APP5 - Image processing - TP 3

Tom Mansion and Sophie Nguyen

December 5, 2021

Contents

1 Exercise 1

1.1 Benchmark of the point of interest detection methods

Because there are fragments to locate on all the fresco, we are looking for a method that has a good point detection coverage of all of the image.

Also, because our fragments have a black border due to the fact that they are png, the method need to detect points inside the fragments and not only at the border.

1.1.1 Harris

From the documentation `https://docs.opencv.org/4.x/d4/dc6/tutorial_py_template_matching.html`

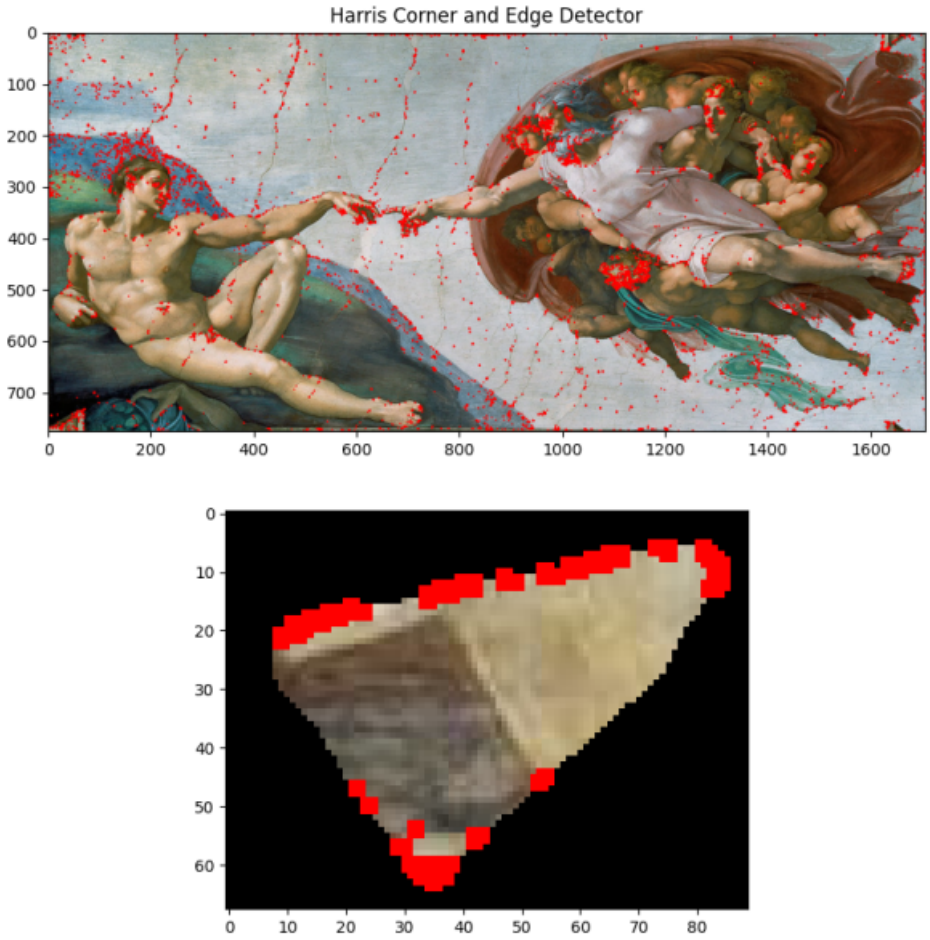

Figure 1: Harris results with the fresco and the first fragment

Notes

- Too few points on the fresco, some parts of it are lacking points
- Only the border of the fragment has points of interest

1.1.2 Shi-Tomasi

From the documentation `https://docs.opencv.org/4.x/d4/d8c/tutorial_py_shi_tomasi.html`

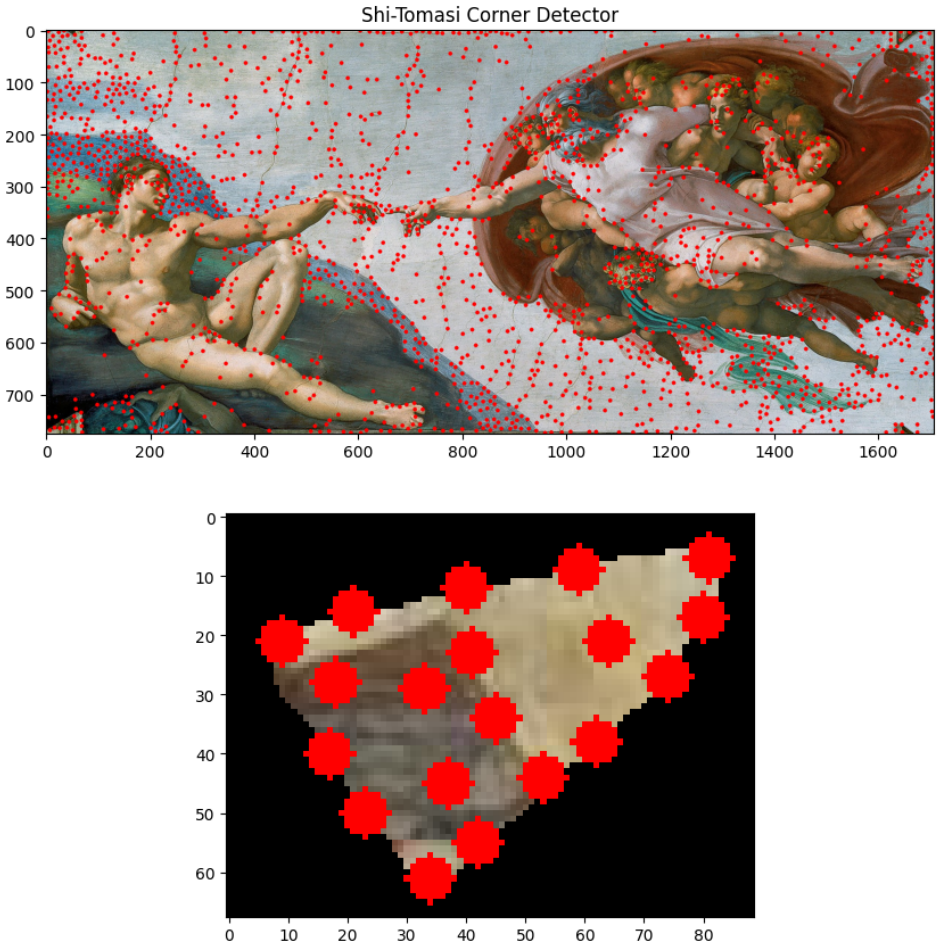

Figure 2: Shi-Tomasi results with the fresco and the first fragment

Notes

- Same amount of points, but more spread out
- Some points are on the fragment border, but they can be removed

– Some points are on the fragment, but they doesn't seem to correlate

1.1.3 SIFT

From the documentation `https://docs.opencv.org/4.x/da/df5/tutorial_py_sift_intro.html`

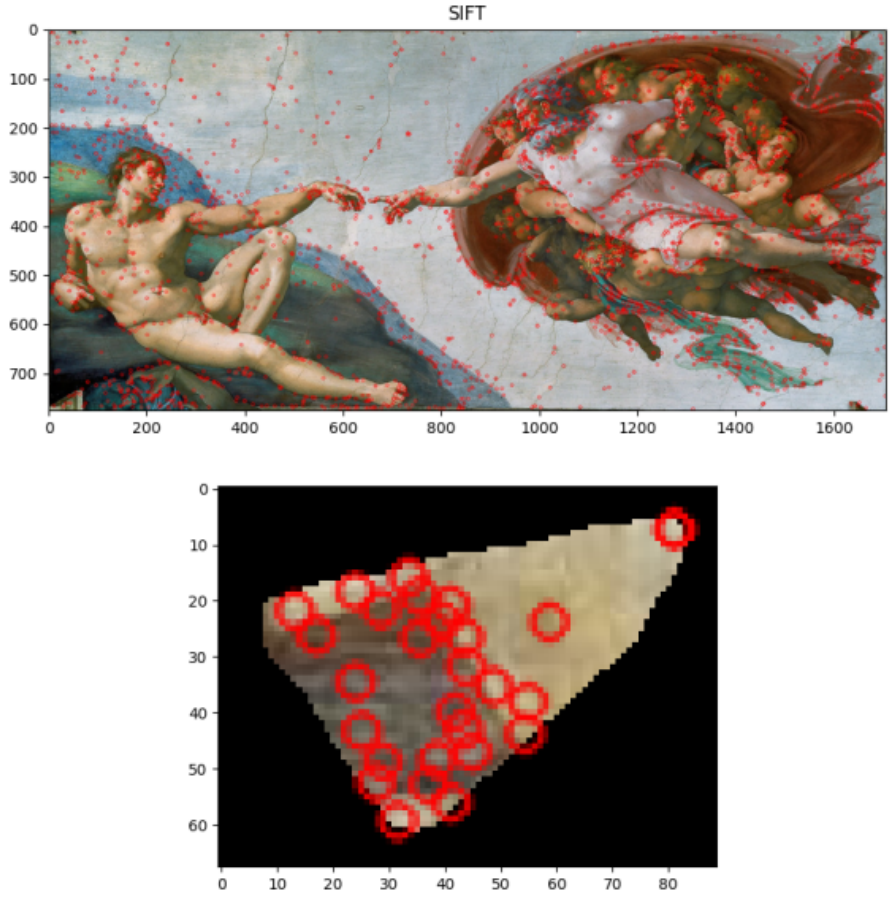

Figure 3: SIFT results with the fresco and the first fragment

Notes

– Low amount of points

- The points do not correlate well to the fresco, the lines in the sky are less detected than with the other methods
- No points are on the fragment border
- But no clear correlation

1.1.4 Surf

Not implemented by OpenCV ("*non-free*" module).
From the documentation `https://docs.opencv.org/4.x/df/dd2/tutorial_py_surf_intro.html`

1.1.5 Fast

From the documentation `https://docs.opencv.org/4.x/df/d0c/tutorial_py_fast.html`

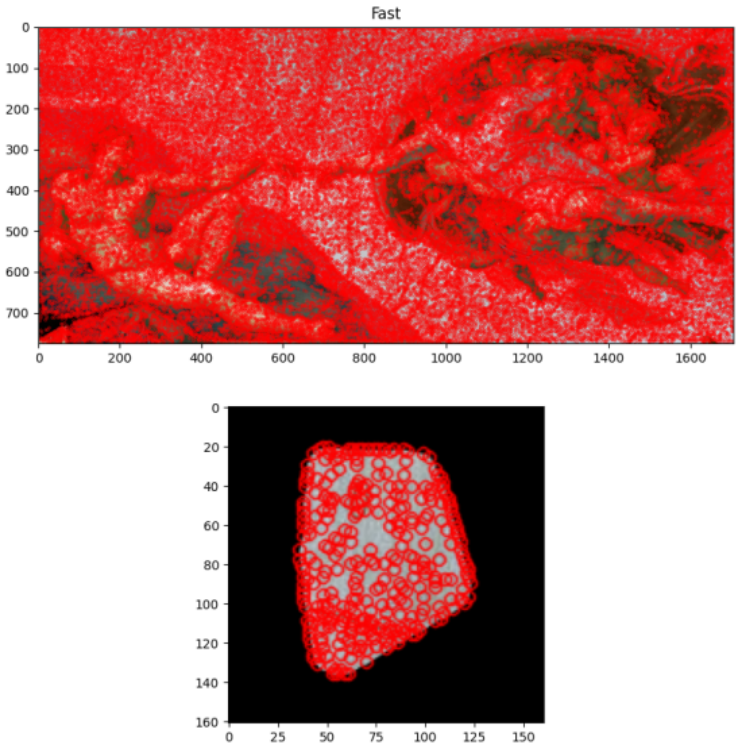

Figure 4: Fast results with the fresco and the fragment n°20

Notes

- Very high amount of points, good coverage but maybe too much
- Some points are on the fragment border, but they can be removed
- Too much detected points on the fragment

1.1.6 ORB

From the documentation `https://docs.opencv.org/4.x/d1/d89/tutorial_py_orb.html`

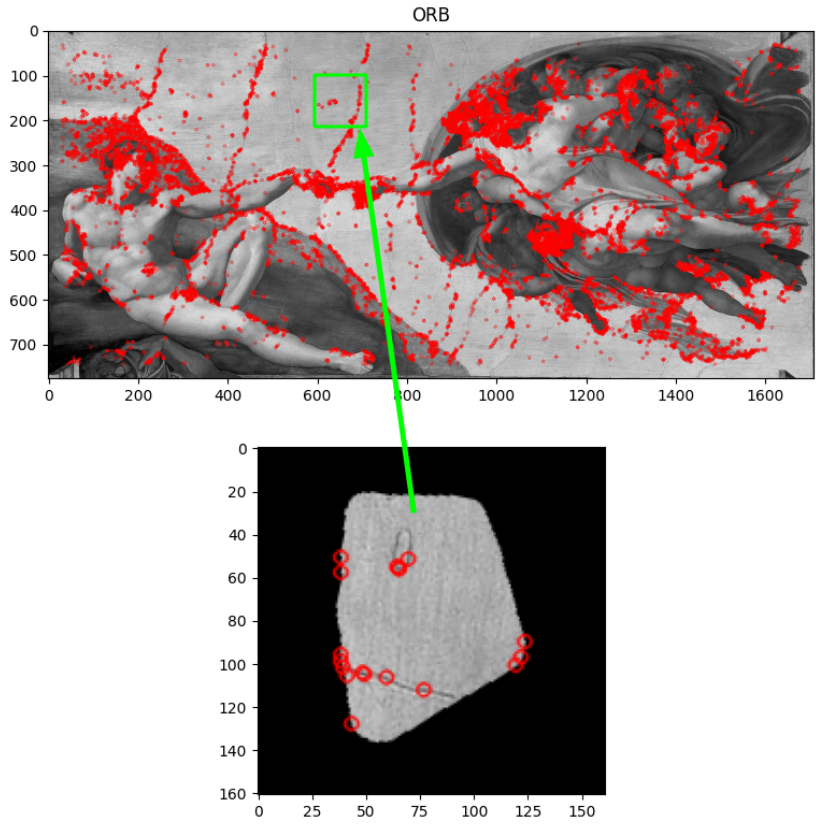

Figure 5: ORB results with the fresco and the fragment n°20

Notes

- High amount of points, a little bit of area with no points
- Some points are on the fragment border, but they can be removed
- Good correlation on the fresco and on the fragment

1.1.7 Conclusion

In short, point of interest detection methods:

Method	Fresco coverage	Fragment coverage
Harris	Bad	Bad
Shi-Tomasi	Good	Bad
SIFT 4	Too few points	Bad
Surf	-	-
Fast	Too much points	Too much points
ORB	Good	Good

The **ORB** method is the most convincing, so we are going to use it.

1.2 Benchmark of the associations detection methods

We need to find a method that can associate points of interest on the fragments to the fresco. Because we do not know the rotation of the fragment, the method has to be *'rotation compatible'*. But because the scale of the fragment is the same on the fresco, it does not need to be *'scale compatible'*.

1.2.1 Template Matching

Using the open CV functions : cv.matchTemplate() and cv.minMaxLoc() From the documentation `https://docs.opencv.org/4.x/d4/dc6/tutorial_py_template_matching.html`

Notes

– Can't work because our fragments are rotated

1.2.2 BFMatcher

Using the OpenCV function : `cv.BFMatcher`

From the documentation `https://docs.opencv.org/4.x/dc/dc3/tutorial_py_matcher.html`

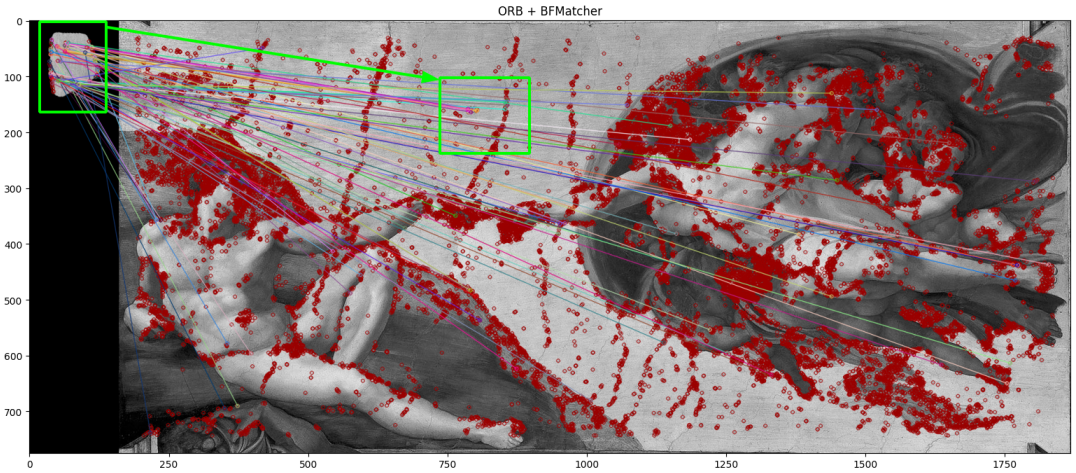

Figure 6: ORB + BFMatcher results with the fresco and the fragment n°20

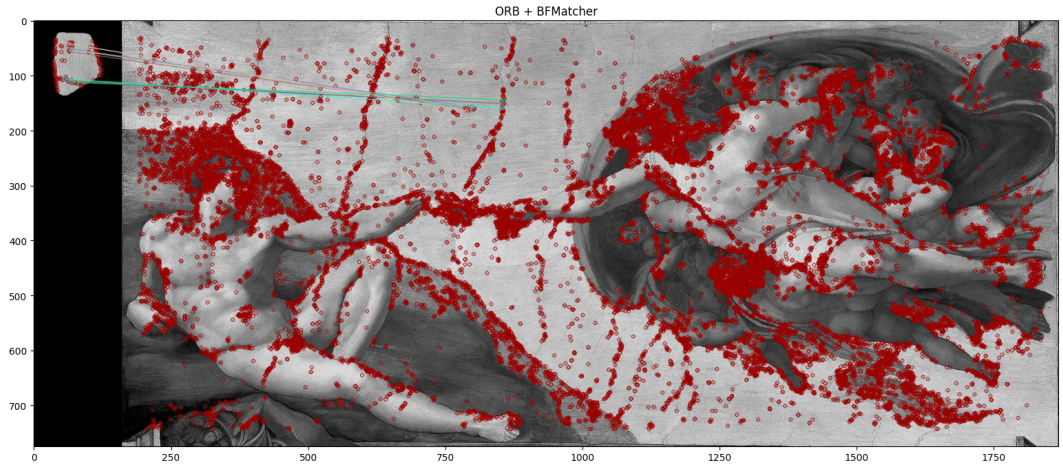

Figure 7: ORB + BFMatcher results with the fresco and the fragment n°20 with the 10 best matches

Notes

– The BFMatcher looks promising, as it has found the position of the fragment on the first try

1.2.3 BFMatcher with KnnMatch

Using the open CV functions : `cv.knnMatch`

From the documentation `https://docs.opencv.org/4.x/dc/dc3/tutorial_py_matcher.html`

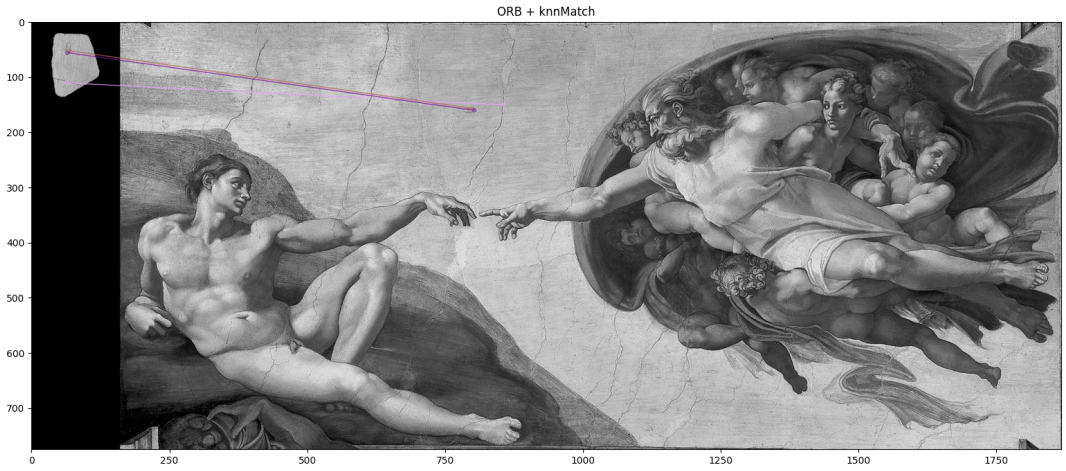

Figure 8: ORB + KnnMatch results with the fresco and the fragment n°20

Notes

– BFMatcher with KnnMatch works well too

1.2.4 Conclusion

In short, point of interest association methods:

Method	Rotation compatible	Does it work ?
Template Matching	No	No
BFMatcher	Yes	Very well
BFMatcher + KnnMatch	Yes	Very well

The **BFMatcher** with the **KnnMatch** method is very convincing, so we are going to use it.

1.3 Test on all the fragments

To succeed in the exercise, we need to find match for the maximum number of fragments. So we made several tests with ORB and BFMatcher / Knn with different parameters.

The parameters are : **Fresco kp**, number of key-points on the fresco and **Dist. threshold**, distance in px for two key-points to be considered as a match.

And we are evaluating : **Frag. 2 or more kp**, the number of fragments with 2 or more detected key-points(pk) on it and **Frag. 2 or more matches**, the number of fragments with 2 or more key-point association matches. We want them both to be the higher possible.

N°	Fresco kp	Dist. threshold	Frag. 2 or more kp	Frag. 2 or more matches
1	30000	0.75px	232/328 (70%)	70/232 (30%)

As we can see on this first test, 70% of the fragments have 2 or more detected key-points and only 30% of those have 2 or more matches, we need to improve that.

N°	Fresco kp	Dist. threshold	Frag. 2 or more kp	Frag. 2 or more matches
2	30000	0.85px	232/328 (70%)	145/232 (62%)
3	30000	0.95px	232/328 (70%)	209/232 (90%)

We can see that by increasing the distance in pixels for two key-points to be considered as a match, more matches are detected, but the matches are less accurate :

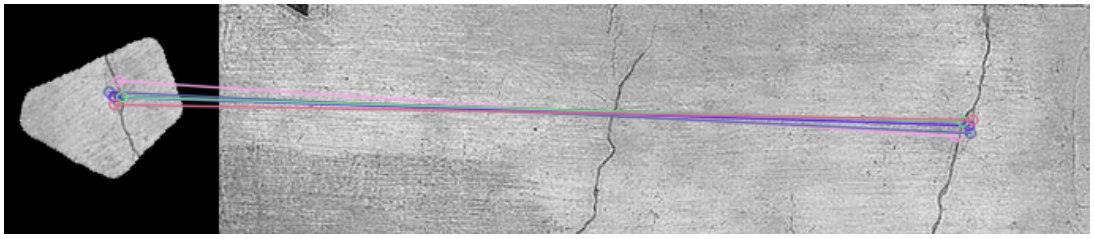

Figure 9: Matches on the fragment n°58 with a dist threshold of 0.75px

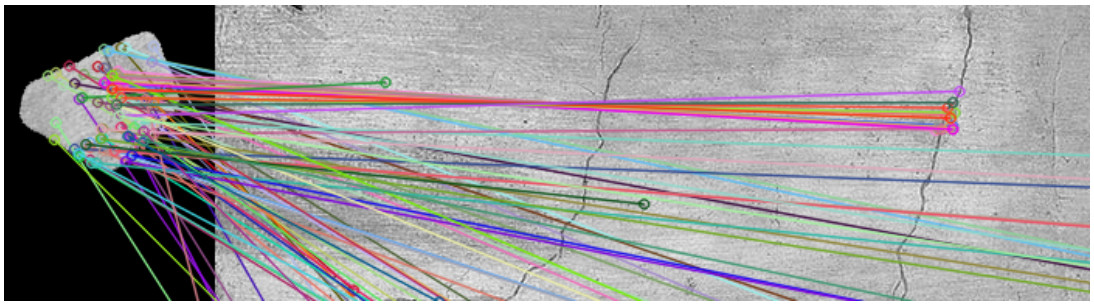

Figure 10: Matches on the fragment n°58 with a dist threshold of 0.95px

2 Exercise 2

2.1 Question 1

2.1.1 Minimum number of matches K

To find x, y (coordinates of the fragment pivot $\mathcal{C}$, its center, on the fresco), θ (rotation to apply, in degrees), we need at least **2** matches of key-points because a straight line needs at least two points to be defined. Using this straight line on the fragment and on the fresco, we can compute the rotation with **tangent** formula.

2.1.2 How to compute fragment parameters

Known and useful parameters Let's say we have our two associations on two key-points, $\mathcal{A}'$ and $\mathcal{B}'$ on the fragment, associated each other to $\mathcal{A}$ and $\mathcal{B}$ on the fresco.

- Coordinates of $\mathcal{A}'$ on the fragment : $(x_{\mathcal{A}'}, y_{\mathcal{A}'})$
- Coordinates of $\mathcal{A}$ on the fresco : $(x_{\mathcal{A}}, y_{\mathcal{A}})$
- Coordinates of $\mathcal{B}'$ on the fragment : $(x_{\mathcal{B}'}, y_{\mathcal{B}'})$
- Coordinates of $\mathcal{B}$ on the fresco : $(x_{\mathcal{B}}, y_{\mathcal{B}})$

The **pivot** on the fragment, $\mathcal{C}'$, is located at the center, in other words at $(\frac{width-1}{2}, \frac{height-1}{2})$.

Finding rotation θ To get rotation θ, we have to determine the vectors $\vec{u}$ and $\vec{v}$ associated respectively to the two straight lines $(\mathcal{AB})$ and $(\mathcal{A}'\mathcal{B}')$.

- $\vec{u} \begin{pmatrix} x_{\mathcal{B}} - x_{\mathcal{A}} \\ y_{\mathcal{B}} - y_{\mathcal{A}} \end{pmatrix}$
- $\vec{v} \begin{pmatrix} x_{\mathcal{B}'} - x_{\mathcal{A}'} \\ y_{\mathcal{B}'} - y_{\mathcal{A}'} \end{pmatrix}$

If these vectors are not collinear, we can find the intersection angle θ between the two lines using the **cosine formula** :

$$cos(\theta) = \frac{\vec{u}.\vec{v}}{||\vec{u}||.||\vec{v}||}$$

$$\boxed{\Leftrightarrow \theta = cos^{-1}(\frac{\vec{u}.\vec{v}}{||\vec{u}||.||\vec{v}||})}$$

where $\vec{u}.\vec{v}$ is the **dot product** of $\vec{u}$ and $\vec{v}$:

$$\vec{u}.\vec{v} = u_1 * v_1 + u_2 * v_2 = (x_{\mathcal{B}} - x_{\mathcal{A}}) * (x_{\mathcal{B}'} - x_{\mathcal{A}'}) + (y_{\mathcal{B}} - y_{\mathcal{A}}) * (y_{\mathcal{B}'} - y_{\mathcal{A}'})$$

and $||\vec{u}||$ and $||\vec{v}||$ the **vectors lengths** :

$$||\vec{u}|| = \sqrt{u_1^2 + u_2^2} = \sqrt{(x_{\mathcal{B}} - x_{\mathcal{A}})^2 + (y_{\mathcal{B}} - y_{\mathcal{A}})^2}$$
$$||\vec{v}|| = \sqrt{v_1^2 + v_2^2} = \sqrt{(x_{\mathcal{B}'} - x_{\mathcal{A}'})^2 + (y_{\mathcal{B}'} - y_{\mathcal{A}'})^2}$$

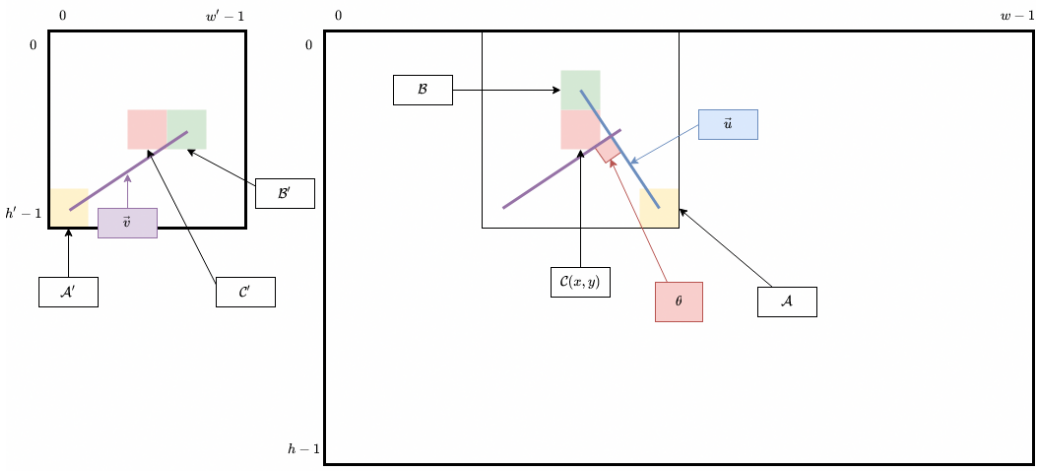

Figure 11: Finding angle θ using match coordinates

For now θ is in radian so for the solution output and placing, we need to convert it into **degrees** :

$$\theta_{(deg)} = \frac{360 \times \theta_{(rad)}}{2\pi} = \frac{180 \times \theta_{(rad)}}{\pi}$$

Finding pivot coordinates (x, y) Now that we have θ, we can find the pivot coordinates on the fresco.

First of all, we have to get the **translation** t of the fragment.

Using the rotation matrix R for left-handed coordinate system :

$$R = \begin{bmatrix} \cos\theta & \sin\theta \\ -\sin\theta & \cos\theta \end{bmatrix}$$

The rotated coordinates (x_k^r, y_k^r) of the two key points are given by :

$$\begin{bmatrix} x_k^r \\ y_k^r \end{bmatrix} = R\mathbf{v} = \begin{bmatrix} \cos\theta & \sin\theta \\ -\sin\theta & \cos\theta \end{bmatrix} \begin{bmatrix} x'_k \\ y'_k \end{bmatrix} = \begin{bmatrix} x'_k \cos\theta + y'_k \sin\theta \\ -x'_k \sin\theta + y'_k \cos\theta \end{bmatrix}$$

Using one key point, we can now get the **translation** t, as we know the coordinates (x_k, y_k) on the fresco :

$$t = \begin{bmatrix} x_k - x_k^r \\ y_k - y_k^r \end{bmatrix}$$

To get the final pivot $\mathcal{C}$ coordinates (x, y), we have to apply R then t to the fragment coordinates :

$$\boxed{\begin{bmatrix} x \\ y \end{bmatrix} = t + R \begin{bmatrix} x' \\ y' \end{bmatrix}}$$

2.2 Question 2

2.2.1 Approach

Model initialization

1. We pick randomly n matches (minimum 2). If the fragment matches count is exactly n (when $n > 2$), we then stop the loop and return the model with these parameters.

2. A model (x, y, θ) is generated depending on the matches parameters.

Error threshold To check if a key point, computed using a model, corresponds to the key point on the fresco, we use a **threshold** because it is not always perfect. We see if the key point is somewhere in a circle around the fresco key point. The circle radius is defined by `RADIUS_THRESHOLD`.

Working with correct matches To reduce the number of wrong matches, we sort first by `distance` parameter given by `DMatch` class, that represents the degree of similarity between two descriptors. We keep only a percentage (that can be set by `GOOD_MATCH_PERCENT` constant) of first sorted matches (see `get_good_matches` function).

Choosing the best model We had multiple ways to do so, such as looking at error for each model inlier and taking the lowest error. At the end, we had followed the instruction of the exercise and took only the model with the greatest number of inliers.

2.2.2 Result

See below in exercise 3 for the reconstitution.

The result is a bit satisfying but not perfect though. Some fragments are well placed while others are placed on the right area but not with the right rotation.

Depending on the parameters (maximum number of iterations, error threshold, number N of matches required to create a model, matcher threshold and good matches percentage), the algorithm can take more or less time up to 10 minutes.

If the matcher threshold is high (> 0.85), it would be interesting to set the good matches percentage to a lower value (e.g. 20%) so that the noise is less considered.

For Domenichino's fresco, the performances are worst because its size is bigger than Michelangelo's fresco.

2.3 Question 3

2.3.1 Approach

After RANSAC filtering, we check if the number of model inliers K is greater than N. If so, then we call a function, `get_model_regression`, that computes a model taking into account all inliers parameters.

Its parameters, x, y and θ are the average of all possible models generated using the inliers.

2.3.2 Result

We do not see much difference between the version with regression and the previous version. Few fragments are well placed or their position and rotation are sometimes rectified.

Maybe we did not implemented the right required regression. It is hard to tell as the results may vary per run.

All the results below have a `THRESHOLD_RADIUS` set to 15 pixels.

With parameters (`MATCHER_THRESHOLD` = 0.9, `GOOD_MATCH_PERCENT` = 0.15, `MAX_ITER` = 5000) :

Figure 12: Results without regression and with regression

With parameters (`MATCHER_THRESHOLD` = 0.75, `GOOD_MATCH_PERCENT` = 0.3, `MAX_ITER` = 10000) :

Figure 13: Results without regression and with regression

With parameters (`MATCHER_THRESHOLD` = 0.85, `GOOD_MATCH_PERCENT` = 0.3, `MAX_ITER` = 10000) :

Figure 14: Results without regression and with regression

2.4 Difficulties encountered

- Finding θ rotation angle and the center C coordinates was tricky, we had to use simple parameters for $\mathcal{A}$ and $\mathcal{B}$ and make drawings to understand the mathematics behind it.
- We could improve the RANSAC filter and regression but we missed the time to do so.

2.5 Possible improvements

- Compute a model with $N > 2$ random matches (we made an attempt but the result was less satisfying than with $N = 2$ matches).
- Detect if all possible model matches have been done so that we can stop the loop, to improve algorithm performance.
- Compare model and match key points pixel colors.
- Compare model and match key points angles.

3 Exercise 3

3.1 First solution evaluation

We are now evaluating our generated results with our previously made evaluation tool.

This testing phase will also allow us to try different parameters and observe their effect. The first parameter that we have tried is the matching iteration number from 10 to 10000 :

```
Using the following margins delta_x=1, delta_y=1 and delta_alpha=1
solution_ORB_BFMATCHER_10000_MAX_ITER, with 177 missing, 0 wrong, 113 incorrect, and 9/299 good fragments, has a surface precision of 6%
solution_ORB_BFMATCHER_1000_MAX_ITER, with 177 missing, 0 wrong, 110 incorrect, and 12/299 good fragments, has a surface precision of 8%
solution_ORB_BFMATCHER_100_MAX_ITER, with 177 missing, 0 wrong, 112 incorrect, and 10/299 good fragments, has a surface precision of 7%
solution_ORB_BFMATCHER_10_MAX_ITER, with 177 missing, 0 wrong, 114 incorrect, and 8/299 good fragments, has a surface precision of 6%
solution_ORB_BFMATCHER_3000_MAX_ITER, with 177 missing, 0 wrong, 113 incorrect, and 9/299 good fragments, has a surface precision of 6%
solution_ORB_BFMATCHER_5000_MAX_ITER, with 177 missing, 0 wrong, 113 incorrect, and 9/299 good fragments, has a surface precision of 6%
```

Figure 15: Evaluation of our first solution with different iteration number with dx=1 dy=1 da=1

```
Using the following margins delta_x=60, delta_y=60 and delta_alpha=60
solution_ORB_BFMATCHER_10000_MAX_ITER, with 177 missing, 0 wrong, 83 incorrect, and 39/299 good fragments, has a surface precision of 24%
solution_ORB_BFMATCHER_1000_MAX_ITER, with 177 missing, 0 wrong, 84 incorrect, and 38/299 good fragments, has a surface precision of 23%
solution_ORB_BFMATCHER_100_MAX_ITER, with 177 missing, 0 wrong, 86 incorrect, and 36/299 good fragments, has a surface precision of 22%
solution_ORB_BFMATCHER_10_MAX_ITER, with 177 missing, 0 wrong, 90 incorrect, and 32/299 good fragments, has a surface precision of 20%
solution_ORB_BFMATCHER_3000_MAX_ITER, with 177 missing, 0 wrong, 81 incorrect, and 41/299 good fragments, has a surface precision of 24%
solution_ORB_BFMATCHER_5000_MAX_ITER, with 177 missing, 0 wrong, 87 incorrect, and 35/299 good fragments, has a surface precision of 22%
```

Figure 16: Evaluation of our first solution with different iteration number with dx=60 dy=60 da=60

As we can see on the evaluation, the surface precision of our first solution is not great and maximum number of iterations do not have a clear effect on the precision.

We can clearly observe with the next comparison that there only are few differences between each solution with different iterations.

Figure 17: Results with 3000 5000 and 10000 iterations

3.2 Possible improvements

To improve our surface precision we need to find matches for more fragments. Indeed, even if only 30% of our fragment position guesses are correct, more than 50% of the fragments are missing from our solution.

4 Exercise 4

4.1 Euclidean distance

Strategy There is a property that tells that the distance between two points is the same on the fragment and on the fresco.

To take advantage of this property, we have built this strategy :

- Take the **two** best matches between a fragment and the fresco
- Calculate the distance between the two points on the fragments and the two points on the fresco
- If the two distances are different by a certain pixel number (let say 2px), we replace one of the two match with another and calculate the distances again
- Repeat until each combination of matches has been tested (in this case, there is no match) or if one of the match couples has points distances closer than our parameter (in this case, we have a match).

Pros and cons **Pros :**

- The strategy is very simple.
- This strategy greatly limits the incoherence of the matches.
- Low calculation time with a possibility of multi threading

Cons :

- If there is many matches, the calculation time might be longer.
- Having equal distances doesn't mean that the match is good, more test are required like testing that the colors are more or less the same.