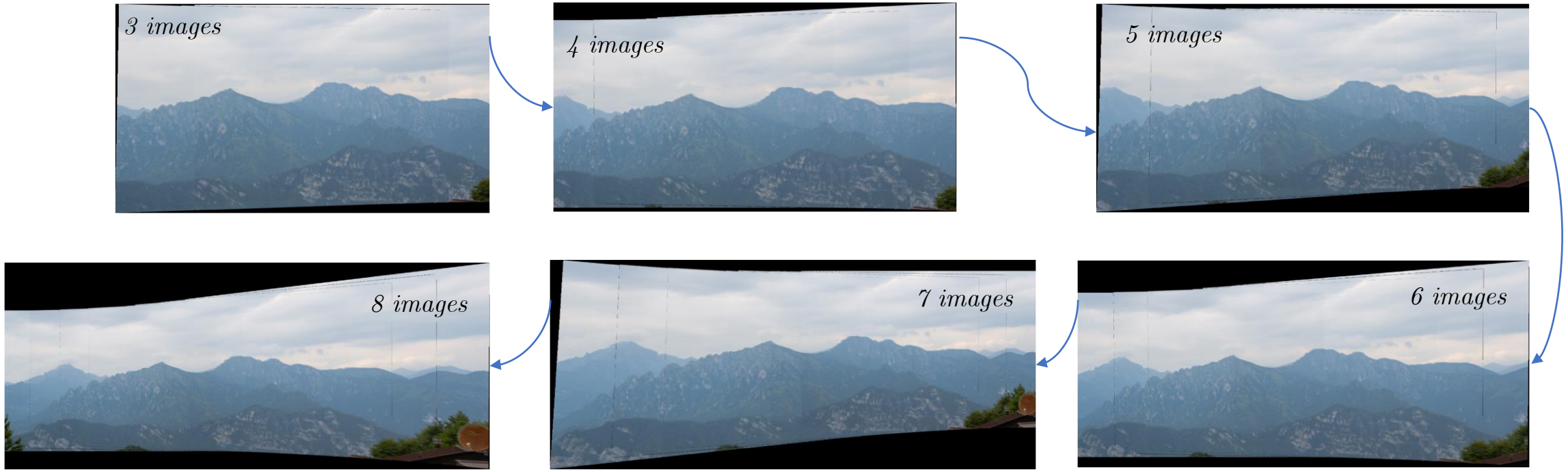
Project objectives and developed features

Three image stitching strategies

In this case, **fromCenter** option was selected. As can be seen, most of the distortion is on the left side, because the number of images (15) is odd.

Project objectives and developed features

Three image stitching strategies



To better show how the **fromCenter** option works, these are the followed steps until all the images are combined.
So, if the number of images is odd, the final distortion will be similar to the one achieved from the first2last option.

Project objectives and developed features

Splitting option of the mosacing procedure

- A more effective approach to the «fromCenter strategy» is given by the ***splitting option***. In fact, the idea is to create two separate panorama images (left-center; center-right or up-center; center-down) and stitch them together.
- The resulting mosaic will have a so called «butterfly» shape, beeing distorted on the extreme sides, while sharp and smaller on the central part.
- This option can be choosen in conjunction with left2right or right2left, but isn't so compatible with the fromCenter option.
- One of the main reasons why this option was developed was to mitigate the «not enaugh RAM memory problem» in addition to the overdistorted panoramas that are created if the number of images grows (>10-15 images).

Project objectives and developed features

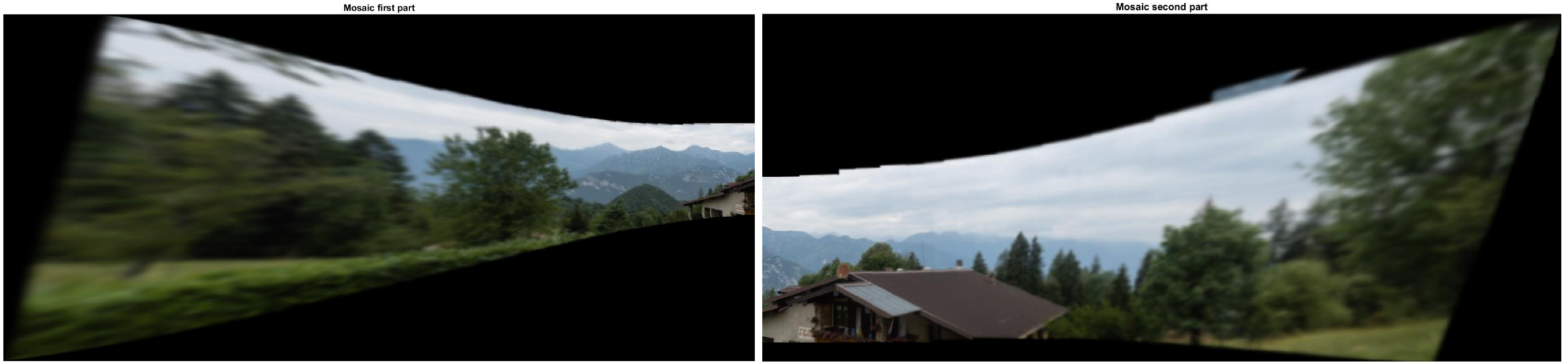
Splitting option of the mosacing procedure

This mosaic is the result of **40 images** combined together. With my code it wouldn't be possible to achieve such a result, without using the ***split*** option. The out of focus effect on the borders is due, not only to the warping, but also to the heavy resizing of the images (to avoid memory issues).

Project objectives and developed features

Splitting option of the mosacing procedure

The splitting process consists in creating the first and second parts of the mosaic, independently, to then merge them together.

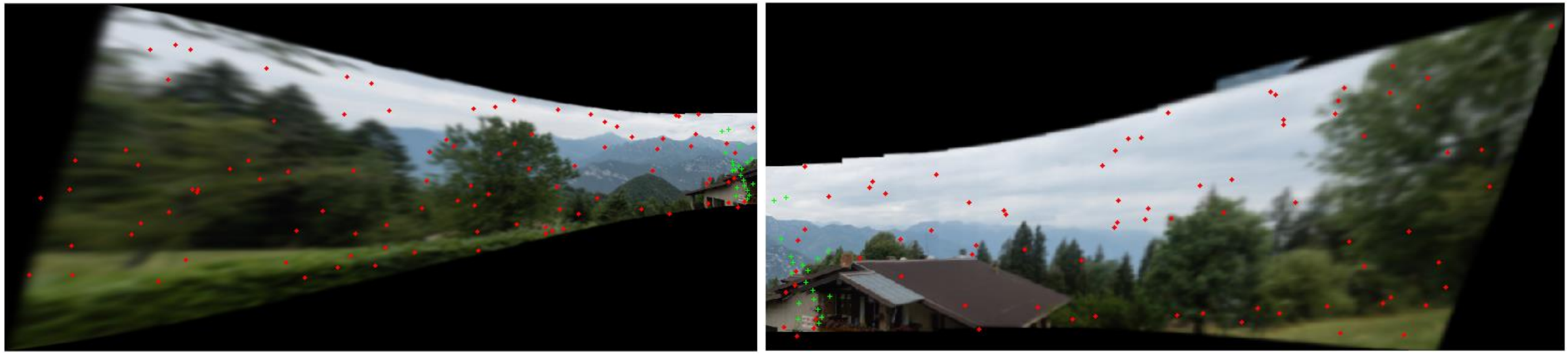


The first (left) and second (right) parts that will compose the final mosaic are shown.

Project objectives and developed features

Splitting option of the mosacing procedure

Once the correspondent points between the two mosaics are found, using SIFT, the final panorama is created. To notice that a mask could be applied to make the process faster and more robust, even though it has not been implemented inside the script.



Inliers (green) and outliers (red) from the SIFT salient points detection process are shown.

Project objectives and developed features

Cropping algorithm

- As explained, during the warping process, black areas are added around the images. Even though the script tries to minimize them, the final panorama will always have borders that steal attention from the photograph.
- An iterative procudure tries to retrieve the largest «real» image borders.
- I decided to implement it using MATLAB polygons:
  1. *Create an irregular polygon whose vertices are the borders of the image* (not black/NaN pixels)
  2. *Find the largest inscribed rectangle in the irregular polygon* (that will be used to crop the mosaic)
- To do this, the user is asked to provide the panorama orientation (vertical/horizontal) and where the least amount of distortion is, inside the image. These are information that I thought would be reasonable to ask the user, being them not so trivial to determine a priori.

Project objectives and developed features

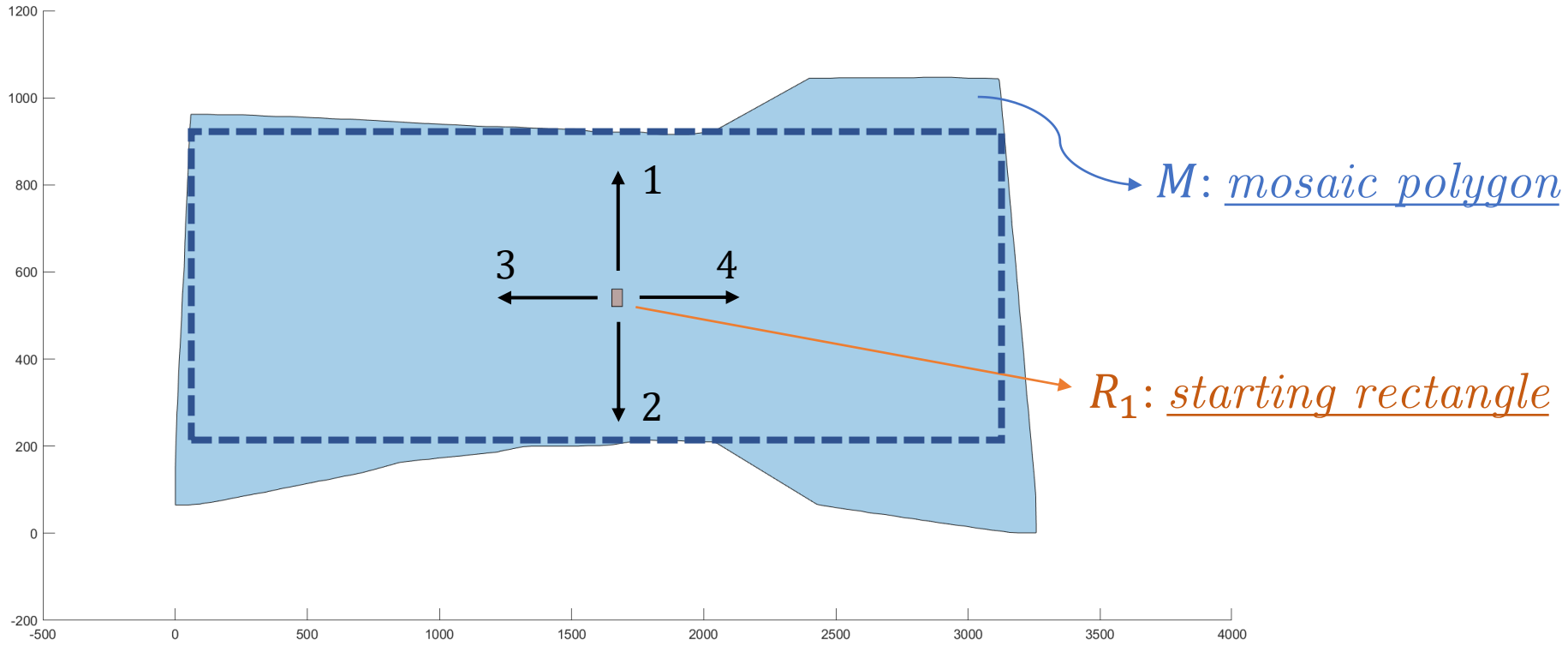
Cropping algorithm

- In brief, the algorithm works as follows (let's assume it's a horizontal panorama):

1. The mosaic polygon *"M"* is created and a small rectangle *"R_1"* is drawn near the centroid;
2. R_1 is extended horizontally (left and then right) until it comes out of M (checked with intersection and difference polygon functions). R_1's vertices are kept in memory;
3. Another rectangle R_2 is drawn, according to the user information, inside the least distorted section (typically, for horizontal mosaics, "center");
4. R_2 is extended vertically (up and then down) until it comes out of M (checked with intersection and difference polygon functions). R_2's vertices are kept in memory;
5. The mosaic is cropped according to R_1 horizontal boundings and to R_2 vertical ones.

Project objectives and developed features

Cropping algorithm



The above drawing is trying to schematize the cropping algorithm's previously described steps.

Brief discussion about Photoshop (CS6 2018) and MATLAB's Computer Vision Toolbox results

- As a small part of the project, the two mentioned software have been explored to compare the obtained mosaicing results.
- ***Photoshop*** offers the "*Photomerge*" function that allows to combine multiple photos into a single continuous image. On the official website there are guidelines about how the images should be taken and they resonate with the theory behind this project (keep the same focal length, exposure and aperture settings, overlap the images by at least 40%, use a rotating tripod).
- In particular, several merging layouts can be chosen: "Auto", "Perspective", "Cylindrical", "Collage", "Reposition". "Auto" chooses between "Perspective" (reference frame on the center image) and "Cylindrical" (suited for wide panoramas).

Brief discussion about Photoshop (CS6 2018) and MATLAB's Computer Vision Toolbox results

Three of the previously mentioned mosaicing layouts.

- The blending inside Photoshop is done using layers and masks, the fundamentals of this software. So, the image stitching process is efficient and robust.
- **MATLAB's Computer Vision Toolbox** offers several prewritten functions to deal with salient points detection, image warping, color blending and mosaicing in general. To have an additional point of view, I tried to modify the original image stitching tutorial, to make use of my homography estimation process (and correspondences detection) and use their image warping pipeline. In particular, through the main livescript there are 3 options: SURF+C.V.Homography; SIFT+myHomography; SIFT+C.V.Homography

Brief discussion about Photoshop (CS6 2018) and MATLAB's Computer Vision Toolbox results

- As an interesting note, the *"Feature Based Panoramic Image Stitching"* tutorial uses the SURF (Speeded Up Robust Features) method to detect salient points. Citing Wikipedia: *"the standard version of SURF is several times faster than SIFT and claimed by its authors to be more robust against different image transformations than SIFT".*
- Moreover, instead of iteratively working on the images (as I do in my script) only the transformation matrices from a reference frame to the other (homographies) are computed to be later used on the panorama creation. I imagine that this approach could speed up the overall process.
- Other than that, the prewritten script and mine have a similar high-level structure.

Mosaicing examples and comparisons

- In the following slides several examples are shown and compared to a professional image processing software (Photoshop CS6) and a MATLAB script available on the official website.
- Also, "bad results" achieved from the developed script are considered and motivated.

This is the end result of the developed MATLAB script, that stitches all the images (in this case 33) and shows the cropped mosaic, if requested.

Mosaicing examples and comparisons

Maderno bay area (33 vertical images - reflex)

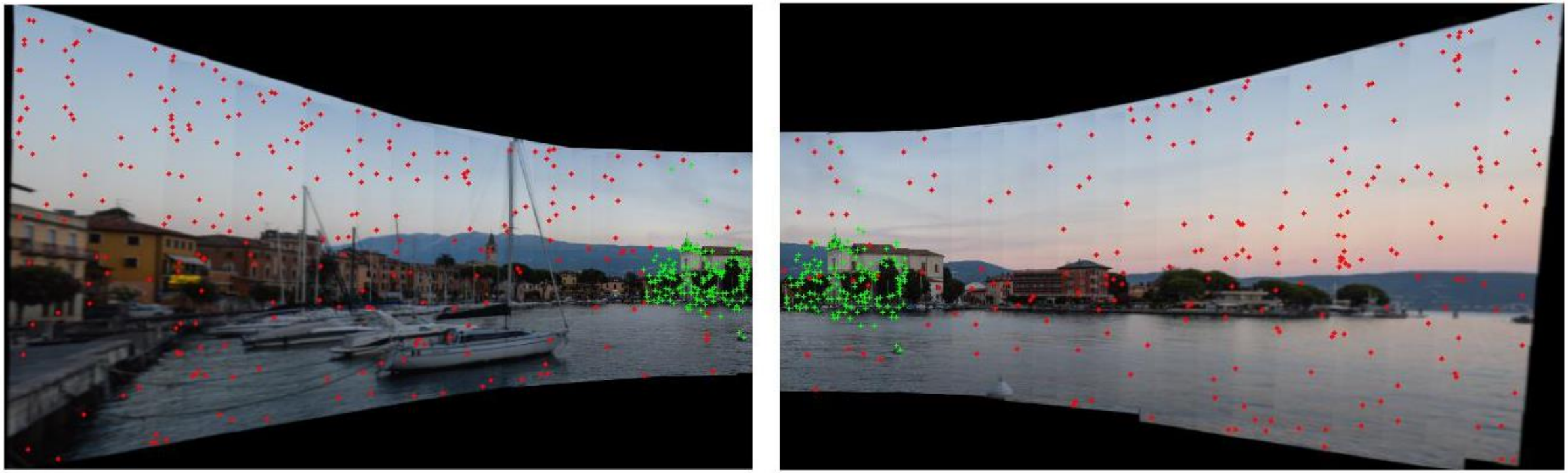
Mosaic first part

Mosaic second part

In this example, using «split mode» the first and second mosaic are shown.

Mosaicing examples and comparisons

Maderno bay area (33 vertical images - reflex)

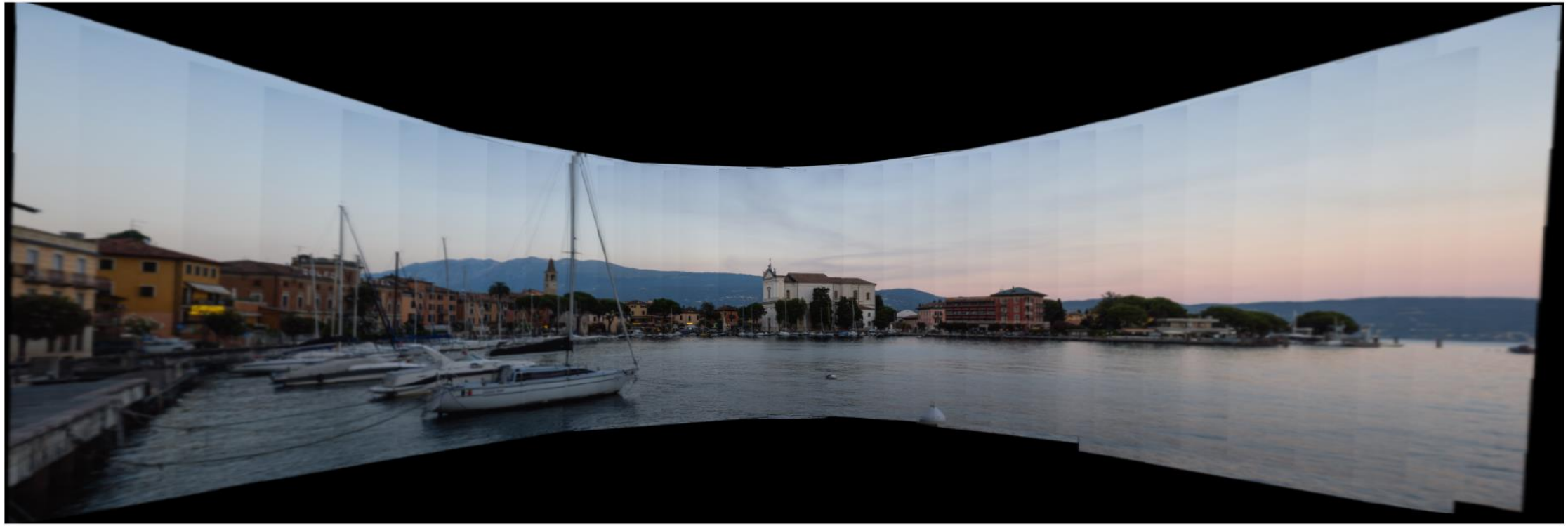


Next, by default, inliers (green dots) and outliers (red dots) corrispondences between the 2 mosaics are shown.

Mosaicing examples and comparisons

Maderno bay area (33 vertical images - reflex)

Full mosaic (First+Second)



The full panorama image is shown.

Mosaicing examples and comparisons

Maderno bay area (33 vertical images – reflex, lens correction)

Full mosaic (First+Second)

In this case, all 33 images were **preprocessed with Lightroom**, **enabling lens correction profile** for my Nikon lens.
Not to repeat myself, only the full mosaic is shown

Mosaicing examples and comparisons

Maderno bay area (33 vertical images – reflex, lens & upright corrections)

Full mosaic (First+Second)

In this case, all 33 images were **preprocessed with Lightroom, enabling lens correction profile** for my Nikon lens **and automatic upright corrections.** Not to repeat myself, only the full mosaic is shown

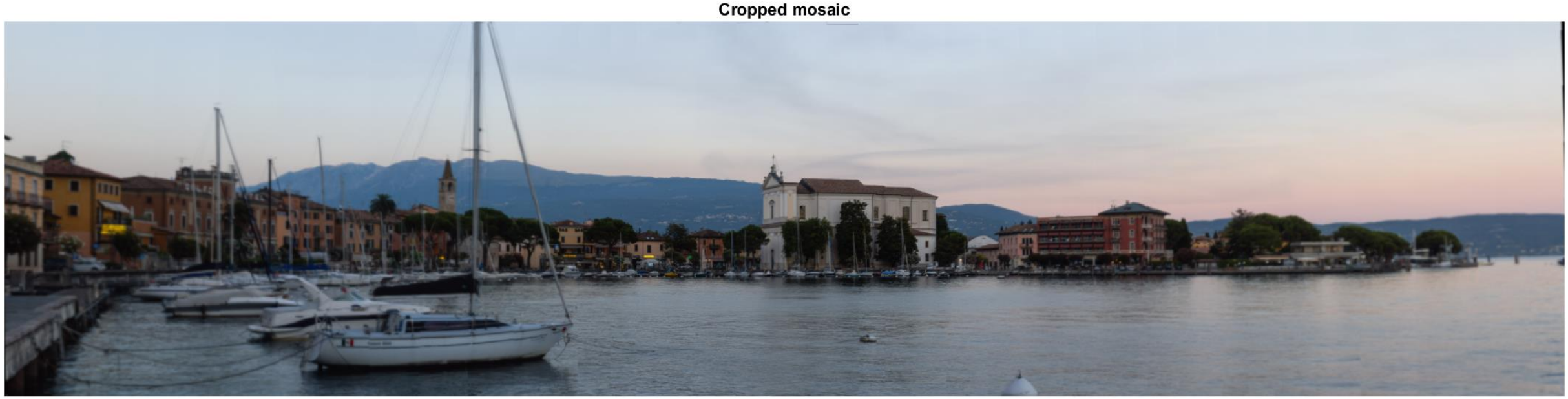
Mosaicing examples and comparisons

Maderno bay area (the 3 cropped versions)

If requested by the user, the cropped version of the mosaic, that maximizes meaningful information, is shown.
In this slide, the 33 **«standard» images** mosaic is shown.

Mosaicing examples and comparisons

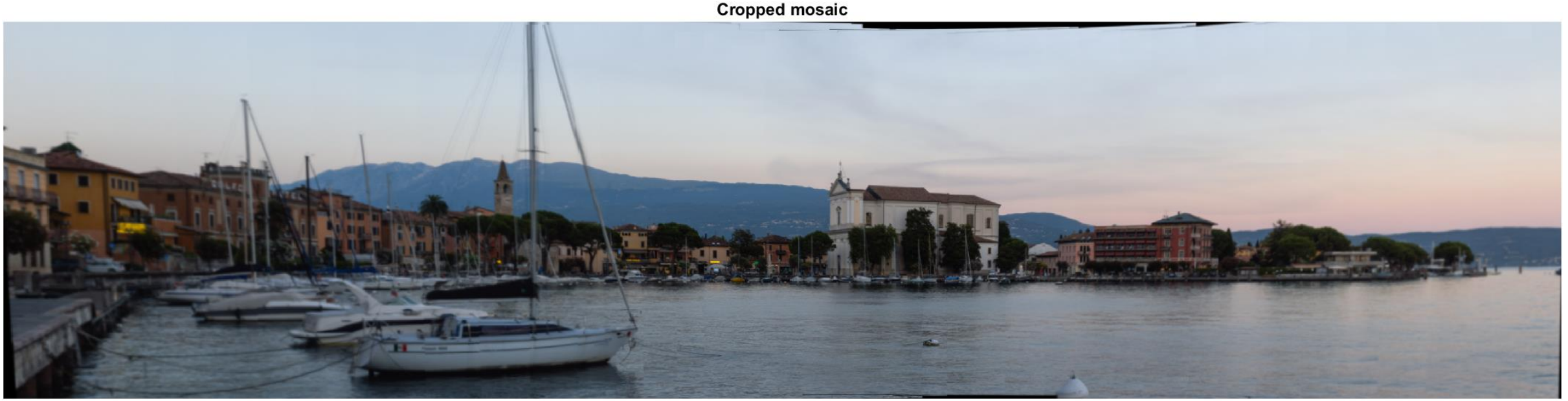
Maderno bay area (the 3 cropped versions)



If requested by the user, the cropped version of the mosaic, that maximizes meaningful information, is shown.
In this slide, the 33 preprocessed images (**only lens correction**) mosaic is shown.

Mosaicing examples and comparisons

Maderno bay area (the 3 cropped versions)



If requested by the user, the cropped version of the mosaic, that maximizes meaningful information, is shown.
In this slide, the 33 preprocessed images (**lens correction + automatic upright correction**) mosaic is shown.

Mosaicing examples and comparisons

Maderno bay area (Photoshop's *photomerge*)

In this case, the 33 original images have been processed through the «**photomerge**» option inside Photoshop. The stitching strategy has been left to Automatic, meaning that the software does the optimal choice, in this case, «**Cylindrical layout**».

Mosaicing examples and comparisons

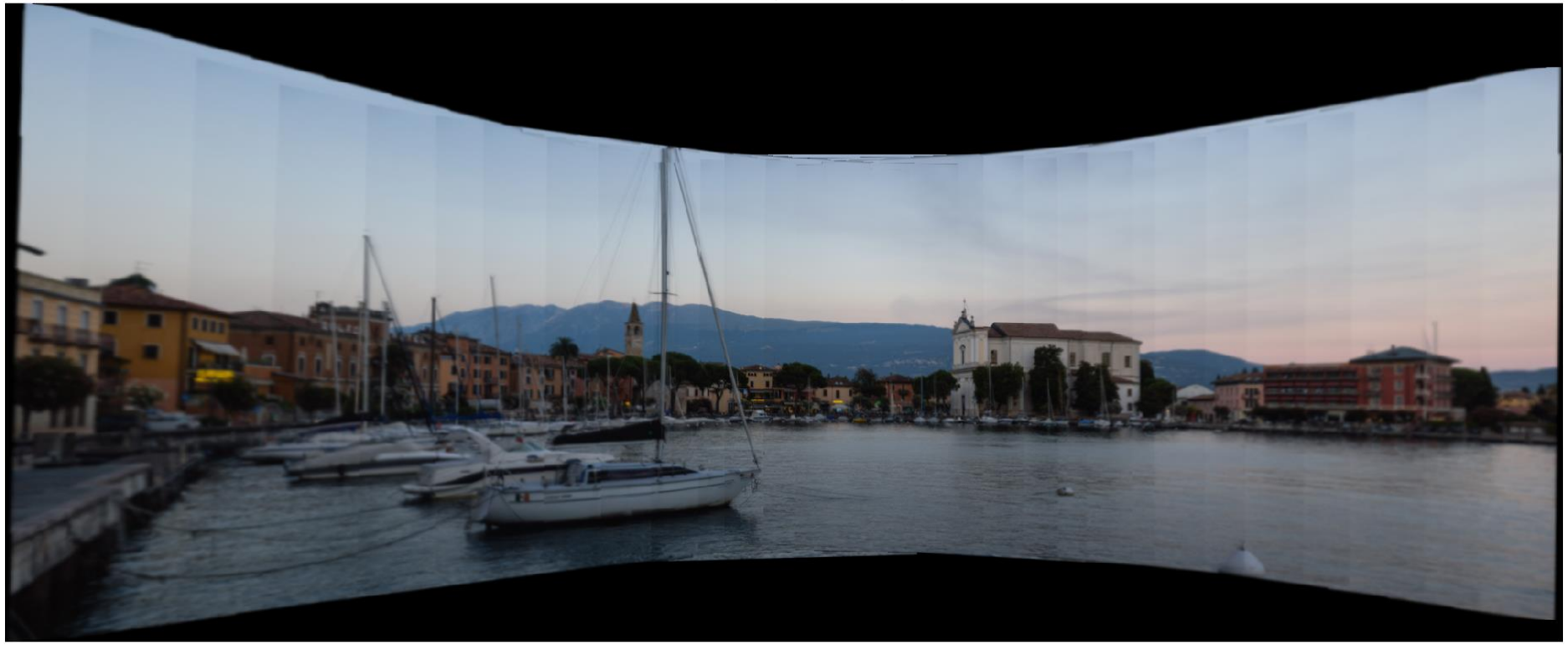
Maderno bay area (Photoshop's *photomerge*)

In this case, the 33 preprocessed images (lens correction + automatic upright correction) have been processed through the «photomerge» option inside Photoshop. The stitching strategy has been left to **Automatic**, meaning that the software does the optimal choice, in this case, «*Perspective layout*».

Mosaicing examples and comparisons

Maderno bay area (my MATLAB project)

Full mosaic (First+Second)



To make the comparison fair, only **25 images** were used. In this case, the *developed software* enabling *splitting option* was used.

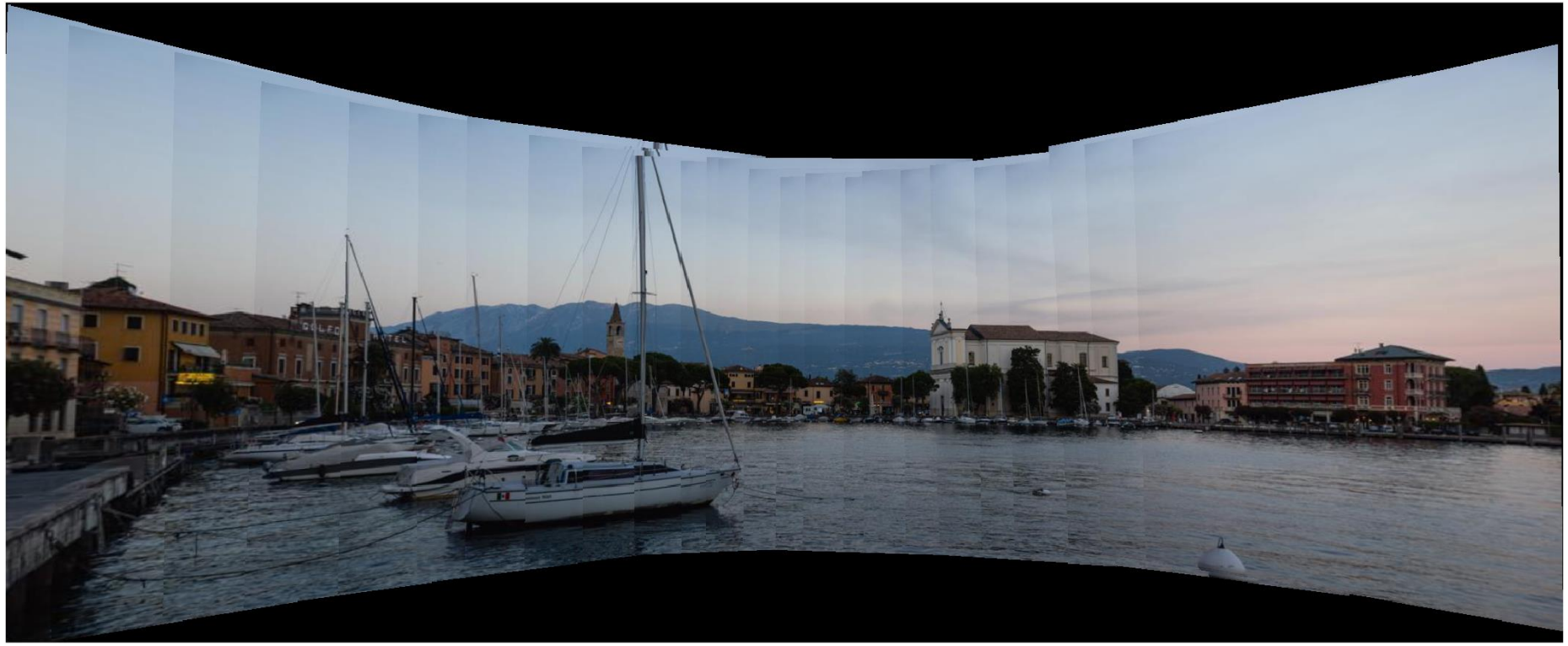
Mosaicing examples and comparisons

Maderno bay area (C.V.Toolbox – SIFT & myHomography)

First **25 images** mosaic (speed/memory problems), with the same resolution of the MATLAB project's script. This is the result of using *SIFT method* in conjunction with *my homography estimation function* and the rest of the tutorial's pipeline.

Mosaicing examples and comparisons

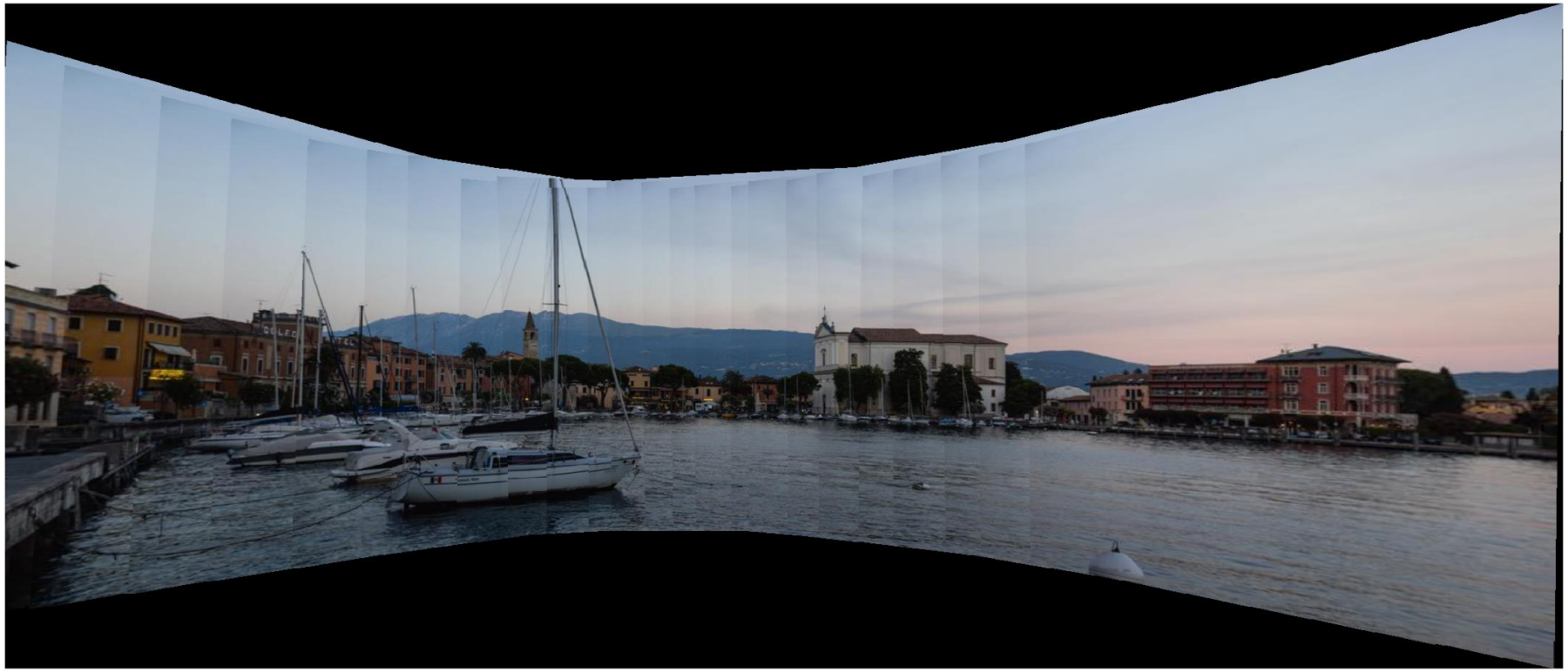
Maderno bay area (C.V.Toolbox – SIFT & C.V.Homography)



First **25 images** mosaic (speed/memory problems), with the same input image resolution of the previous examples. This is the result of using *SIFT method* in conjunction with the *estimateGeometricTransform2D* MATLAB function and the rest of the tutorial's pipeline.

Mosaicing examples and comparisons

Maderno bay area (C.V.Toolbox – SURF & C.V.Homography)



First **25 images** mosaic (speed/memory problems), with the same input image resolution of the previous examples. This is the result of using *SURF method* in conjunction with the *estimateGeometricTransform2D* MATLAB function and the rest of the tutorial's pipeline.

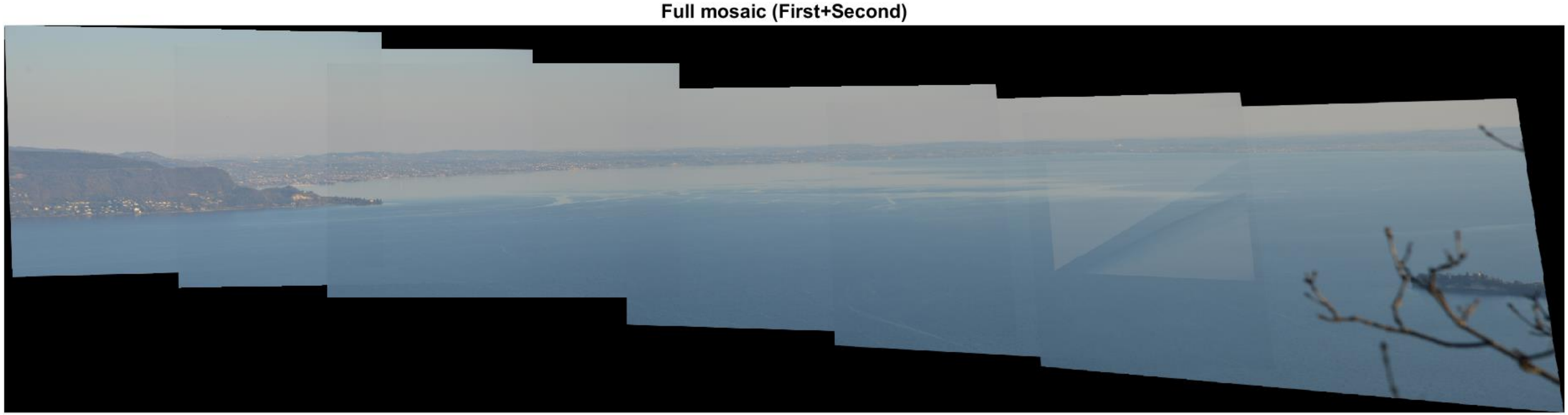
Mosaicing examples and comparisons

Maderno bay area (comments)

- This first batch of comparison shows that the **developed MATLAB script** isn't worst than the **Computer Vision Toolbox tutorial**. Performances can change depending on the homography estimation parameters. Sometimes one single "not so good" homography can lead to small or big artifacts in the final image. *I tried to run the scripts until a pleasant result was achieved.*
- Input image resolution can play a big role on the resulting mosaic. The number of correspondences, the homography goodness and the "out of focus" effect caused by the warping process depend on it. Also, color blending is maybe worst in the tutorial's code than in my script (or at least comparable).
- SURF method seems to work good and faster than SIFT even if I haven't deepened too much its behavior.
- **Photoshop**, of course does the best job, even using "Auto" layering mode, correcting the lens distortion and vignetting. It's hard to notice the overlayed parts.
- Interestingly, Photoshop doesn't benefit by pre-corrected (Lightroom) images, while *my script does.* My cropping method seems to work fine, even if not perfectly.

Mosaicing examples and comparisons

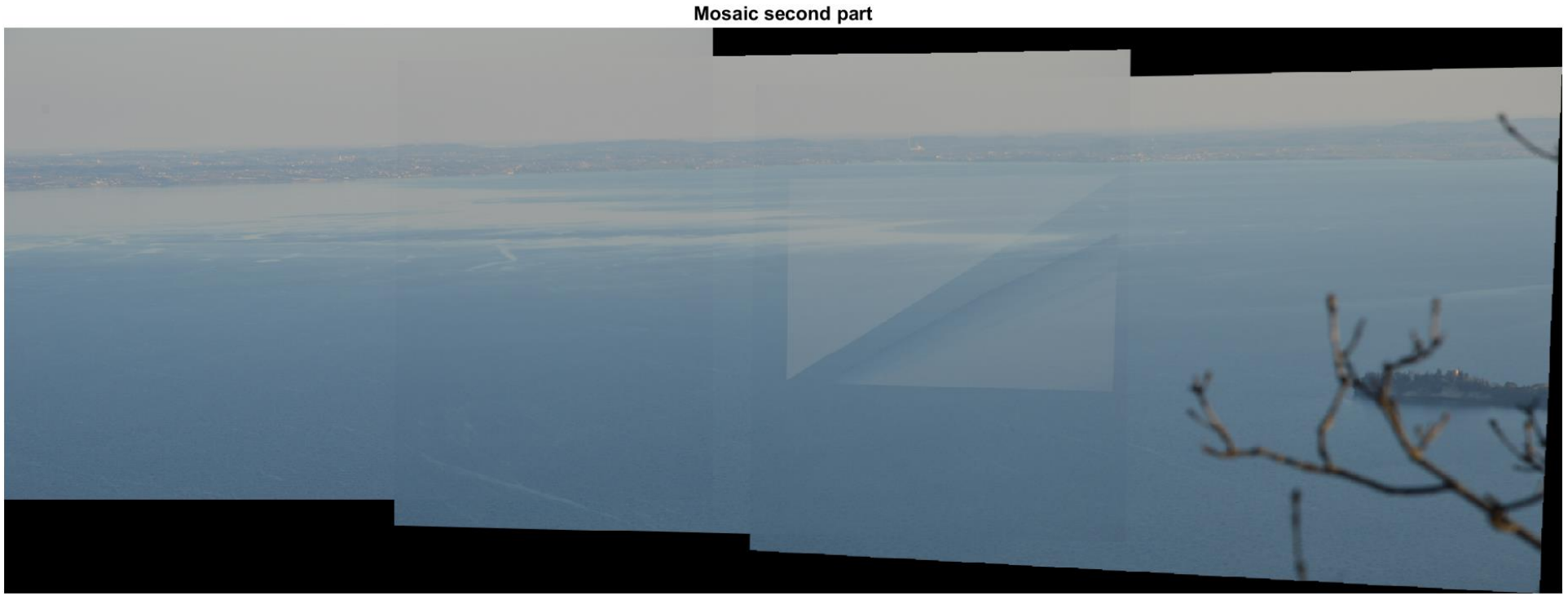
Garda Lake (7 horizontal images – reflex)



The above picture shows a strangely **difficult** mosaicing (only 7 images), because of the *water and sky wide presence*, that make both corrispondent points detection and homography estimation processes complex. This result was achieved tweaking resolution, threshold and iteration parameters.

Mosaicing examples and comparisons

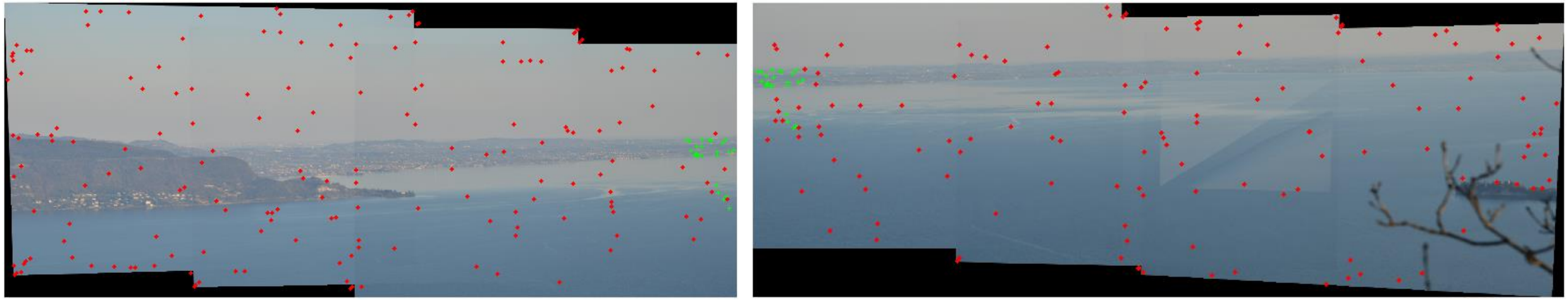
Garda Lake (7 horizontal images – reflex)



Taking a closer look at the second part of the mosaic (splitting mode was selected), an artifact caused by a bad estimated homography can be seen. As mentioned, water and in general low contrast areas or moving parts can damage the whole process.

Mosaicing examples and comparisons

Garda Lake (7 horizontal images – reflex)



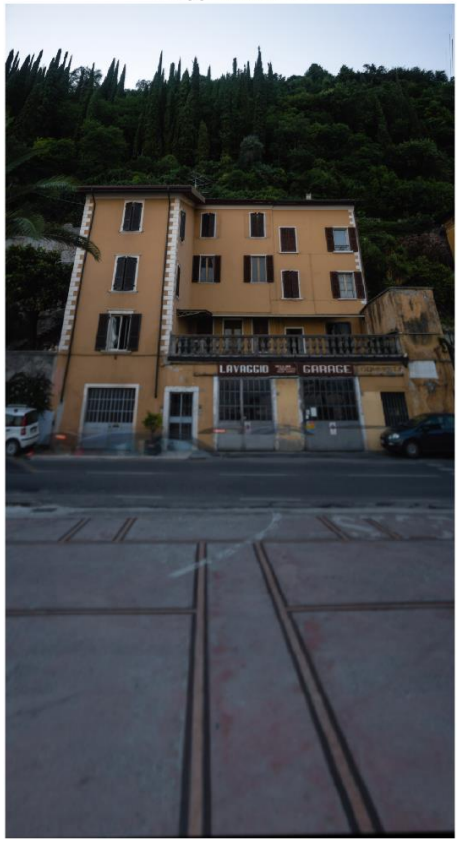
Also, resolution was set to 1000px on the wider edge and the number of iterations was set to 5000, while keeping a threshold of 5 during the homography estimation process of the final mosaic. What I noticed is that just 1 or few outliers detected as inliers far from the central portion can cause the final mosaic to fail. To solve this isssue a mask could be implemented. In the developed software there's also the option to decrease the number of used inliers by the robust homography estimation function.

Mosaicing examples and comparisons

Car repair shop (33 horizontal images – reflex)

Full mosaic

Cropped mosaic



The shown figures refer to a vertical panorama, result of the ***first2last*** mosaicing strategy (without split). As can be seen, most of the distortion is on the lower part of the image.

Mosaicing examples and comparisons

Car repair shop (33 horizontal images – reflex)

Full mosaic

Cropped mosaic

The shown figures refer to the same vertical panorama, result of the ***last2first*** mosaicing strategy (without split). As can be seen, most of the distortion is on the upper part of the image. In my opinion the overall proportions are more realistic in this case, if compared to the previous one.

Mosaicing examples and comparisons

Car repair shop (C.V.Toolbox – SIFT & myHomography)

The shown mosaic is the result of using *SIFT method* in conjunction with *my homography estimation function* and the rest of the tutorial's pipeline.
In this case, all 33 images were used.

Mosaicing examples and comparisons

Car repair shop (C.V.Toolbox – SIFT & C.V.Homography)

The shown mosaic is the result of using *SIFT method* in conjunction with the *estimateGeometricTransform2D* MATLAB function and the rest of the tutorial's pipeline. In this case, all 33 images were used.

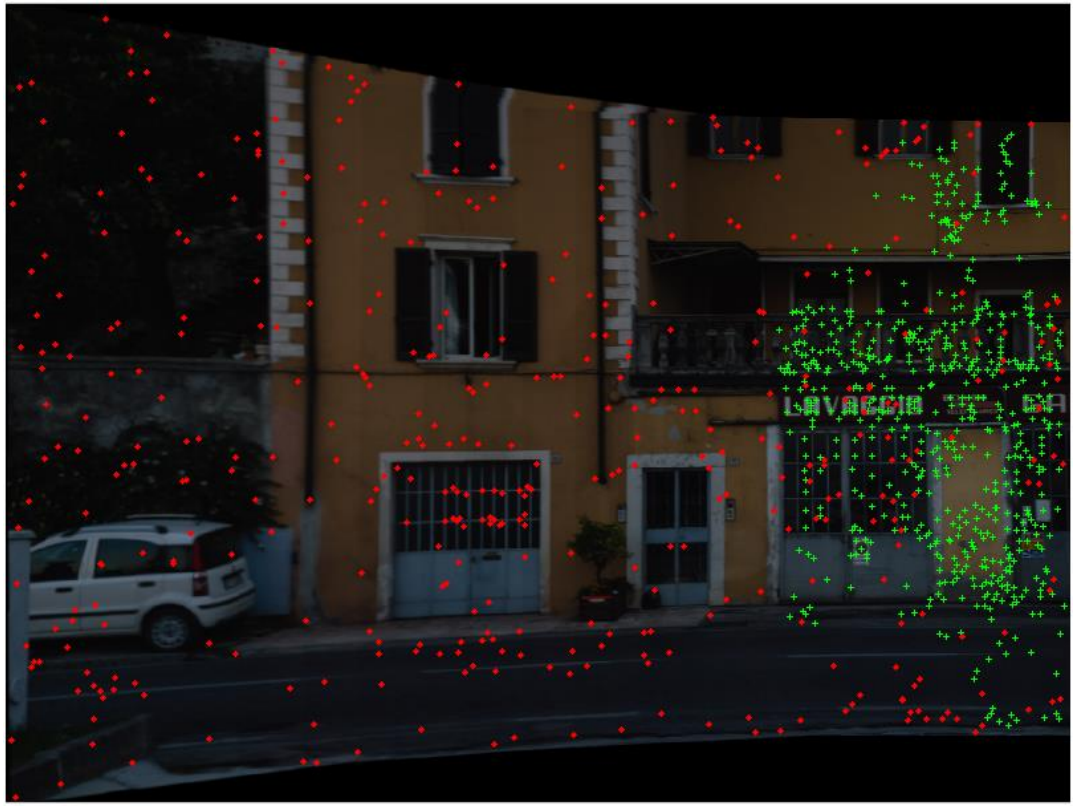
Mosaicing examples and comparisons

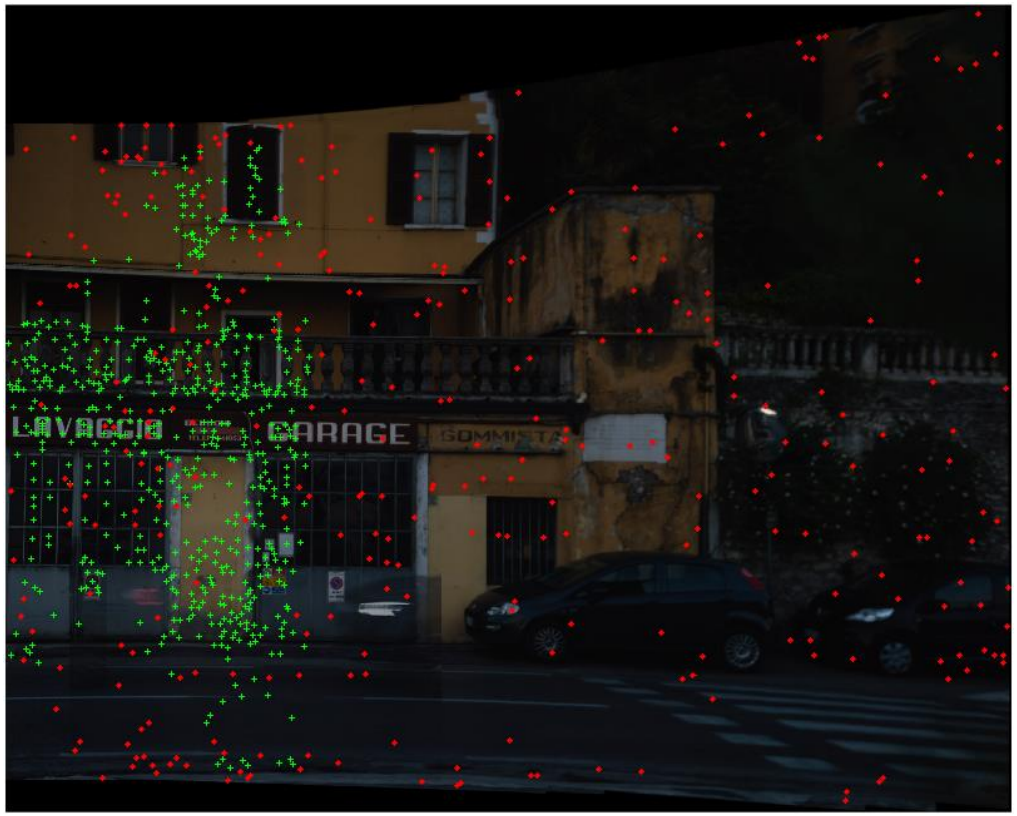
Car repair shop (C.V.Toolbox – SURF & C.V.Homography)

The shown mosaic is the result of using *SURF method* in conjunction with the *estimateGeometricTransform2D* MATLAB function and the rest of the tutorial's pipeline. In this case, all 33 images were used.

Mosaicing examples and comparisons

Car repair shop (23 vertical images – reflex)





In this case, an **underexposed** view is used. Performances aren't really affected: salient point detection and color blending process work fine.

Mosaicing examples and comparisons

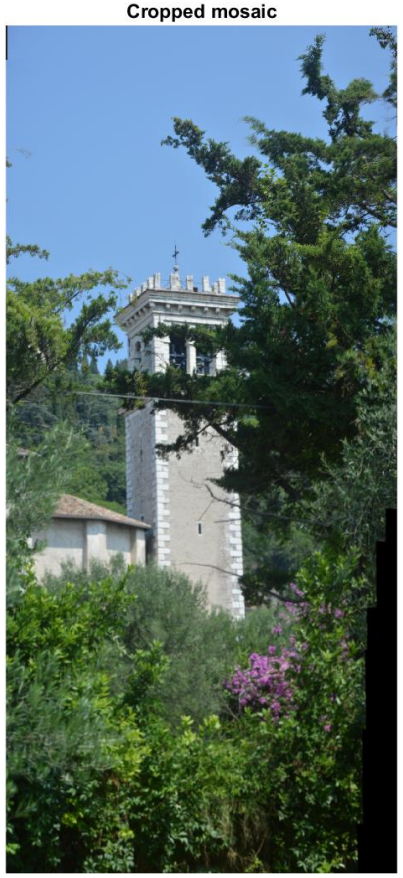
Car repair shop (23 vertical images – reflex)

Full mosaic (First+Second)

To better see the end result, exposure was significantly increased. As can be clearly seen from the warping, splitting mode was choosen.

Mosaicing examples and comparisons

Toscolano's church (14 horizontal images – reflex)



This slide compares, respectively, two stitching strategies:

- ***first2last*** (bottom-up) ***without split***, cropping from the upper part;
- ***first2last*** (bottom-up) ***with split***, cropping from center.

The two final images are similar, but the cropping algorithm performs significantly better in the first scenario.

Mosaicing examples and comparisons

«Low contrast» mountains (39 vertical images – reflex)

Mosaic first part

Mosaic second part

In this case, the split strategy was used to merge 39 low contrast images.
Despite the wide water and bather presence, the mosaicing is pretty good.

Mosaicing examples and comparisons

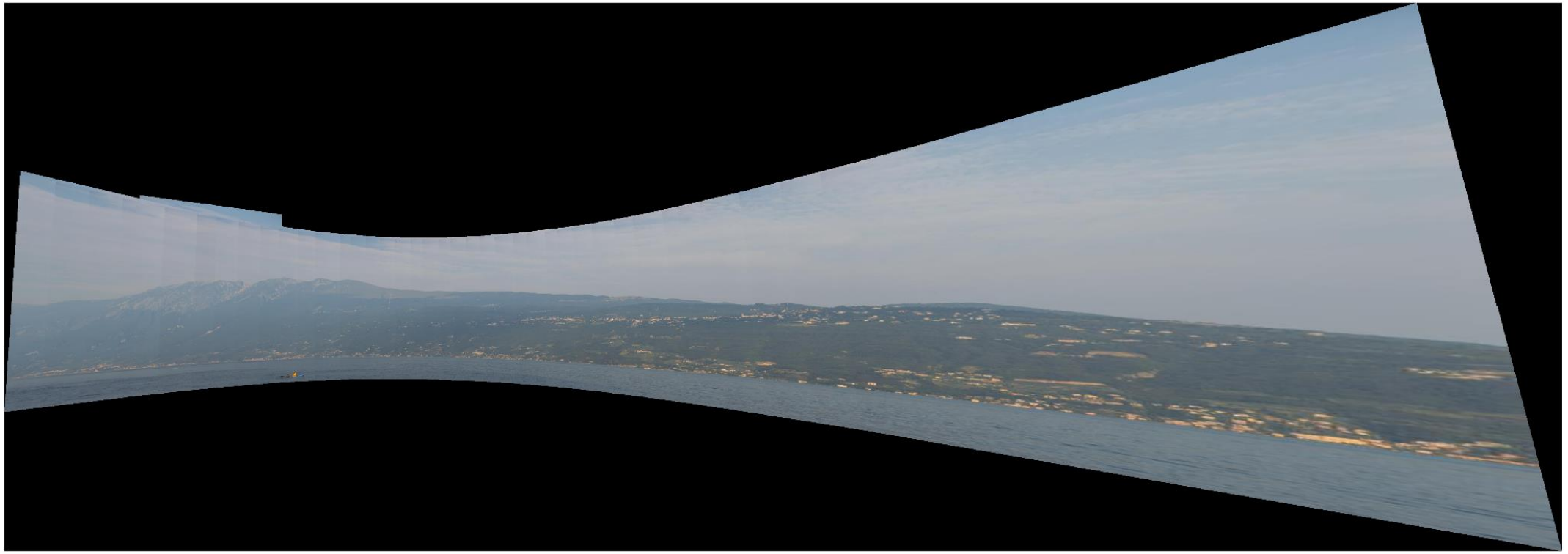
«Low contrast» mountains (39 vertical images – reflex)

Cropped mosaic

The above picture shows the cropped version of the mosaic.

Mosaicing examples and comparisons

«Low contrast» mountains (C.V.Toolbox – SIFT & myHomography)



The shown mosaic is the result of using *SIFT method* in conjunction with *my homography estimation function* and the rest of the tutorial's pipeline. In this case, all 39 images were used.

Mosaicing examples and comparisons

«Low contrast» mountains (C.V.Toolbox – SIFT & C.V.Homography)

The shown mosaic is the result of using *SIFT method* in conjunction with the *estimateGeometricTransform2D* MATLAB function and the rest of the tutorial's pipeline. In this case, all 39 images were used.

Mosaicing examples and comparisons

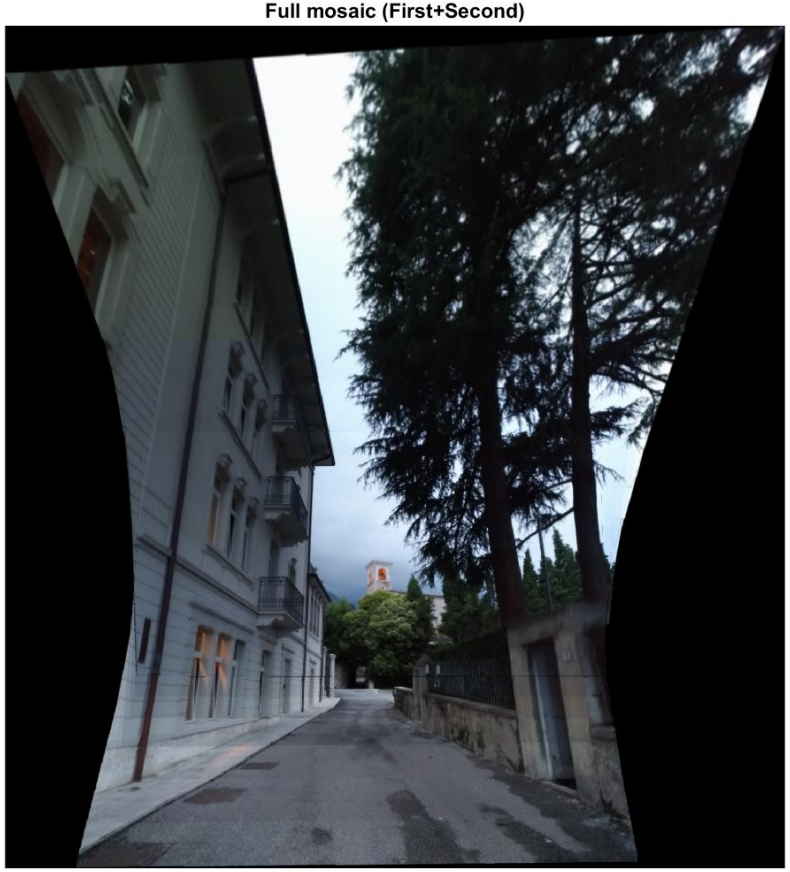
«Low contrast» mountains (C.V.Toolbox – SURF & C.V.Homography)

```
Error using vision.internal.geotrans.algEstimateGeometricTransform>checkRuntimeStatus (line 99)
matchedPoints1 and matchedPoints2 do not have enough points. The number of points in each set
must be at least 4.

Error in vision.internal.geotrans.algEstimateGeometricTransform (line 70)
    checkRuntimeStatus(statusCode, status, sampleSize);

Error in estimateGeometricTransform2D (line 152)
    vision.internal.geotrans.algEstimateGeometricTransform(...

Error in ImageStitchingCVToolbox_SCRIPT (line 162)
        tforms(n) = estimateGeometricTransform2D(matchedPoints, matchedPointsPrev,...
```

In this case, using *SURF method* in conjunction with the *estimateGeometricTransform2D* MATLAB function, the script throws this error. The **number of matching points is not sufficient** for going forward with the pipeline. Increasing the resolution doesn't seem to help much.

Mosaicing examples and comparisons

Fasano's church (14 horizontal images – smartphone)



In this case, the best result is obtained by using the ***splitting option***. Also, the cropped view is pleasent, considering that all the pictures were taken by a smartphone in a relatively low light condition.

Mosaicing examples and comparisons

Toscolano's church and paper mill (8 horizontal images – reflex)

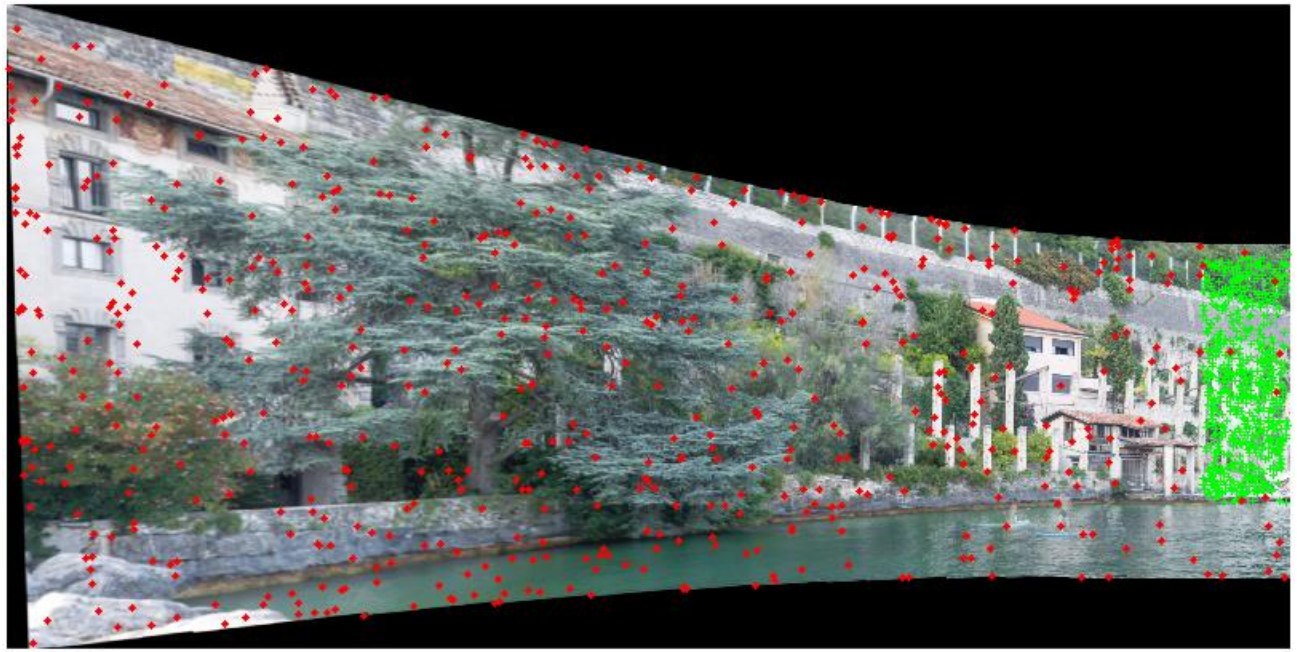
Full mosaic

Cropped mosaic

At the left, the full and cropped versions of the mosaic using ***last2first*** strategy.

Mosaicing examples and comparisons

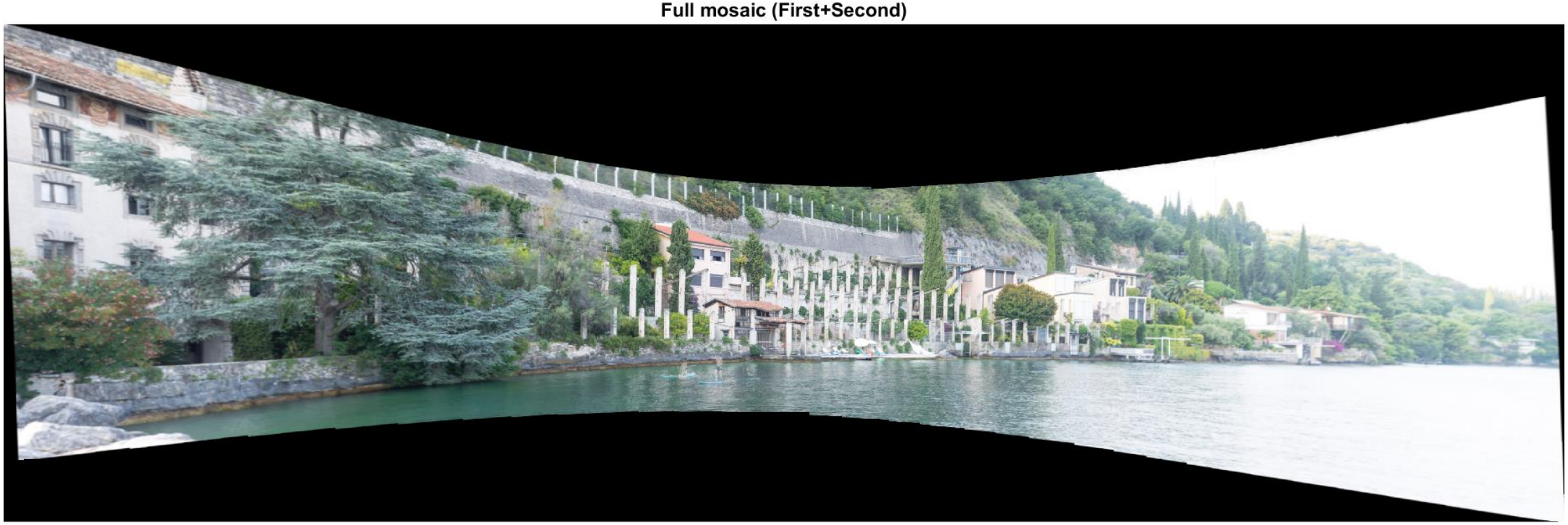
View from Toscolano's dock (25 vertical images – reflex)



In this case, the ***split strategy*** was used to merge the images.
Despite the water and bather presence, the mosaicing is pretty good.

Mosaicing examples and comparisons

View from Toscolano's dock (25 vertical images – reflex)



The above image shows the full mosaic. In this case there are *different lighting conditions and exposures across the image.*
Using a reflex, all the setting are kept the same, so a tradeoff between underexposing and overexposing has to be found.

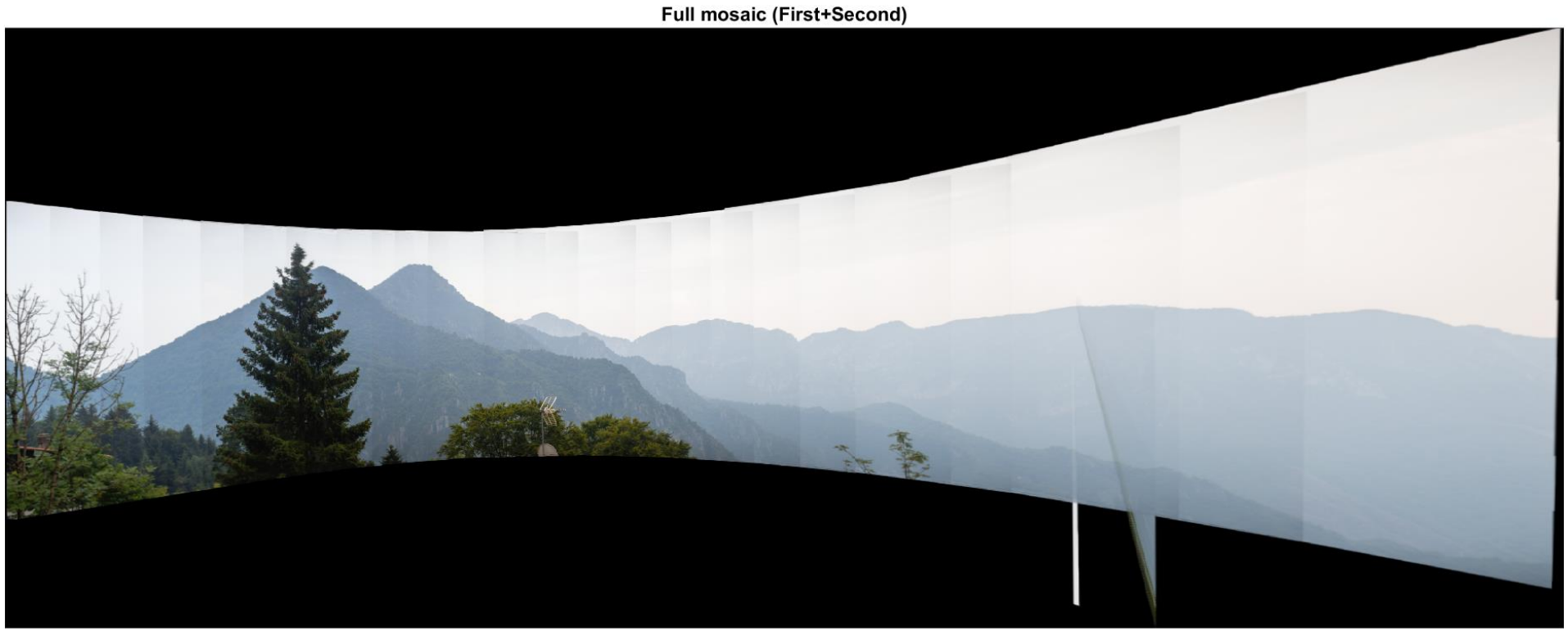
Mosaicing examples and comparisons

View from Toscolano's dock (25 vertical images – reflex)

The above picture shows the cropped version of the mosaic.

Mosaicing examples and comparisons

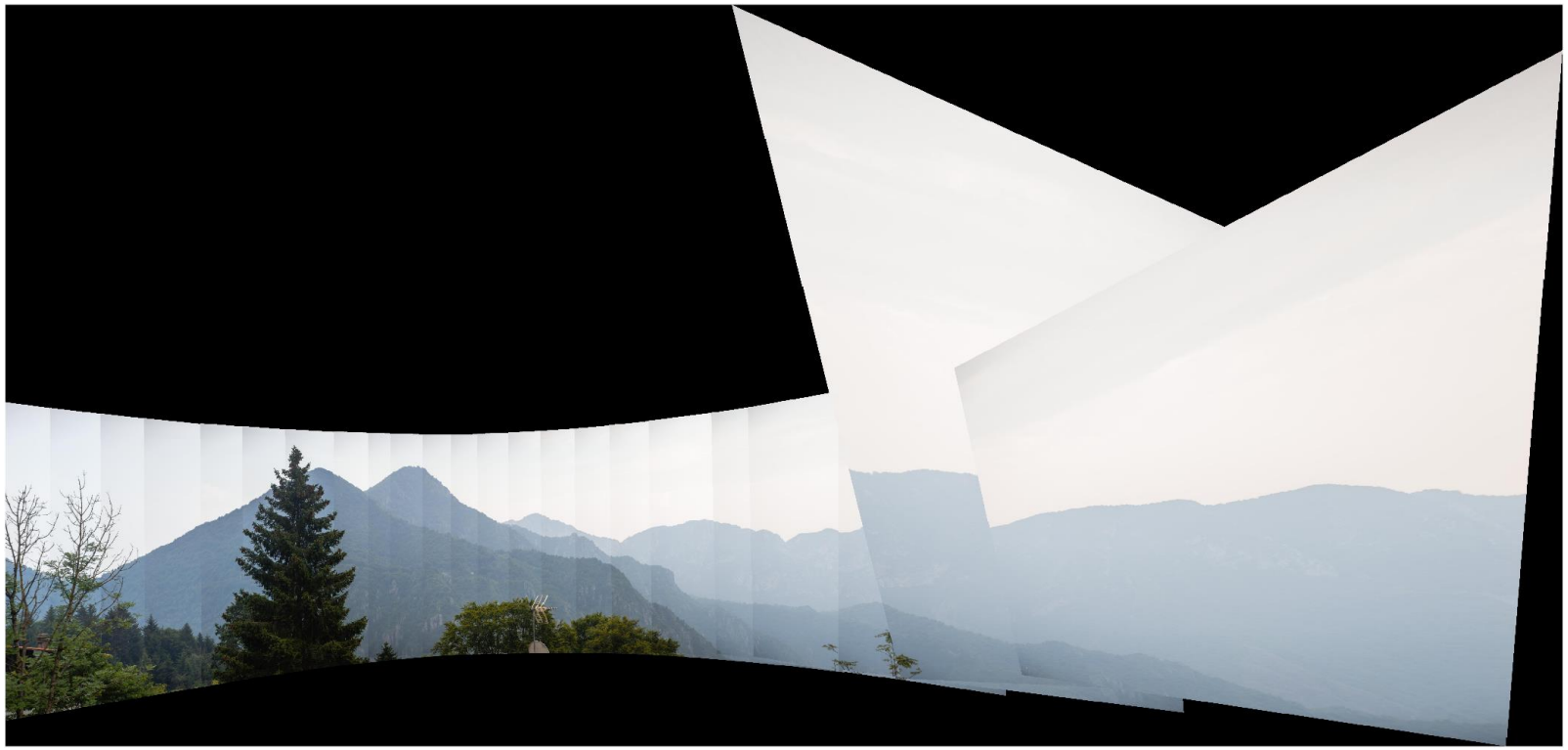
View from Fobbia (1) (23 vertical images – reflex)



Also in this case, the ***split strategy*** was used to merge the images.
The final mosaic shows some artifacts on the right side.

Mosaicing examples and comparisons

View from Fobbia (1) (C.V.Toolbox – SIFT & myHomography)



The shown mosaic is the result of using *SIFT method* in conjunction with *my homography estimation function* and the rest of the tutorial's pipeline. In this case, all 23 images were used.

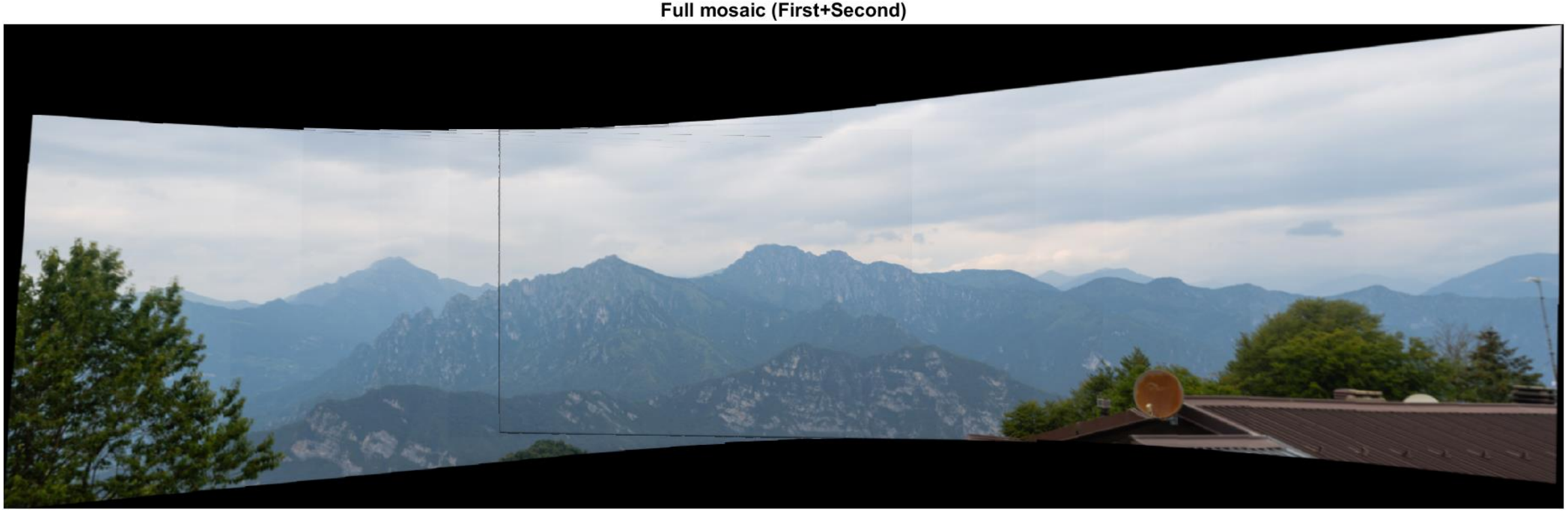
Mosaicing examples and comparisons

View from Fobbia (1) (C.V.Toolbox – SIFT & C.V.Homography)

The shown mosaic is the result of using *SIFT method* in conjunction with the *estimateGeometricTransform2D* MATLAB function and the rest of the tutorial's pipeline. In this case, all 23 images were used.

Mosaicing examples and comparisons

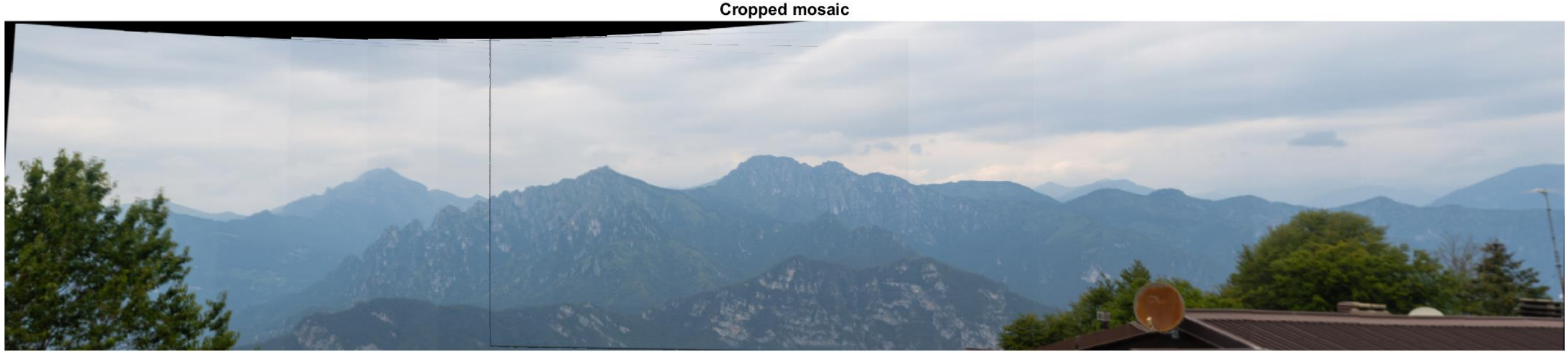
View from Fobbia (2) (15 horizontal images – reflex)



Also in this case, the ***split strategy*** was used to merge the images.
The final mosaic shows some artifacts on the left side.

Mosaicing examples and comparisons

View from Fobbia (2) (15 horizontal images – reflex)



The above picture shows the cropped version of the mosaic.

Mosaicing examples and comparisons

View from Fobbia (2) (C.V.Toolbox – SIFT & myHomography)

The shown mosaic is the result of using *SIFT method* in conjunction with *my homography estimation function* and the rest of the tutorial's pipeline. In this case, all 15 images were used.

Mosaicing examples and comparisons

View from Fobbia (2) (C.V.Toolbox – SIFT & C.V.Homography)

The shown mosaic is the result of using *SIFT method* in conjunction with the *estimateGeometricTransform2D* MATLAB function and the rest of the tutorial's pipeline. In this case, all 15 images were used.

Mosaicing examples and comparisons

View from Fobbia (2) (C.V.Toolbox – SURF & C.V.Homography)

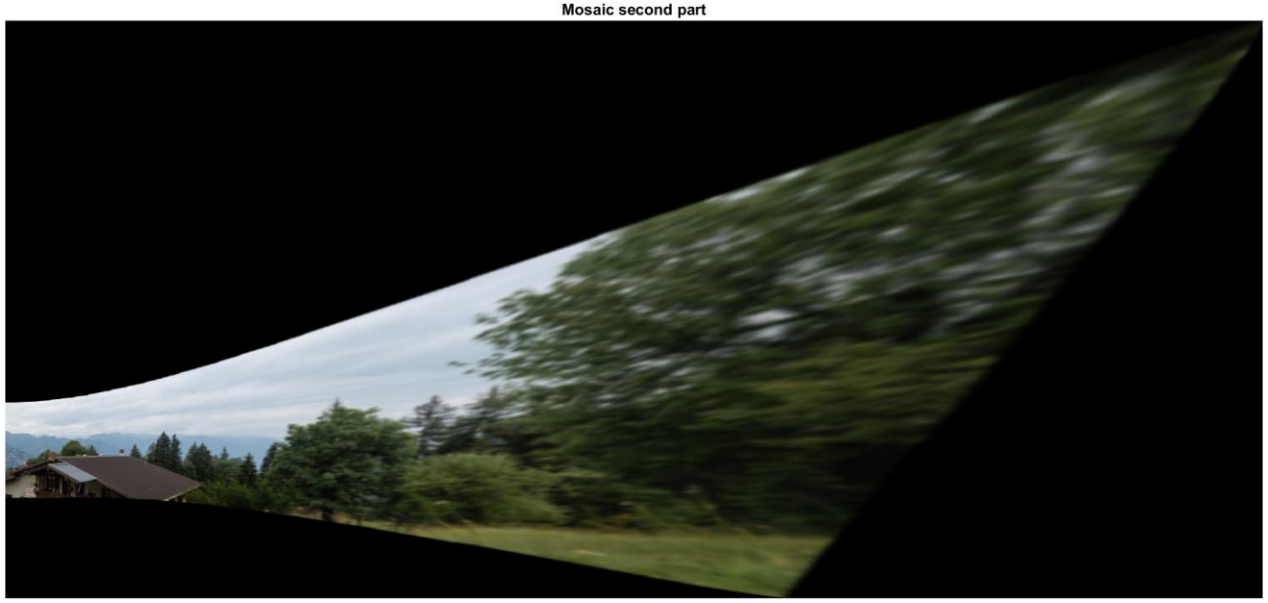
The shown mosaic is the result of using *SURF method* in conjunction with the *estimateGeometricTransform2D* MATLAB function and the rest of the tutorial's pipeline. In this case, all 15 images were used.

Mosaicing examples and comparisons

View from Fobbia (3) (40 vertical images – reflex)

Mosaic first part

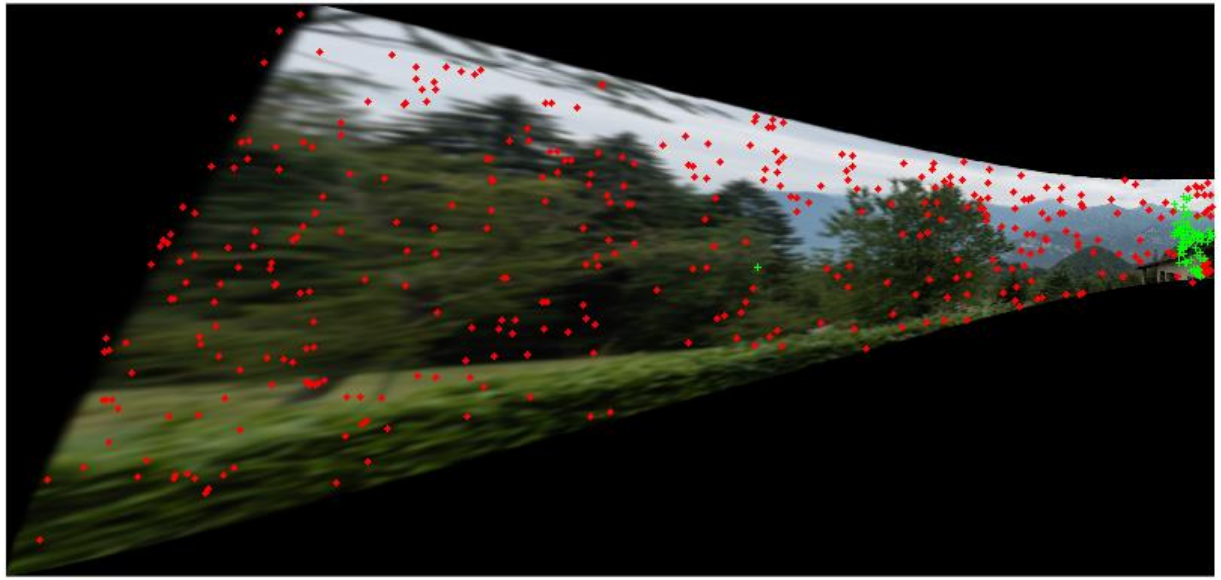
Mosaic second part

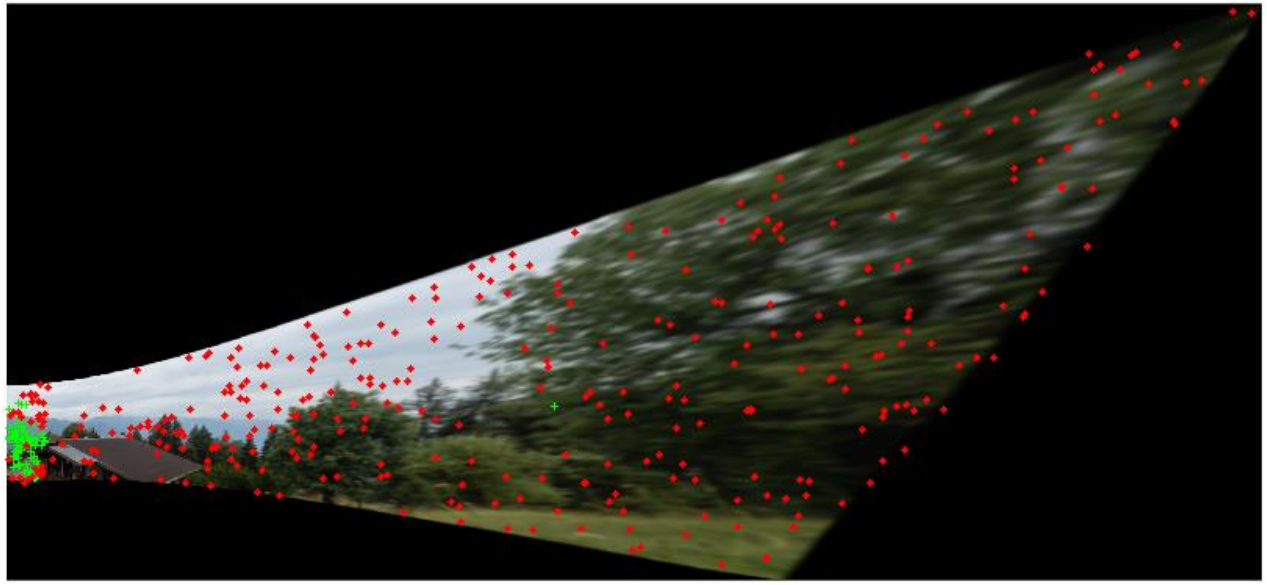


Also in this case, the ***split strategy*** was used to merge the images.
The «blurring effect» on the sides is due to the low resolution of the images (400px on the long side).

Mosaicing examples and comparisons

View from Fobbia (3) (40 vertical images – reflex)





Unfortunately, only one inliers can impact a lot on the final result. This example is brought to attention to show that a mask would help a lot when the number of images increases and the resolution is kept relatively low to be able to handle such a large image.

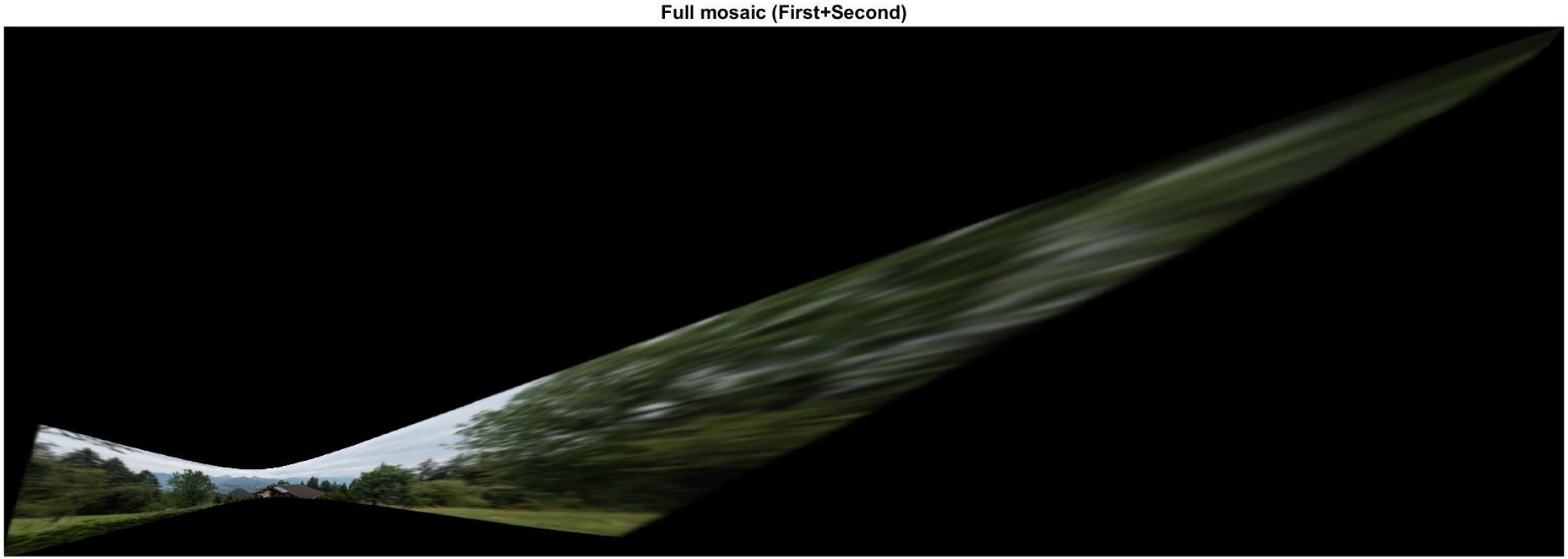
Mosaicing examples and comparisons

View from Fobbia (3) (40 vertical images – reflex)

In fact, the final mosaic isn't representing the whole panorama, even though the two separate views are correct.

Mosaicing examples and comparisons

View from Fobbia (3) (40 vertical images – reflex)



By repeating the final stitching step and, if needed, twiking the homography estimation parameters we can get the correct panorama (even if not so pleasent because of the high distortion).

Mosaicing examples and comparisons

View from Fobbia (3) (Photoshop's *photomerge*)

The above picture shows the resulting mosaicing from the «photomerge» function inside Photoshop CS6 2018.
The ***Automatic*** choise of mosaicing layout was left (looking at the following slides, «*Spherical*» was choosen).

Mosaicing examples and comparisons

View from Fobbia (3) (Photoshop's *photomerge*)

The above picture shows the resulting mosaicing from the «photomerge» function inside Photoshop CS6 2018.
The ***Cylindrical*** mosaicing layout was choosen.

Mosaicing examples and comparisons

View from Fobbia (3) (Photoshop's *photomerge*)

The above picture shows the resulting mosaicing from the «photomerge» function inside Photoshop CS6 2018.
The ***Spherical*** mosaicing layout was choosen.

Mosaicing examples and comparisons

View from Fobbia (3) (Photoshop's *photomerge*)

The above picture shows the resulting mosaicing from the «photomerge» function inside Photoshop CS6 2018.
The ***Collage*** mosaicing layout was choosen (even if not suited for the type of image).

Mosaicing examples and comparisons

View from Fobbia (3) (comments)

- From this comparison we can say that **Photoshop** is able to handle ***bigger format images***, even though they still had to be resized to 2000px on the long side to complete the process in a reasonable time.
- Also, the *different layout options* were explored, and they led to similar results. I'd say that Photoshop 2018 performs the best choosing by itself (leaving "Auto" layout).
- The **developed MATLAB script** couldn't produce an acceptable result and the maximum resolution (which impacts on the mosaicking performances) was around 400px on the long side.
- On its defense, Photoshop couldn't produce the panorama image using the "Perspective" layout (it takes too long, no sign of progress for tens of minutes). Judging from the distortion effect (*butterfly*), the *Perspective layout* should be the most similar mosaicing method to the one developed inside MATLAB.
- As a last note, the different layout modes are thought for different scenarios. For example, the "Collage" option isn't suited to the used photos (because a tripod was used for taking the pictures and they only differ by a rotation).

Mosaicing examples and comparisons

View from Fobbia (4) (37 vertical images – reflex)

Full mosaic (First+Second)

The above picture shows a similar panorama to the previous one. In this case the end result is slightly better.

Mosaicing examples and comparisons

View from Fobbia (4) (37 vertical images – reflex)

Cropped mosaic

The above picture shows the cropped version of the mosaic.

Mosaicing examples and comparisons

View from Fobbia (4) (Photoshop's *photomerge*)

The above picture shows the resulting mosaic from the «photomerge» function inside Photoshop CS6 2018.
The ***Automatic*** choice of mosaicing layout was left.

Mosaicing examples and comparisons

View from Fobbia (5) (17 vertical images – reflex)

Full mosaic (First+Second)

The above picture shows a closer view to the Fobbia location in the middle of the mountains. This is one of the best achieved results.

Mosaicing examples and comparisons

View from Fobbia (5) (17 vertical images – reflex)

Cropped mosaic

The above picture shows the cropped version of the mosaic. To notice that these closer views are shot with the maximum *focal length* (105mm), so lens distortion and vignetting weren't corrected but didn't ruin the end result.

Mosaicing examples and comparisons

View from Fobbia (6) (20 vertical images – reflex)

The above picture shows the mountains behind the previous view.

Mosaicing examples and comparisons

View from Fobbia (6) (20 vertical images – reflex)

The above picture shows the cropped version of the mosaic. Being the distortion not symmetric, the cropping algorithm struggles.

Mosaicing examples and comparisons

Water presence in pictures

- The following slides show a few examples where a panorama is created but the **results are not so good**, mainly because of the *water* inside the image. In fact, inliers that are used to estimate the homography matrices, are found only on the background mountains.
- Also, the *color blending isn't so successful*: the stitched images can be clearly distinguished looking at the final mosaic.
- **Comparisons** with the Computer Vision Toolbox script are made, to see if there are significant differences.
- Moreover, the first two shown mosaics are composed by images taken with a ***handheld*** reflex, and not fixed on a tripod. This makes the stitching process harder and shouldn't be done.
- It is interesting to see that, sometimes, when SURF method is used the mosaicing process fails because of the not enough number of inliers. Despite that, SURF seems to extract "better" salient points than SIFT and the overall result looks more pleasant.

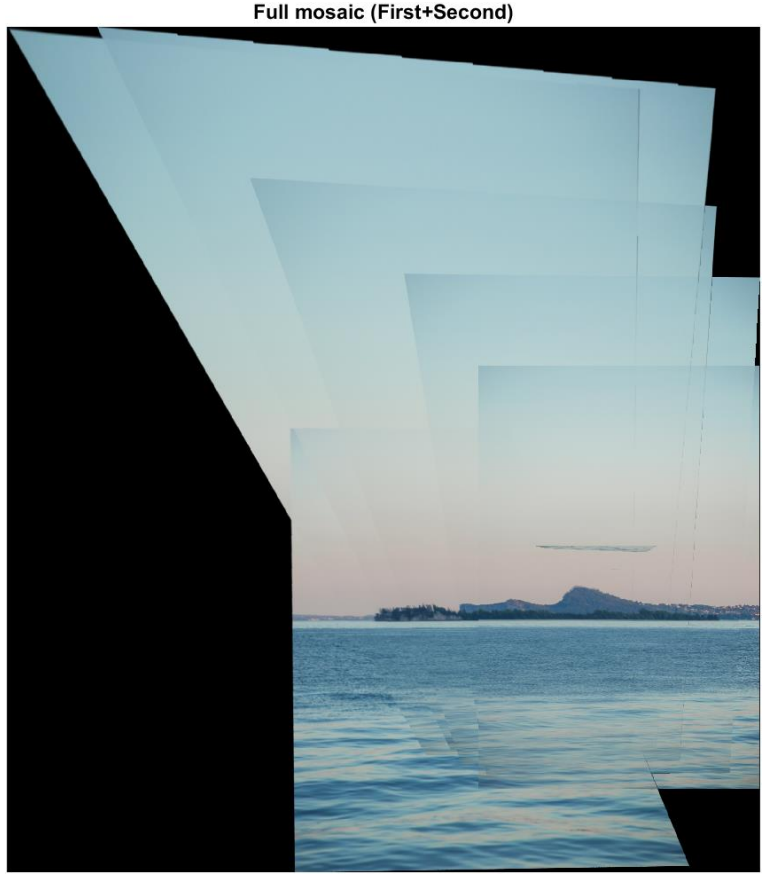
Mosaicing examples and comparisons

«Rocca di Manerba» (14 vertical images – reflex)

These two images are the first and second part of the mosaic showing ***inliers*** and ***outliers***. The only good corrispondences are on the mountains being the fixed objects. All the 14 pictures were taken without a tripod.

Mosaicing examples and comparisons

«Rocca di Manerba» (14 vertical images – reflex)



The final mosaic isn't very representative of the scene. In fact, the first part is mainly shown.

Mosaicing examples and comparisons

«Rocca di Manerba» (C.V.Toolbox – SIFT & myHomography)

The shown mosaic is the result of using *SIFT method* in conjunction with *my homography estimation function* and the rest of the tutorial's pipeline. In this case, all 14 images were used.

Mosaicing examples and comparisons

«Rocca di Manerba» (C.V.Toolbox – SIFT & C.V.Homography)

The shown mosaic is the result of using *SIFT method* in conjunction with the *estimateGeometricTransform2D* MATLAB function and the rest of the tutorial's pipeline. In this case, all 14 images were used.

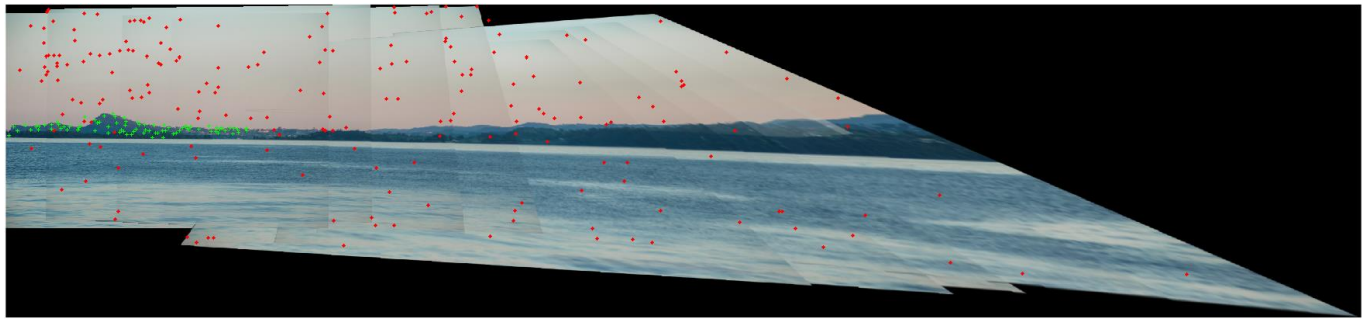
Mosaicing examples and comparisons

«Rocca di Manerba» (C.V.Toolbox – SURF & C.V.Homography)

The shown mosaic is the result of using *SURF method* in conjunction with the *estimateGeometricTransform2D* MATLAB function and the rest of the tutorial's pipeline. In this case, all 14 images were used.

Mosaicing examples and comparisons

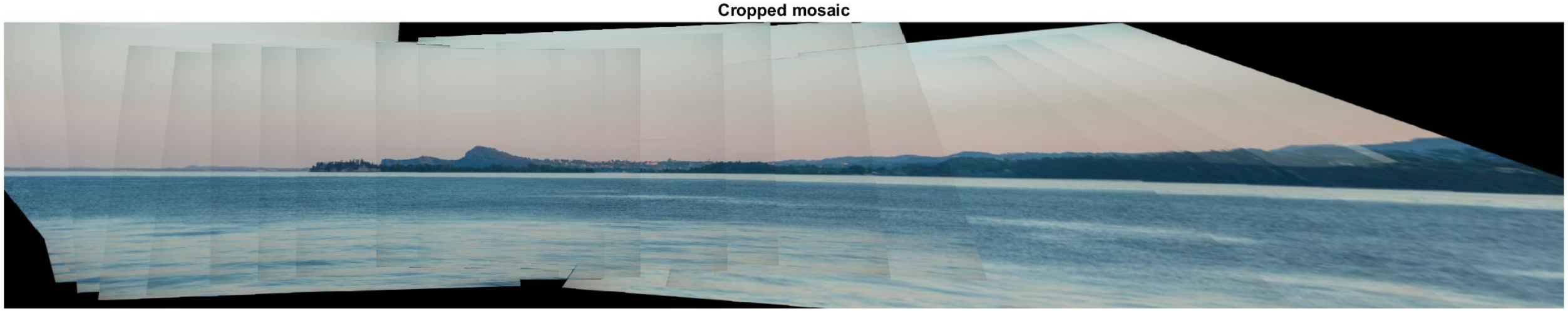
«Rocca di Manerba» (2) (18 vertical images – reflex)



The above images represent the first and second part of the mosaic showing ***inliers*** and ***outliers***.
The only good corrispondences are on the mountains being the fixed objects. All the 18 pictures were taken **without a tripod**.

Mosaicing examples and comparisons

«Rocca di Manerba» (2) (18 vertical images – reflex)



The above picture shows the cropped version of the mosaic. Being the distortion not symmetric, the cropping algorithm struggles.

Mosaicing examples and comparisons

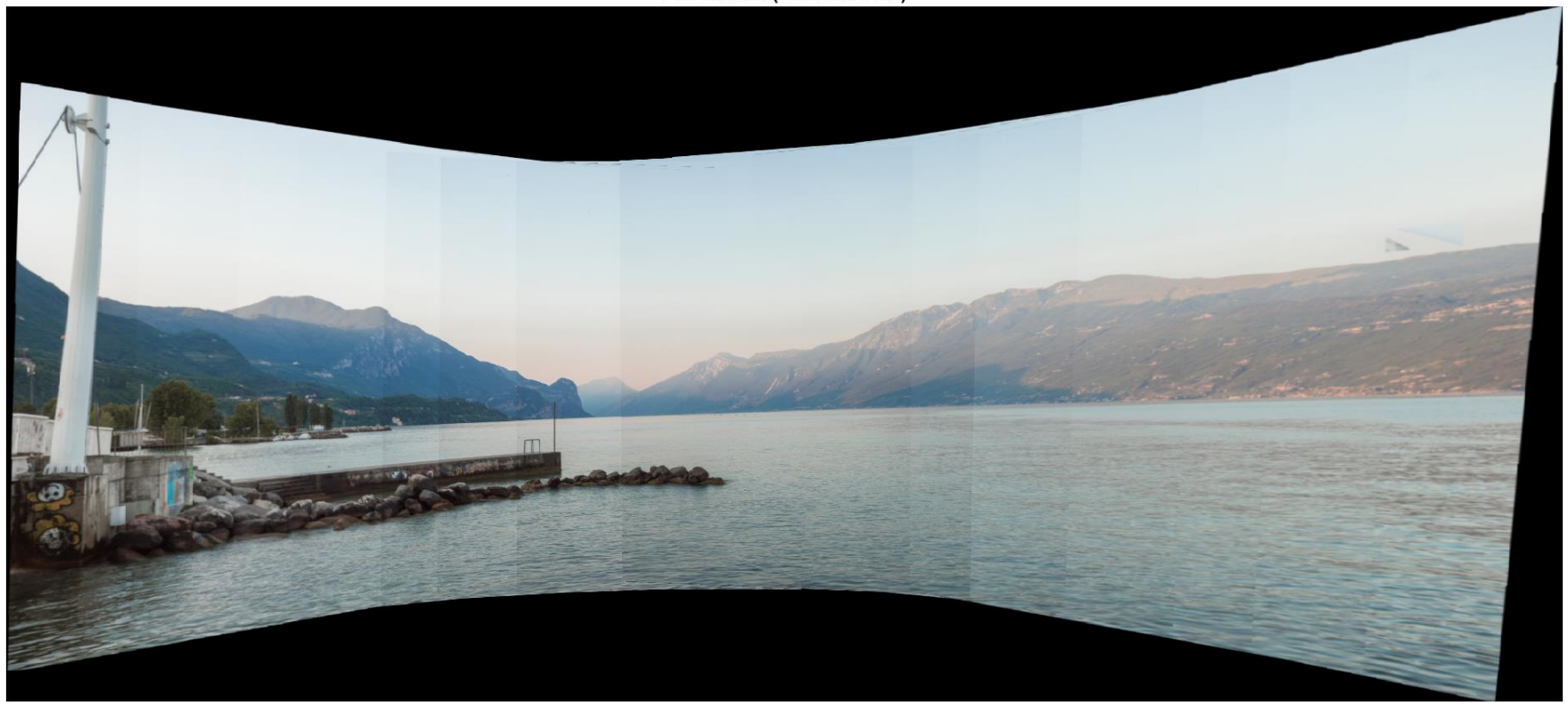
Near the bridge (15 vertical images – reflex)

In this case, the ***split strategy*** was used to merge the images.
Despite the water presence, the mosaicing is pretty good. All the inliers are found on the rocks and on the mountains.

Mosaicing examples and comparisons

Near the bridge (15 vertical images – reflex)

Full mosaic (First+Second)

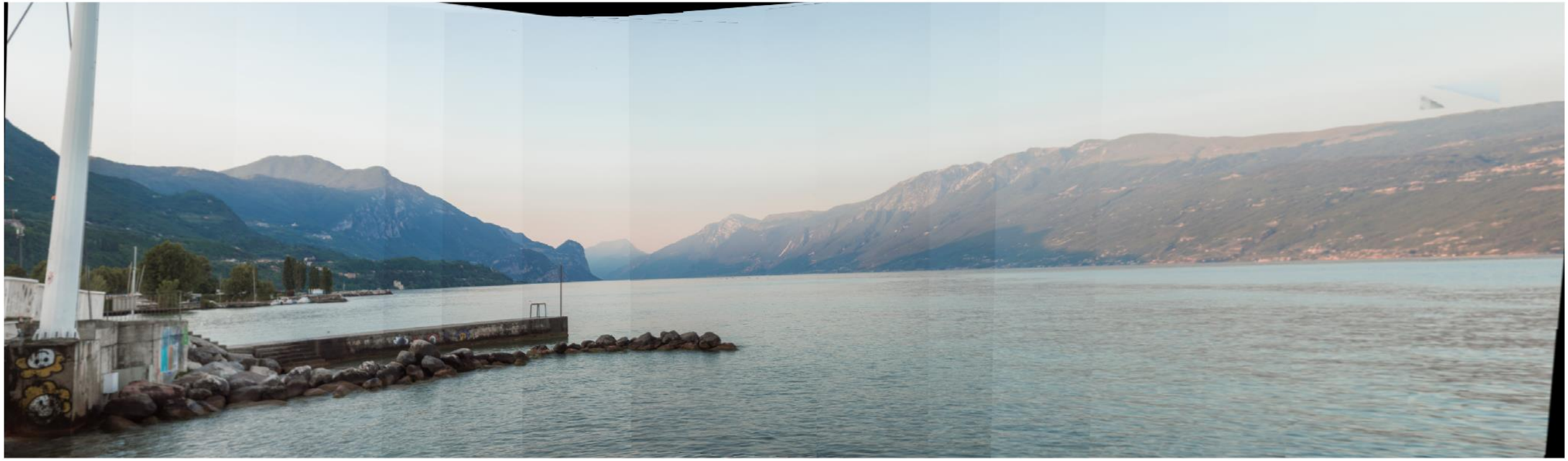


The above picture shows the full mosaic.

Mosaicing examples and comparisons

Near the bridge (15 vertical images – reflex)

Cropped mosaic



The above picture shows the cropped version of the mosaic.

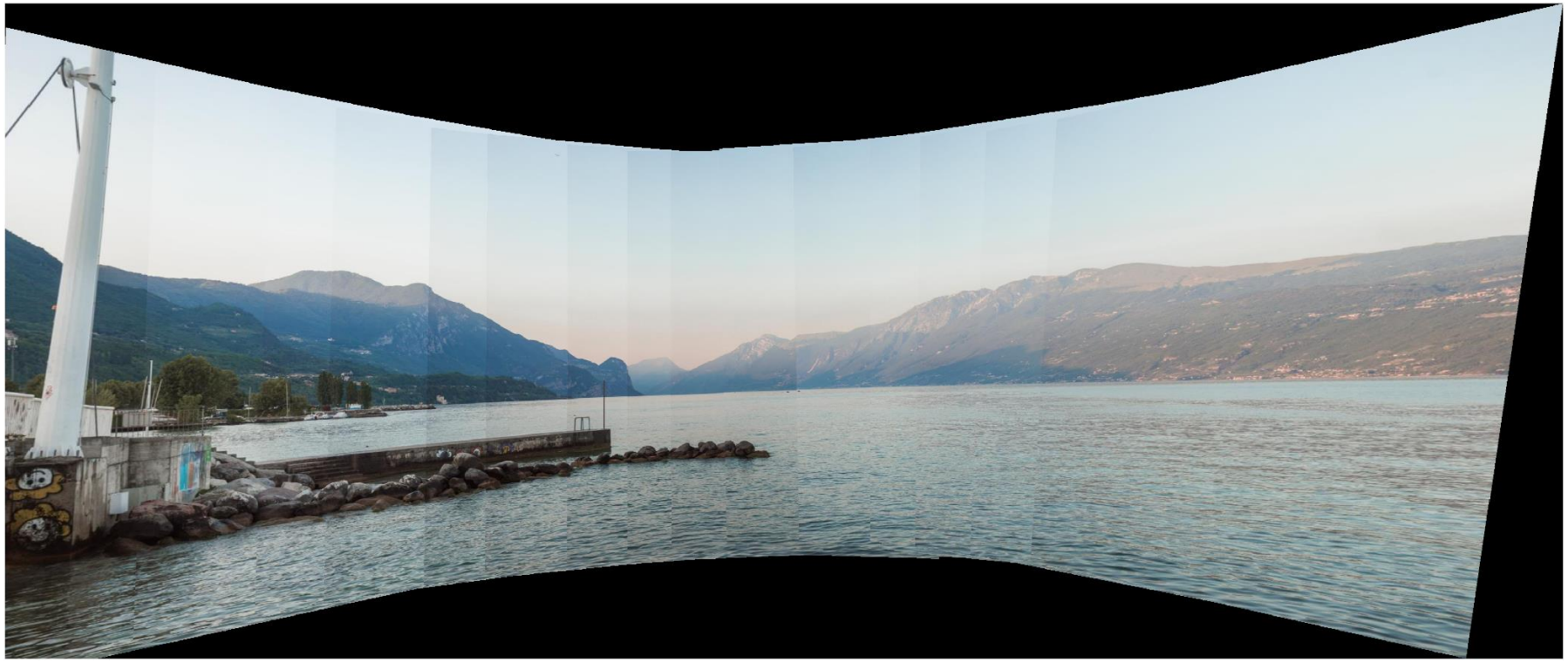
Mosaicing examples and comparisons

Near the bridge (C.V.Toolbox – SIFT & myHomography)

The shown mosaic is the result of using *SIFT method* in conjunction with *my homography estimation function* and the rest of the tutorial's pipeline. In this case, all 15 images were used.

Mosaicing examples and comparisons

Near the bridge (C.V.Toolbox – SIFT & C.V.Homography)



The shown mosaic is the result of using *SIFT method* in conjunction with the *estimateGeometricTransform2D* MATLAB function and the rest of the tutorial's pipeline. In this case, all 15 images were used.

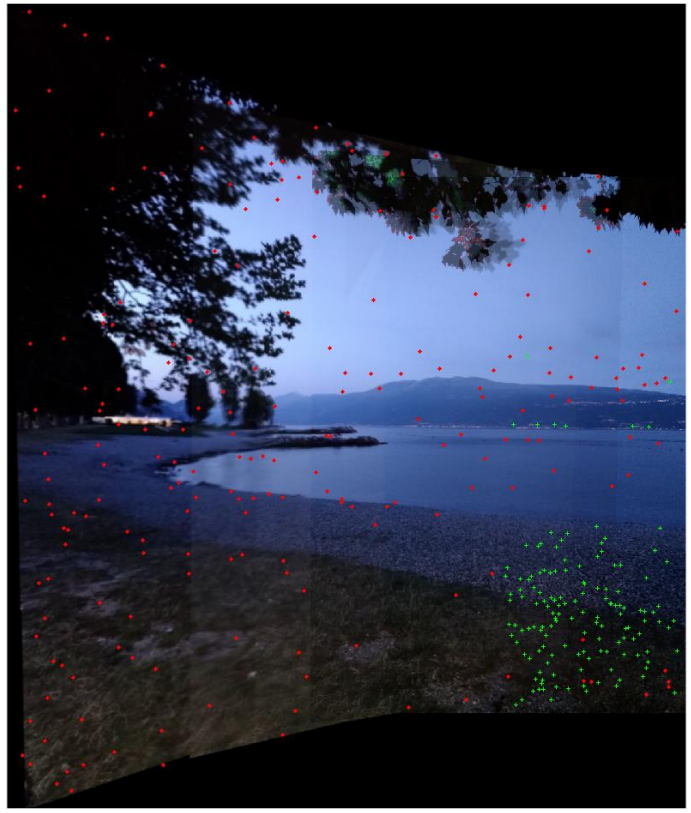
Mosaicing examples and comparisons

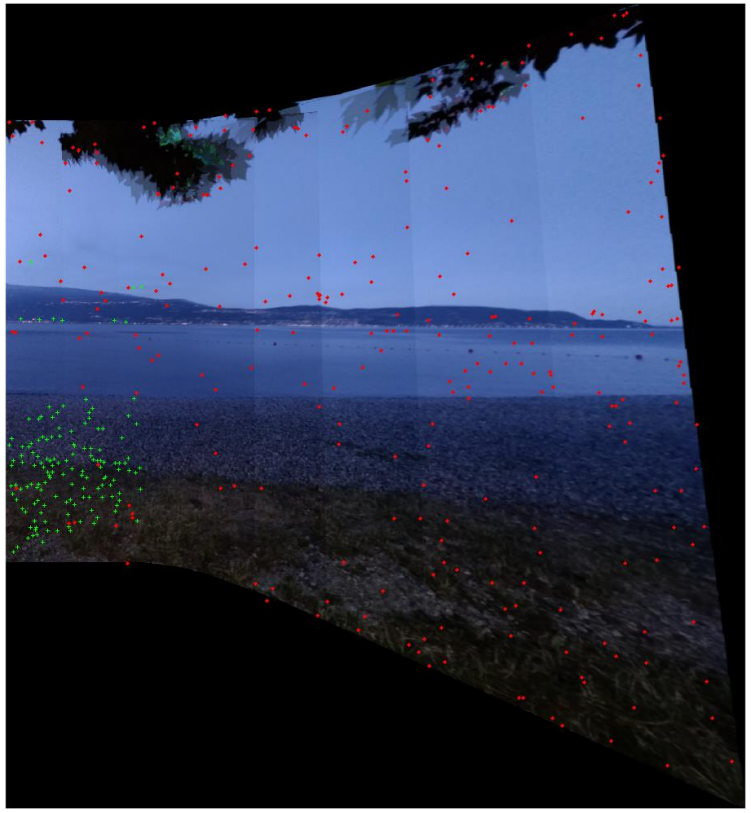
Smartphone pictures

- The following slides show several attempts to create panorama images using ***low quality photos***. Also, in some cases, they don't respect the theoretical guidelines, having lots of moving objects or non-planar scenes.
- *The MATLAB script's results are* ***compared*** *to Photoshop's and, in some cases to the mobile phone's Camera App.*
- It's interesting to see that also Photoshop struggles in these "extreme" scenarios and results are comparable if not worst.
- Photoshop, as previously mentioned merges the images keeping the separate layers, so an expert could change the mask layouts in order to obtain a better results. Of course, lots of other image processing tools are present inside the software.

Mosaicing examples and comparisons

Lake at night (12 vertical images – smartphone)





As shown, inliers are mainly found on the ground. On the right the full mosaic is shown.

Mosaicing examples and comparisons

Lake at night (12 vertical images – smartphone)

Cropped mosaic

The above picture shows the cropped version of the mosaic.

Mosaicing examples and comparisons

Lake at night (12 vertical images – smartphone)

The above picture shows the resulting mosaic from the «photomerge» function inside Photoshop CS6 2018.
The ***Automatic*** choice of mosaicing layout was left.

Mosaicing examples and comparisons

Lake at night (C.V.Toolbox – SIFT & myHomography)

The shown mosaic is the result of using *SIFT method* in conjunction with *my homography estimation function* and the rest of the tutorial's pipeline. In this case, all 12 images were used.

Mosaicing examples and comparisons

Lake at night (C.V.Toolbox – SIFT & C.V.Homography)

The shown mosaic is the result of using *SIFT method* in conjunction with the *estimateGeometricTransform2D* MATLAB function and the rest of the tutorial's pipeline. In this case, all 12 images were used.

Mosaicing examples and comparisons

Lake at night (C.V.Toolbox – SURF & C.V.Homography)

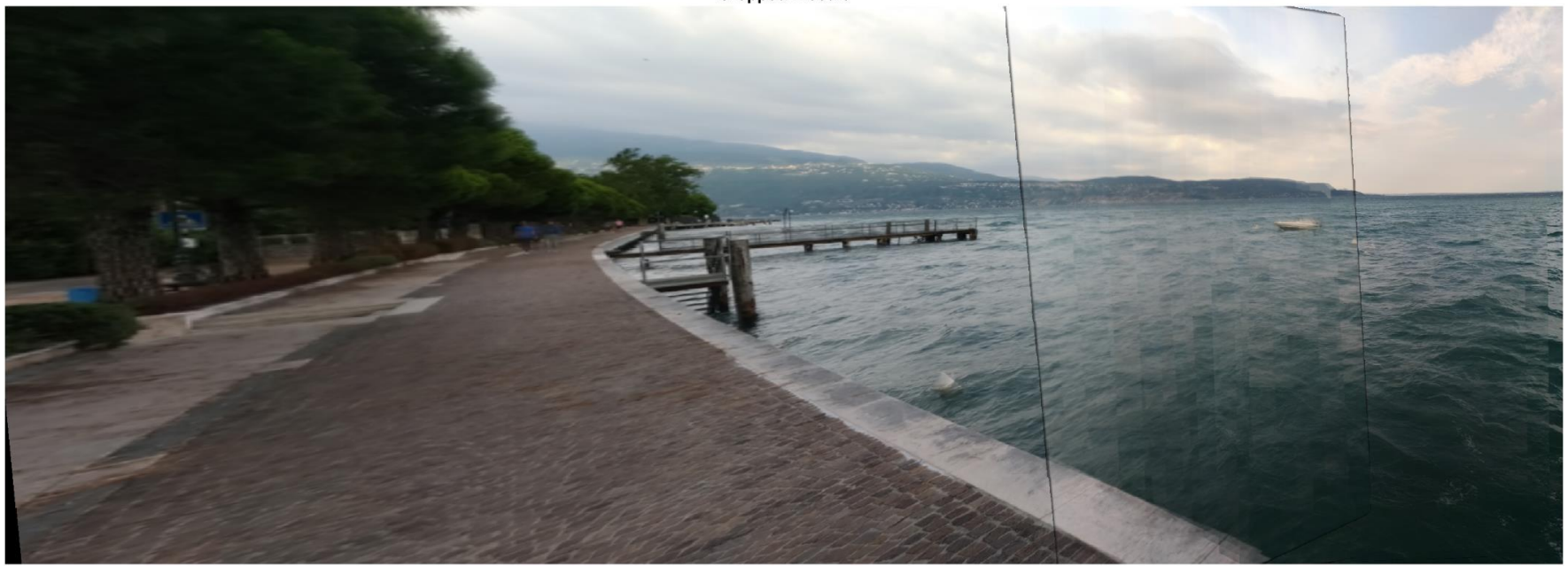
The shown mosaic is the result of using *SURF method* in conjunction with the *estimateGeometricTransform2D* MATLAB function and the rest of the tutorial's pipeline. In this case, all 12 images were used.

Mosaicing examples and comparisons

Boat and waves (24 vertical images – smartphone)

Full mosaic (First+Second)

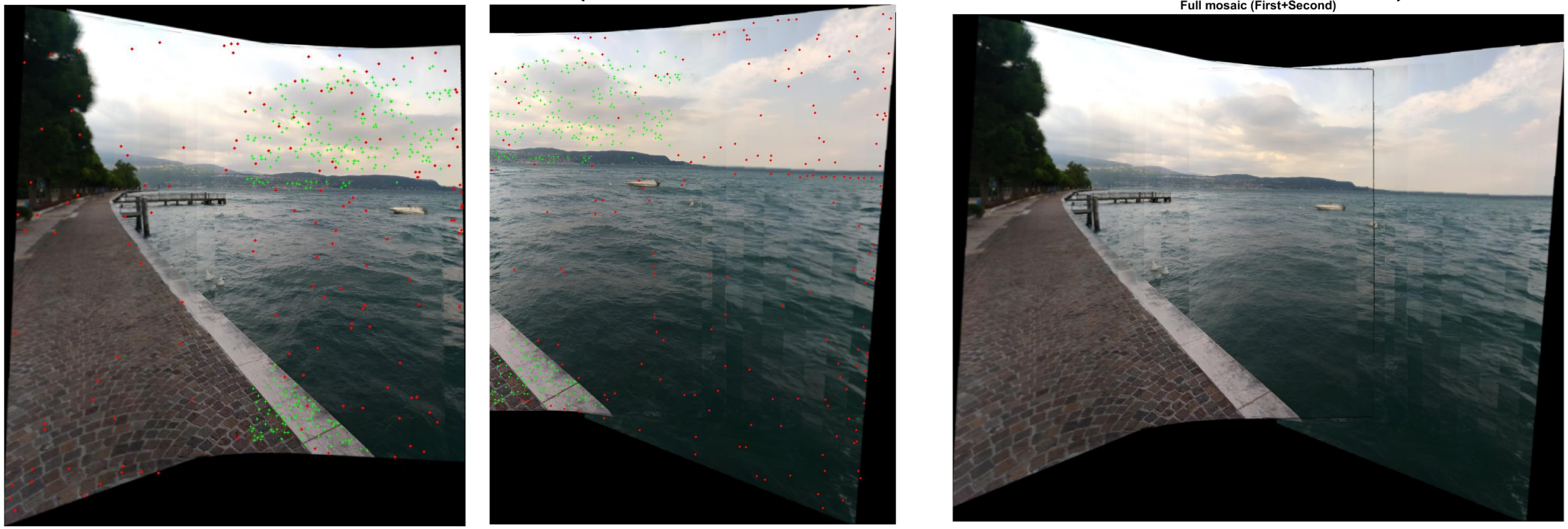
Cropped mosaic



As the above pictures show, the final mosaic (obtained after several tries), isn't very pleasing, beeing the scenerio not so trivial.

Mosaicing examples and comparisons

Boat and waves (24 vertical images – smartphone)



The above pictures show a better result, after a resolution increase (1000px max). My MATLAB script struggled to get a good result because of the relatively low quality HDR photos and, more importantly, the high amount of «moving objects». It's interesting to see that inliers have only been found on the sidewalks and on the background mountains

Mosaicing examples and comparisons

Boat and waves (24 vertical images – smartphone)

The above picture shows the cropped version of the improved (higher input resolution images) mosaic.

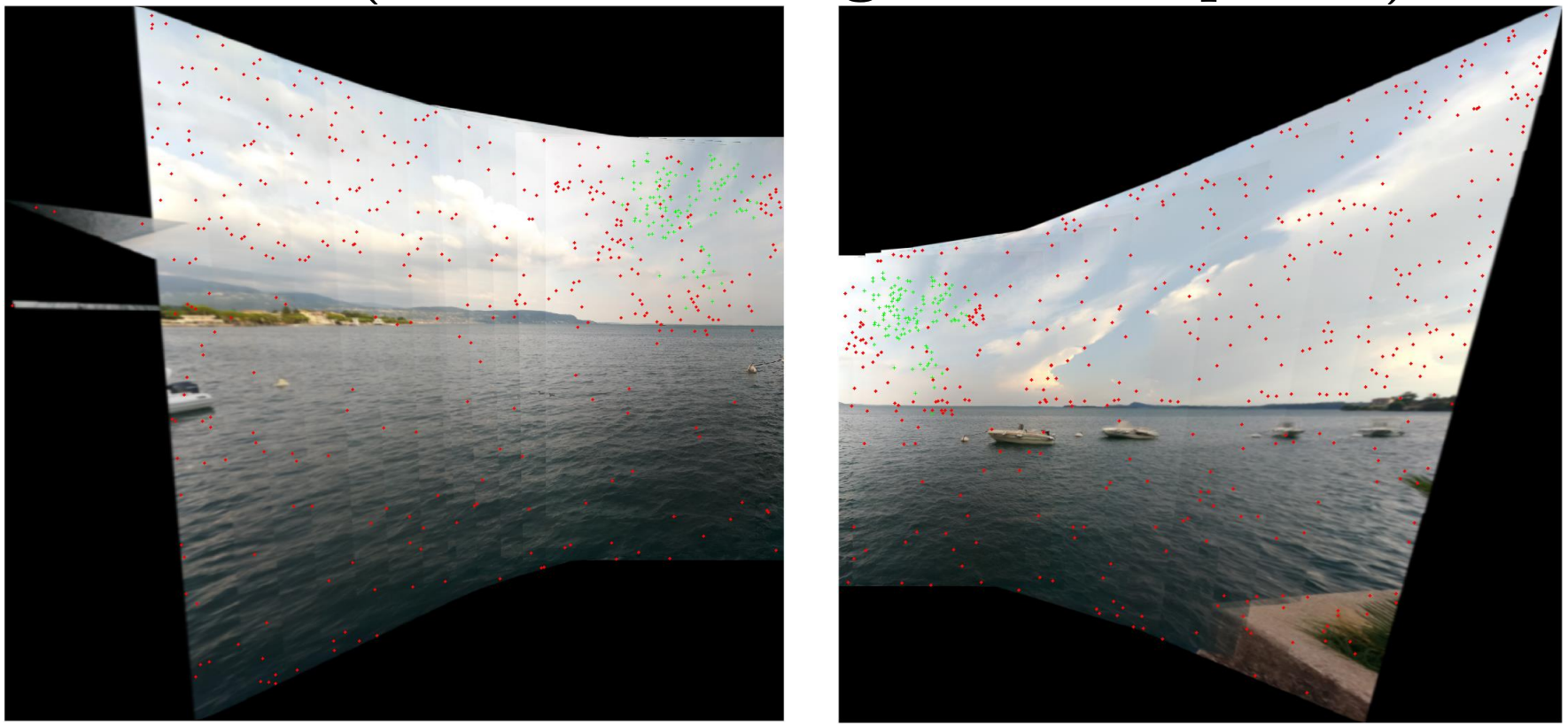
Mosaicing examples and comparisons

Boat and waves (Photoshop's *photomerge*)

The picture on the side shows the resulting mosaic from the «photomerge» function inside Photoshop CS6 2018. The ***Automatic*** choice of mosaicing layout was left.
I think, even though I couldn't find any official information about it, that Photoshop uses a different and far better approach to solve the mosaicing problem. Waves and boats, from distance, look crisp and without discontinuities. Color blending is almost perfect without tweaking the individual layers.

Mosaicing examples and comparisons

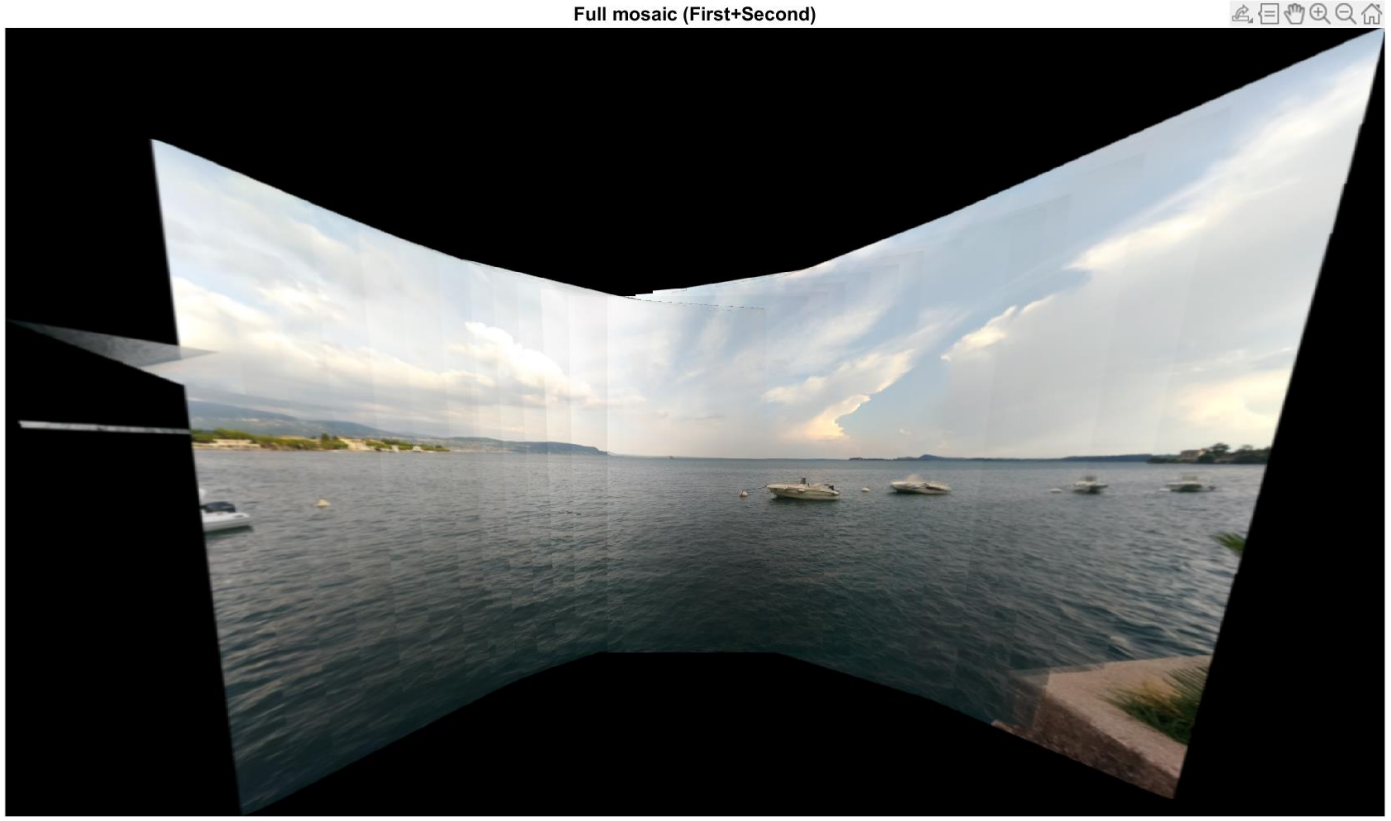
Boats (27 vertical images – smartphone)



In this case, the script struggles a lot to find inlier corrispondences. In fact, most of them are inside the clouds.

Mosaicing examples and comparisons

Boats (27 vertical images – smartphone)



The above image shows the resulting mosaic.

Mosaicing examples and comparisons

Boats (27 vertical images – smartphone)

Cropped mosaic

The above image shows the cropped version of the mosaic.

Mosaicing examples and comparisons

Boats (Photoshop's *photomerge*)

The above picture shows the resulting mosaic from the «photomerge» function inside Photoshop. ***Automatic*** mosaicing layout was left.

Mosaicing examples and comparisons

Boats (*smartphone's panorama* function)

The above pictures show the mosaics obtained using the ***smartphone's camera*** and slowly dragging it from left to right. The result is pleasent, considering the low quality photos and the reduced amount of time required. Two views are inserted beacuse *every time the result slightly changes*.

Mosaicing examples and comparisons

Not planar scene (8 horizontal images – smartphone)

Full mosaic

Cropped mosaic

In this case, the **elements inside the picture are not on the same plane** (it shouldn't be done to have a good result, but was done on purpose.)

Mosaicing examples and comparisons

Not planar scene (Photoshop's *photomerge*)

The picture on the side shows the resulting mosaic from the «photomerge» function inside Photoshop CS6 2018. The ***Automatic*** choice of mosaicing layout was left.
To notice that the church on the background is blurred.

Mosaicing examples and comparisons

Not planar scene (*smartphone's panorama* function)

The picture on the side shows the panorama obtained using the ***smartphone's camera*** and slowly dragging it from bottom to up. The result is very pleasent (despite the low dynamic range), considering the low quality smartphone and the reduced amount of time required.

Mosaicing examples and comparisons

Not planar scene (2) (11 horizontal images – smartphone)

Also in this case, the elements inside the picture are not on the same plane. *On the left, my script's result, on the right Photoshop's result*.

Mosaicing examples and comparisons

Not planar scene (2) (11 horizontal images – smartphone)

The picture on the side shows the panorama obtained using the ***smartphone's camera*** and slowly dragging it from bottom to up. The result is very pleasent (despite the low dynamic range), considering the low quality smartphone and the reduced amount of time required.

Conclusions

- This *Computer Vision* project deepened the ***mosaicing problem***, comparing multiple results obtained both from developed scripts and professional image processing software.
- Image resolution, computational power, speed, optimization are all aspects that must be taken into consideration to reach pleasant results.
- Also, it's important to correctly set the algorithms' parameters and sometimes (for example, in case of not planar scenes, handheld taken pictures or low quality images), they should be tweeked. As shown, not in all cases results were pleasant and possible (sometimes SIFT or SURF fail to find enough correspondences, maybe the robustness could discreased in those cases, to increase the number of inliers).