Introduction to Computer Vision – Homework 2
RA192617 – Edgar Rodolfo Quispe Condori
RA192618 – Darwin Ttito Concha
Institute of Computing, University of Campinas (UNICAMP)
Campinas-SP, Brazil, 13083-852

1 Input data

In this project we use three different videos, the technical details for each video are shown in table 1.

Complete results of the experiments are available at: *this link*. Successful stabilization with keypoints matching will be download in the directory **myoutput** with make command.

Name	Time(seconds)	Frames #	Problem
p2-1-0.avi	20	198	Scale
p2-2-0.av	26	260	Translation and Rotation
p2-2-0.avi	10	104	Rotation

Table 1: Technical detail of input videos.

The original videos were recorded in 30fps but we considers 10fps to reduce the experiments execution time. The size of each frame is 640×480 pixels.

2 Keypoints Detector and Descriptor

The algorithms that we implemented are based on the ORB to detect and SIFT to describe keypoints, it has slightly modification that were done based on experimental results and requested task.

2.1 Keypoints Detector

We used ORB like interest point detector: the original approach uses FAST(Features from Accelerated Segment Test) to detect interests points and assigns an orientation based in the intensity centroid.

Originally, FAST uses non-maximum suppression for removing adjacent corners and machine learning to improve the performance of the algorithm. In our implementation we do not use machine learning because the explanation is not clear. Our implementation consider three parameters :

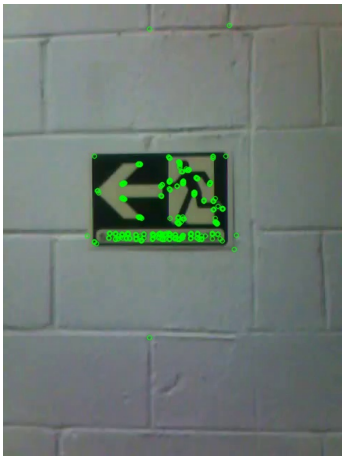
- *Threshold*: this parameter define if each of the sixteen neighbor points is lighter, darker or similar to reference point.
- *N* : condition to determine if a point is a keypoint(at least N consecutive neighbors must be lighter or darker to reference point).
- *Non-Maximum Suppression*: this parameter signals when remove adjacent corners that are too close between them.

FAST does not produce a measure of cornerness (it has large responses along edges). As similar to the original implementation of ORB we employ a Harris corner measure to order and select P keypoints. Another limitation of FAST is that it does not produce orientation. To compensate this issue we use a approach similar to SIFT(ORB use moments of the patch around the keypoint). This technique take a window around of each keypoint and collect gradient directions and magnitudes. Then, a histogram is created for this and the amount added to the bin of the histogram is proportional to the magnitude of gradient at each points. Also, we used a gaussian weighting function, this function is multiplied by the magnitude. The farther away, the lesser the magnitude weight.

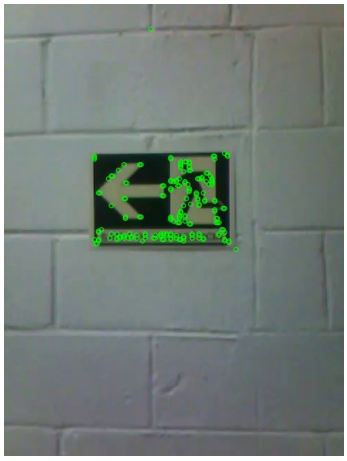
Figure 1 shows results with different parameter settings.

According to figure 1 we can see that with parameter *threshold*, the number of keypoints grows inversely proportional to its value. If we choose a too low value, we start getting false positives and with a too big value, we start ignoring keypoints. This same effect occurs with the parameter N.

In the case of *No-Maximal Suppression*, it removes close keypoints, but we have found that this redundancy of keypoints is util for stabilization.



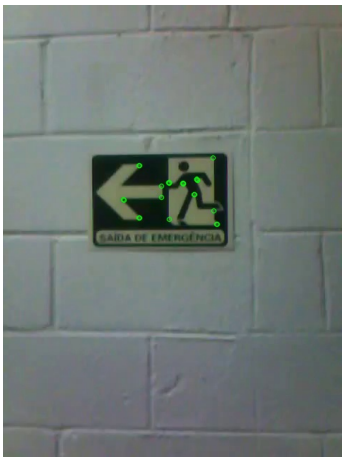
(a) (20, 8, False). 987 keypoints detected



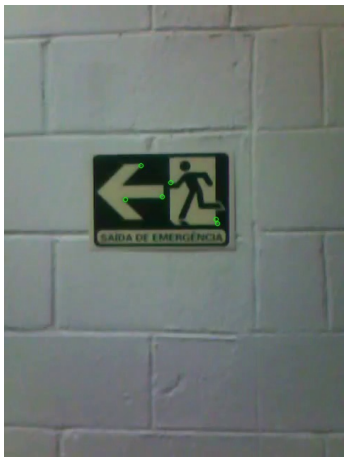
(b) (20, 8, True). 185 keypoints detected

(c) (30, 8, False). 328 keypoints detected

(d) (30, 8, True). 68 keypoints detected



(e) (40, 9, False). 24 keypoints detected



(f) (40, 9, True). 6 keypoints detected

Figure 1: Results of experiment with different parameter settings. The label of each subfigure follows the following format: *(threshold, N, Non-maximum suppression)* and the number of keypoints detected.

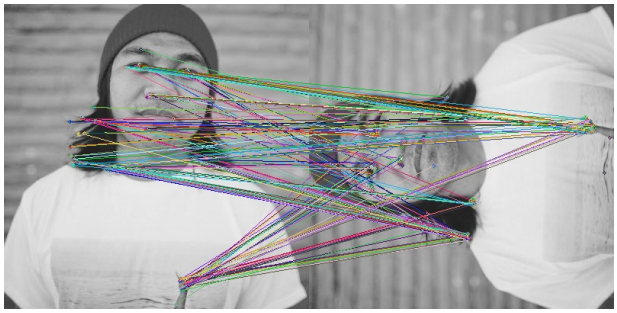
2.2 Keypoints Descriptor

The original SIFT considers a pyramid in order to be invariant to scale, the keypoint descriptor is then computed from a specific level of this pyramid. Analysing the data it is easy to see that the scale difference between adjacent frame is negligibly. Thus, our implementation ignores this stage in order to improve the performance of the algorithm.

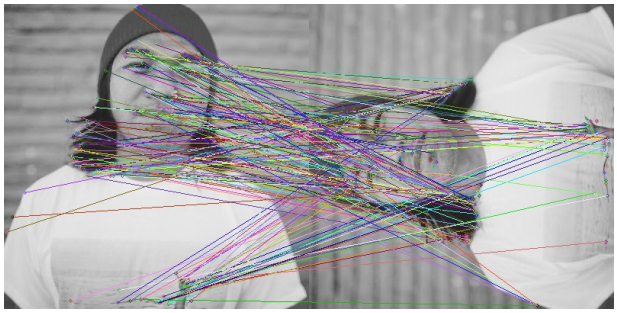
Figure 2 shows results of the implementation.

We can see in figure 2 that the use of Non-Maximum Suppression create more outlier(wrong matches), this is because removing points that are close to one that will be correctly match decrease the number of inliers.

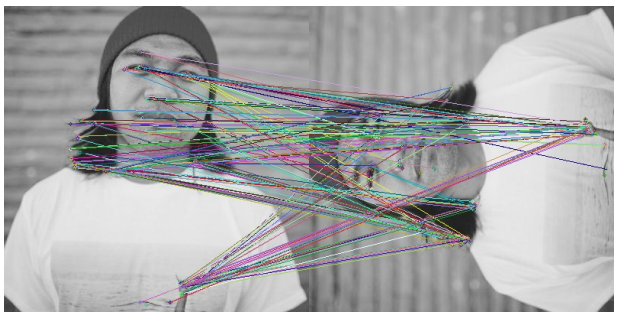
Our experiments suggest to use a setting of (30,8,False).



(a) (20, 8, False).

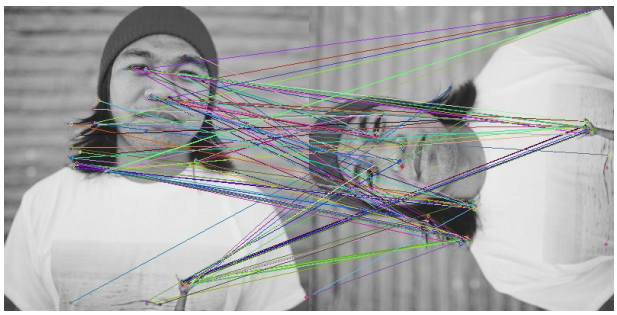


(b) (20, 8, True).

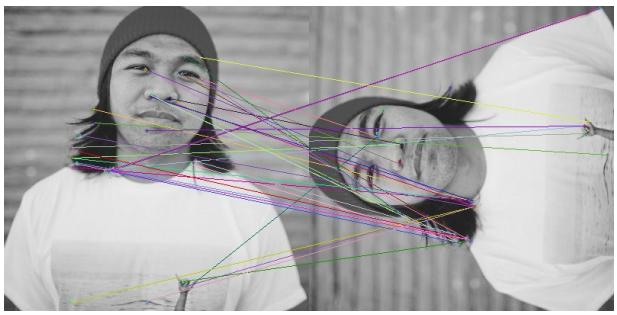


(c) (30, 8, False).

(d) (30, 8, True).



(e) (40, 9, False).



(f) (40, 9, True).

Figure 2: Results of matches with different parameter settings. The label of each subfigure follows the following format: *(threshold, N, Non-maximum suppression)*

3 Match hypothesis

Our implementation considers a simple match algorithm(brute force). For each keypoints of frame i we find the closest match in the frame $i + 1$ iterating over all possible candidates. We compare *L2 Norm* and *cosine* metrics to measure the similarity. Note that this implementation can generate the same match for two different keypoints yielding in an outlier.

An idea for performance improvement of this algorithm, is to use *KNN(K-Nearest Neighbors)*. For real time application this may be a good option because its complexity is $O(n \log n)$, while our approach is $O(n^2)$ (n is the number of keypoints).

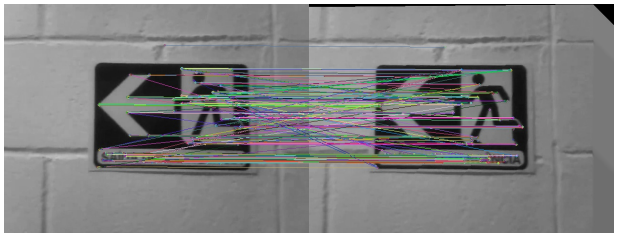
Figure 3 shows the different result comparing *L2 Norm* and *cosine* metrics.

We can see in figure 3 that the result of the experiment are very similar because there is not such an extreme difference between the performance of these metrics. There is an interesting difference in the execution time, as shown in table 2. L2-norm is slightly faster.

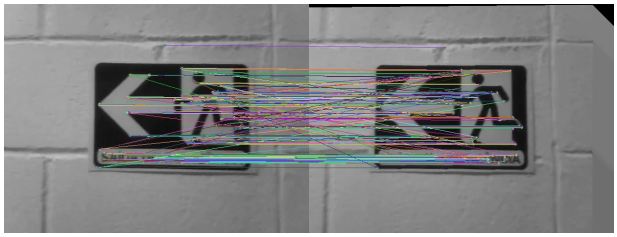
Name	**Time per frame**	
	L2-norm	**Cosine**
p2-1-0.avi	54.446	64.327
p2-2-0.av	37.321	42.982
p2-2-0.avi	54.446	45.262

Table 2: Execution time comparison.

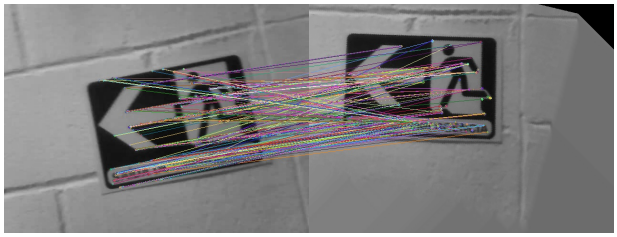
We use *cosine* distance as main similarity metric.



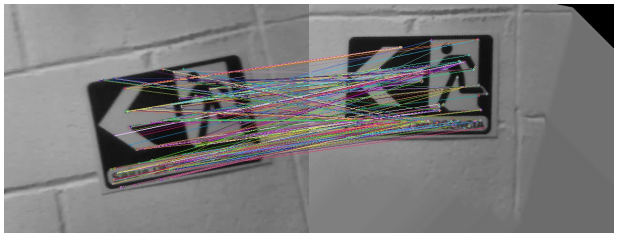
(a) Match hypothesis with cosine distance



(b) Match hypothesis with l2-norm distance



(c) Match hypothesis with cosine distance



(d) Match hypothesis with l2-norm distance

Figure 3: Comparison of match hypothesis with cosine and l2-norm

4 Affine Tranformation Fitting

Our implementation considers **Affine and Projective** transformation.

The number of iteration of RANSAC has been determined experimentally, our implementation has the following modules:

```
'''
    Computes least square
'''
def least_square(src, dst, matches, k_points, transformation = 'affine'):
'''
    Evaluates a transformation parameters
'''
def evaluate_transformation(src, dst, matches, trans_params, threshold = 1, transformation = 'affine'):
'''
    Computes RANSAC
'''
def ransac(src, dst, matches, k = 3, S = 35, threshold = 1, transformation = 'affine'):
```

- *least_square*: this function creates the matrix X, A and Y, depending of the parameter $transformation$. Given N points $x_i, y_i (1 \leq i \leq N)$, in case of affine transformation, the matrix X and Y have to rows for each point:

$$X = \begin{bmatrix} x_1 & y_1 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & x_1 & y_1 & 1 \\ x_2 & y_2 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & x_2 & y_2 & 1 \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ x_N & y_N & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & x_N & y_N & 1 \end{bmatrix} \quad Y = \begin{bmatrix} x'_1 \\ y'_1 \\ x'_2 \\ y'_2 \\ \vdots \\ x'_N \\ y'_N \end{bmatrix}$$

  The parameters we want to find are defined for matrix A, in this case at least three points are needed and the transformation is computed using equation 1.

$$\begin{aligned} x' &= ax + by + c \\ y' &= dx + ey + f \end{aligned} \tag{1}$$

$$A = \begin{bmatrix} a \\ b \\ c \\ d \\ e \\ f \end{bmatrix}$$

  In the case of the projective transformation, the matrix X and Y are defined by:

$$X = \begin{bmatrix} x_1 & y_1 & 1 & 0 & 0 & 0 & -x'_1x_1 & -x'_1y_1 \\ 0 & 0 & 0 & x_1 & y_1 & 1 & -y'_1x_1 & -y'_1y_1 \\ x_2 & y_2 & 1 & 0 & 0 & 0 & -x'_2x_2 & -x'_2y_2 \\ 0 & 0 & 0 & x_2 & y_2 & 1 & -y'_2x_2 & -y'_2y_2 \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ x_N & y_N & 1 & 0 & 0 & 0 & -x'_Nx_N & -x'_Ny_N \\ 0 & 0 & 0 & x_N & y_N & 1 & -y'_Nx_N & -y'_Ny_N \end{bmatrix} \quad Y = \begin{bmatrix} x'_1 \\ y'_1 \\ x'_2 \\ y'_2 \\ \vdots \\ x'_N \\ y'_N \end{bmatrix}$$

The parameters we want to find are defined for matrix A, in this case at least four points are needed and the transformation is computed using equation 2.

$$\begin{aligned} x' &= ax + by + c - gx'x - hx'y \\ y' &= dx + ey + f - gy'x - hy'y \end{aligned} \tag{2}$$

$$A = \begin{bmatrix} a \\ b \\ c \\ d \\ e \\ f \\ g \\ h \end{bmatrix}$$

Matrix A in both cases can be computed using least square. We use $numpy$ for easy implementation, sometimes the inverse does not exists, thus we return and empty vector.

```
x_transpose = np.matrix.transpose(X)
A = np.dot(x_transpose, X)
if np.linalg.det(A) == 0:
  print('Points', k_points, 'are not suitable for the transformation')
  return []
A = np.dot(np.linalg.inv(A), np.dot(x_transpose, Y))
return A
```

- *evaluate_transformation*: this function evaluates how many points fit correctly the compute transformation based on a threshold. The book suggests it to be between one and three. In our implementation we consider a value of two because our matches in the previous step are not perfect and we want to avoid wrong fittings because of outliers. The output of this function are the indexes of the points that fit the given parameters.
- *ransac*: this function computes the RANSAC algorithm, iterating S times, picking k aleatory points. Our main assumptions is that at least there is one valid solution that can be reached in S iterations. The parameter k is set to three for affine and four for projective transformation.

5 Transform

In order to explain our algorithm consider the following notation: i refers to the $i-th$ frame in the original video, and i' refers to the $i-th$ frame in the stabilized video.

Initially, we considered the following algorithm to compute i':

1. extract keypoints(X) from $(i-1)'$
2. extract keypoints(Y) from i
3. find matches(M) between X and Y
4. compute transformation T from M.
5. apply transformation T to i

With this approach, we were obtaining a lot of transformations errors(figure 4) because the differences between i and $(i-1)'$ are complex. This happens because most of the matches are outliers, hence, RANSAC finds a transformation that is wrong yielding in these strange but nice figures 4.

Due to error described above, we defined another approach:

1. extract keypoints(X) from $i-1$

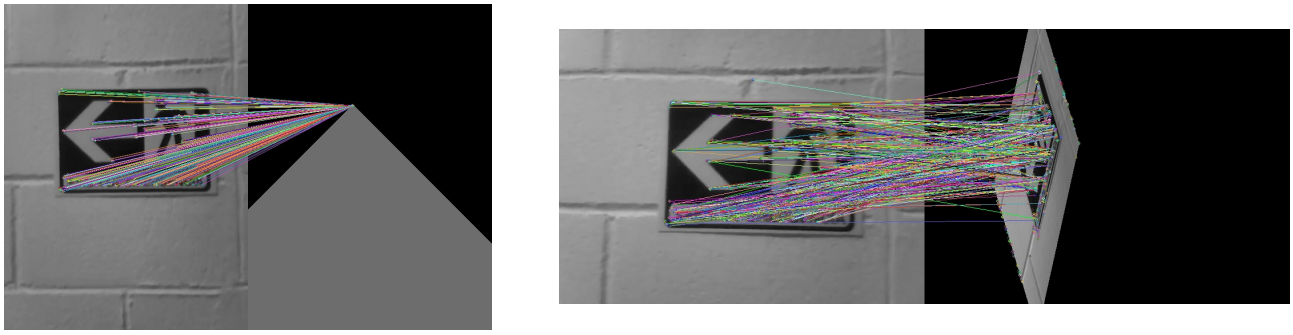


Figure 4: Errors of transformation using initial approach

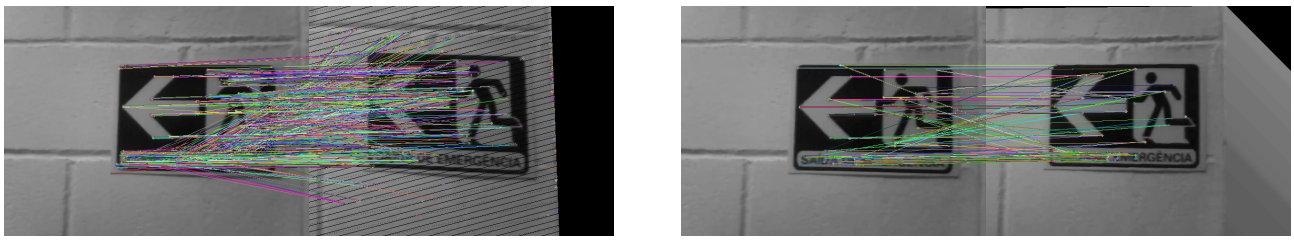
Parameters	Definition	Values
S	# of iterations of RANSAC	36
		100
		200
Non maximum suppression	remove close keypoints	TRUE
		FALSE
Distance metric	similarity between keypoints	L2-norm
		cosine
ORB setting(threshold, N)	keypoints detector	(30,8)
		(30,12)

Table 3: Parameters evaluated.

2. transform keypoints location of $i-1$ based on $(i-1)'$
3. extract keypoints(Y) from i
4. find matches(M) between X and Y
5. compute transformation T from M.
6. apply transformation T to i

We assume that the difference between adjacent frame in the original video is small. Thus, we compare the feature vector of i and $i-1$, but the keypoints location of i and $(i-1)'$.

Once we have the transformation of parameters we apply it for every pixel in i, depending of the movement, some black holes and lines appears(Figure 5). Thus, we interpolate this points with the average of four neighbors and fill with zero the cases when the neighbors are also empty.



(a) Without interpolation

(b) With interpolation

Figure 5: Comparison of results of transformation using interpolation

6 Experiments

In order to measure the stabilization, we use MSE(mean square error). We make experiments changing the parameters show in table 3:

6.1 S

For experiment with this parameter, we set: $non{-}maximum\ suppression = False, distance\ metric = cosine, ORB\ setting = (30, 8)$, because this configuration is the best based on previous sections.

The results of MSE and execution time(mean per frame) are shown in table 4

Name	S	MSE	Execution Time	Successful
p2-1-0.avi	36	25921522.78	62.324	Yes
	100	24672510.21	64.327	Yes
	200	24540916.12	62.691	Yes
p2-1-1.avi	36	44613346.37	43.003	No
	100	30052053.01	42.982	Yes
	200	29430481.39	42.4314	Yes
p2-1-2.avi	36	135858245.35	46.593	No
	100	44096547.92	45.262	Yes
	200	28768129.31	45.623	Yes

Table 4: Results for paratemer *S*.

The book suggest to set $S = 36$, but in two cases this generates unsuccessful stabilization because the number of outlier is high and a bigger number of iteration is needed.

In general setting a bigger value for S results in lower MSE, moreover, when this value is duplicated the execution time grows just a little. Due to this analysis, we consider $S = 200$ as best parameter.

6.2 Non-Maximum Suppression

For experiment with this parameter we set: $S = 200, distance\ metric = cosine, ORB\ setting = (30, 8)$.

The results of MSE and execution time(mean per frame) are shown in table 5

Name	Non-maximum suppression	MSE	Execution Time	Succesfull
p2-1-0.avi	True	42818733.94	15.791	No
	False	24540916.12	62.691	Yes
p2-1-1.avi	True	2531291808.00	16.917	No
	False	29430481.39	42.431	Yes
p2-1-2.avi	True	194895992.71	13.706	No
	False	28768129.31	45.623	Yes

Table 5: Results for paratemer *Non-maximum suppression*.

We can see that the use Non-maximo suppression has a big impact, using it creates too little keypoints and the stabilization is not possible. Thus we consider not to use Non-maximum suppression.

6.3 Distance metric

For experiment with this parameter we set: $non - maximum\ suppression = False, S = 200, ORB\ setting = (30, 8)$.

The results of MSE and execution time(mean per frame) are shown in table 6

Name	Ditance metric	MSE	Execution Time	Succesfull
p2-1-0.avi	L2-norm	24429436.76	54.446	Yes
	cosine	24540916.12	62.691	Yes
p2-1-1.avi	L2-norm	29964112.06	37.321	Yes
	cosine	29430481.39	42.431	Yes
p2-1-2.avi	L2-norm	24429436.76	54.446	Yes
	cosine	28768129.31	45.623	Yes

Table 6: Results for paratemer *Distance metric*.

From the results it seems that $L2 - norm$ is far much better than $cosine$, but considering that one shift of pixels location in the transformed frame can generate an error cost of 307200, this difference turns smaller. We decided use $cosine$.

6.4 ORB setting(threshold, N)

For experiment with this parameter we set: $non - maximum\ suppression = False, distance\ metric = cosine, S = 200$.

The results of MSE and execution time(mean per frame) are shown in table 7

Name	ORB setting(threshold, N)	MSE	Execution Time	Succesfull
p2-1-0.avi	(30,12)	2330901497.00	43.359	No
	(30,8)	24540916.12	62.691	Yes
p2-1-1.avi	(30,12)	89788339.90	29.774	No
	(30,8)	29430481.39	42.431	Yes
p2-1-2.avi	(30,12)	1185961909.67	22.773	No
	(30,8)	28768129.31	45.623	Yes

Table 7: Results for paratemer *ORB setting*

Similar to the Non-maximum suppression parameter, try to generate less keypoints yields in unsuccessful stabilization.