MO446 Computer Vision: Project 3

Juan Hernández
RA: 163128
Email: juan.albarracin@ic.unicamp.br

Miguel Rodriguez
RA: 192744
Email: m.rodriguezs1990@gmail.com

I. Introduction

This work consists of reconstructing a scene in three dimensions from a set of images at different angles of the same scene. To do this task, it is necessary to perform an extraction of keypoints from the first image, then tracking all the keypoints extracted along the different images. This will be done using algorithms of optical flow. Then with the results of the tracking, the algorithm of structure for motion will be used in order to reconstruct the 3D scene of the captured object and find the positions of the cameras in the real-world space.

II. Keypoint selection

For feature selection, four methods were tested: SIFT [1], Harris Corner Detector, Good Features to track [2] and Orb [3]. To perform the tests, we used the OpenCV implementations.

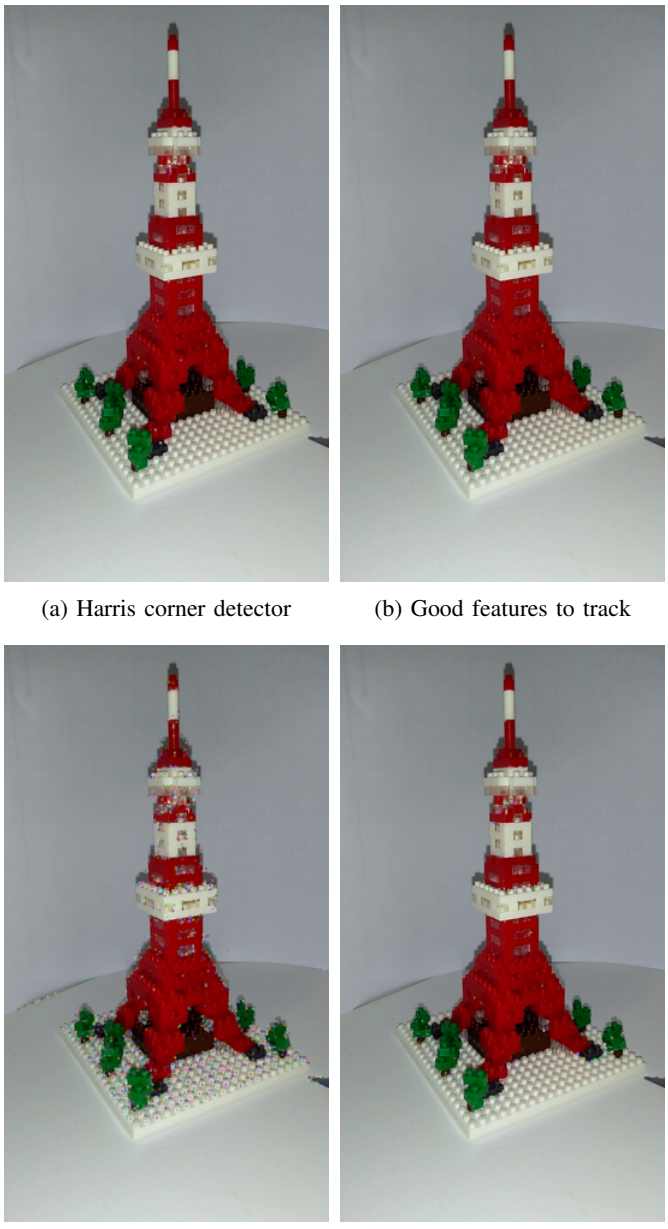
In Fig. 1, we can observe the response of each extractor of key points with respect to the first frame of our video. Figs. 1a, 1b and 1d show a very low performance, different from Fig 1c, because the video was recorded in a very controlled scenario (the less interest points out of the object as possible) and with frontal light, where the descriptor must be invariant to illumination, something that SIFT is known to do well.

III. Feature tracking

The Kanade-Lucas-Tomasi (KLT) algorithm [4] was implemented to track the detected keypoints between the gray-scale frames. As suggested in the enunciate, the flow was calculated by taking the left and lower image derivatives with a delta of one pixel, as well as the time derivative with respect to the next frame. Listing 1 shows our implementation of the algorithm.

Since we needed that all the detected keypoints remain in all the frames of the video, we used two criteria to stop tracking those points. The first criterion to discard was whenever the window constructed around one point of the first frame is not fully contained in the image, and the second criterion is whenever the calculated flow makes the point to go out the image. For every pair of frames, a flag vector, indicating which of the points were discarded, is returned so, when building the motion maps U and V, only those points that could be tracked for all the frames were considered.

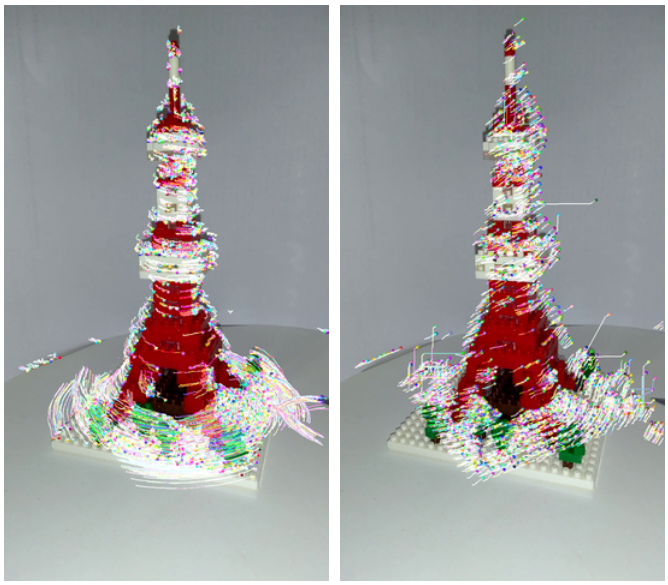
Fig. 2 shows the comparison of the flows obtained by the OpenCV implementation and ours. The images the limitations of our implementation: the calculated flow is way smaller than the actual motion so it keeps behind the actual keypoints of the next frame. It can be seen that the flow tries to follow the motion, but it stays behind so, for the next frame, another points are considering. Checking the motion maps yielded,



(a) Harris corner detector (b) Good features to track

(c) SIFT (d) ORB

Fig. 1: Key points extracted in the first frame of the video using different point extractors.



(a) OpenCV implementation. (b) Our implementation.

Fig. 2: Optical flows of the video taken by the OpenCV implementation and by ours. The image corresponds to the last frame of the sequence.

the mean delta of the flow was normally less than one pixel, and natural motion uses to be higher than that.

```
def get_flow(frame0, frame1, p0, neigh = 15):

  p1 = np.zeros(p0.shape, dtype=np.float32)
  st = np.ones(p0.shape[:1], dtype=np.int8)
  dt = frame1 - frame0
  dx = np.diff(frame0, axis = 1)
  dy = np.diff(frame0, axis = 0)
  w_ran = range(-(neigh // 2), (neigh // 2) + (neigh
     % 2) )

  for p in range(p0.shape[0]):
    x, y = p0[p].astype(np.int16)
    if  x+w_ran[0] < 0 or y+w_ran[0] < 0 \
      or x+w_ran[-1] >= frame1.shape[1]-1 \
      or y+w_ran[-1] >= frame1.shape[0]-1:
      st[p] = 0
      p1[p] = [-1.,-1.]
      continue
    A = np.zeros((neigh ** 2, 2), dtype=np.float32)
    b = np.zeros((neigh ** 2), dtype=np.float32)
    k = 0
    for wx in w_ran:
      for wy in w_ran:
        A[k,0] = dx[y + wy, x + wx]
        A[k,1] = dy[y + wy, x + wx]
        b[k]   = dt[y + wy, x + wx]
        k += 1
    p1[p] = p0[p] + lstsq(A, -b)[0]
    st[p] = 0 <= p1[p,0] < frame1.shape[1] - 1 \
        and 0 <= p1[p,1] < frame1.shape[0] - 1
  return p1, st
```

Listing 1: Final transformation

IV. Structure for motion

This algorithm consists in performing an affine or projective transformation on a set of key points that were tracked over a number of frames. To achieve this reconstruction we use a technique called structure for motion, introduced by Tomasi and Kanade [5], which aims to factorize the motion map (W) obtained in the flow-calculation step

$$W = MS \tag{1}$$

M is a matrix that represents the affine/projective transformations and S, the coordinates of the key points in a three-dimensional plane. In order to be able to obtain these matrices it is necessary to create an approximation from the movement of the observed points, this approximation is achieved with the steps shown below.

A. Creating the motion map

The motion map $\tilde{W}$ is created from the keypoints obtained during the video tracking process (See Section III). $\tilde{W}$ is represented by a matrix of size $2F \times P$, where F is the number of frames of the video and P is the number of points that are present in all frames.

$$\tilde{W} = \begin{bmatrix} X_{11} & \dots & X_{1P} \\ \vdots & & \vdots \\ X_{F1} & \dots & X_{FP} \\ Y_{11} & \dots & Y_{1P} \\ \vdots & & \vdots \\ Y_{F1} & \dots & Y_{FP} \end{bmatrix} \tag{2}$$

The next step is to subtract for each of point of each frame the mean or centroid of the points of each frame using the Eq. 3

$$\begin{aligned} \tilde{X}_{fp} &= X_{fp} - \frac{1}{P}\sum_{P=1}^{P} x_p \\ \tilde{Y}_{fp} &= Y_{fp} - \frac{1}{P}\sum_{P=1}^{P} y_p \end{aligned} \tag{3}$$

B. Compute the singular-value descomposition

In order to approximate W, it is necessary to perform a decomposition of the matrix $\tilde{W}$ in such a way that

$$\tilde{W} = \hat{M}\hat{S}. \tag{4}$$

The authors of the method propose to do the factorization using Singular Value Decomposition (SVD), where

$$\tilde{W} = U\Sigma V^T, \tag{5}$$

so that it will give three matrices in response, U and V^T contain the eigenvectors, and the diagonal matrix Σ contains the eigenvalues of the original matrix $\tilde{W}$.

The authors show that the $\tilde{W}$ matrix has a ranking of three, so only the first three eigenvalues provide information to the

system, and the others are only very minor information or noise. This matrix ranking demonstration allows a reduction of the dimensionality of the matrices, so that only the first three columns of the U matrix are taken, the first three rows of the matrix V^T and the three first values of the diagonal of Σ. The resulting matrices have size $2F \times 3$, $3 \times P$ and 3×3.

Then using Eq. 4, we can deduce that $\hat{M}$ and $\hat{S}$ can be obtained:

$$\begin{aligned} \hat{M} &= U \\ \hat{S} &= \Sigma V^T \end{aligned} \tag{6}$$

C. Calculating the real values of M and S

In order to calculate the real values of M and S it is assumed that the matrices obtained in the previous step are being multiplied by a matrix Q,

$$\begin{aligned} M &= \hat{M}Q \\ S &= Q^{-1}\hat{S} \end{aligned} \tag{7}$$

now the problem is reduced to finding the Q matrix. Using the metric constraint proposed by the authors, it is known that the vectors of the first half of the matrix $\hat{M}$ are orthogonal to the vectors of the second half, so that the following system of equations can be constructed

$$\begin{aligned} \hat{i}_f^T L \hat{i}_f &= 1 \\ \hat{j}_f^T L \hat{j}_f &= 1 \\ \hat{i}_f^T L \hat{j}_f &= 0 \end{aligned} \tag{8}$$

where $\hat{i}_f$ are the vectors of the first half of the matrix $\hat{M}$, $\hat{j}_f$ are the vectors of the second half and finally

$$L = QQ^T, \tag{9}$$

being L a symmetric matrix of size 3×3.

To solve this system we use the technique of least squares, obtaining as answer the matrix L, later this matrix is factorized by means of the Cholesky method, which gives us as result the matrix Q. As a final step, using the Eq. 7, we can obtain the real values of M and S.

The implementation of the structure for motion algorithm can be seen in Listing. 2

D. Obtaining the camera positions

After obtaining the real matrices M and S, it is necessary to obtain the position of the cameras in the real world, for this we will use the matrix M, which is the matrix of the transformations, this is a matrix of size $2F \times 3$,

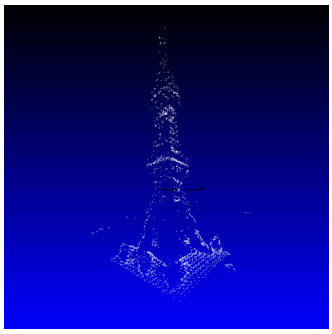
$$M = \begin{bmatrix} i_1^T \\ \vdots \\ i_F^T \\ j_1^T \\ \vdots \\ j_F^T \end{bmatrix} \tag{10}$$

```python
def sfm(W, RANK=4):
    # Normalization
    a_f = np.mean(W, axis=1).reshape(W.shape[0], 1)
    W_aprox = np.matrix(W - a_f)

    # Only in homogeneous
    if RANK==4:
        O = np.ones((W.shape[0]//2, W.shape[1]))
        W_aprox = np.concatenate((W_aprox,O), axis
    =0)

    # SVD
    U, s, V = np.linalg.svd(W_aprox)

    U_ = U[:,:RANK]
    s_ = np.matrix(np.diag(s[:RANK]))
    V_ = V[:RANK,:]

    # Get hat variables
    M_hat = U_
    S_hat = s_ * V_

    # Number of frames
    F = M_hat.shape[0]//2

    # Get linear system
    G = get_g(M_hat, RANK)
    c = get_c(F)

    # Solve linear system
    l = lstsq(G, c)[0]

    # Create square matrix
    L = np.matrix(squareform_diagfill(l))

    # Cholesky
    A = cholesky(L)

    # Update
    M = M_hat * A
    S = inv(A) * S_hat

    return M, S
```

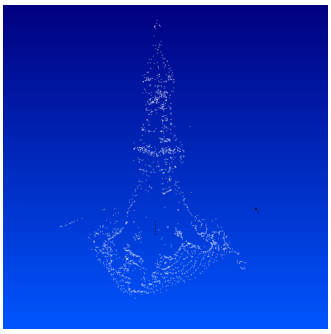
Listing 2: Structure for motion algorithm

```python
def get_camera_centers(M, RANK=3):
    F = M.shape[0] // 3
    C = np.matrix(np.ones((F, RANK)))

    for f in range(F):
        P = M[[f, f+F, f+2*F]]
        cr = -np.cross(M[f,:3],M[f + F,:3])
        C[f,:3] = cr / norm(cr)

    return C
```

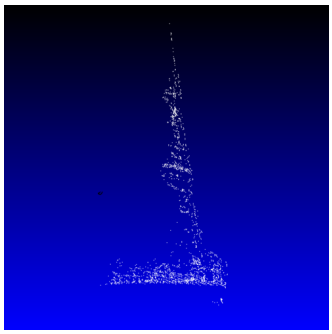
Listing 3: Algorithm for obtaining the position of the cameras



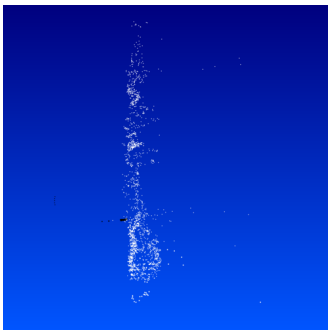
(a) Front view using OpenCV optical flow



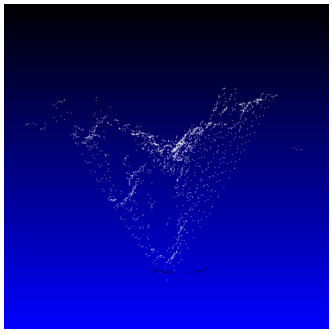
(b) Front view using our optical flow



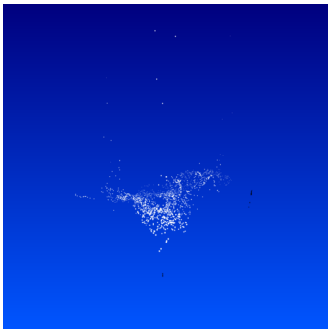
(c) Lateral view using OpenCV optical flow



(d) Lateral view using our optical flow



(e) Top view using OpenCV optical flow



(f) Top view using our optical flow

Fig. 3: Results of applying sfm, the key points of the structure in white and the positions of the chambers in black.

where i_f corresponds to the x-axis of the image plane and the j_f, to the y-axis. Thus we can obtain a vector k_f, which is orthogonal to both axis,

$$k_f = i_f \times j_f \tag{11}$$

this vector k_f is in the direction of the center of the structure created by the points of the matrix S. In Listing 3 we can see the implemented function to obtain the centers of the cameras.

In Fig. 3 we can see the results obtained when plotting the points and cameras obtained when applying the structure for motion pipeline to the video using Opencv's optical flow algorithm and ours.

The Fig. 3b, 3d and 3f show the results obtained using our implementation of optical flow. We can observe that even though our algorithm is not as good as the implementation of OpenCV, but still it is possible to get the shape of the structure. The problem with this method is that the side view and the top view does not have good shape, because the error of the algorithm of optical flow makes the predicted points are not so good and thus can not do such an effective reconstruction. Finally, in black, we can see the positions of the cameras, which do not have much relation with the real movement of the video, therefore we associate this error to our implementation of optical flow which adds certain noise, preventing us of yielding a reliable 3-dimensional representation.

Figs. 3a, 3c and 3e show the results obtained using the OpenCV version of optical flow. We can see in the frontal, lateral and superior view a good reconstruction of the model in 3 dimensions, where it is only possible to see a corner of the tower, which was the one shown for the video of entrance. We also noticed that the depth of the 3-D scene is aligned with the real object. The black points represent the cameras position. As it can be seen in the three figures, the movement of the cameras is circular with respect to the center of the tower, which is exactly the same movement that the camera did with respect to the tower to capture the video.

V. Discussion

A pipeline for structure from motion using optical flow was implemented for this project. It could be seen that optical flow provides information that can be used reliably to infer the 3D structure from objects, without the need of extraction of features to do the matching of the points between frames. We implemented the whole pipeline but, to test the correctness of the structure from motion algorithm, we run with both, our implementation of optical flow and the OpenCv's one. Our optical flow, due to its low capacity to detect large movements, yielded a somehow plane representation of the 3D points and an inaccurate position of the camera. This could be because of the error it attains, at taking points that are not the relevant ones for the next frame (and that keeps propagating to the next frames), adding lots of noise to the model.

Our implementation of structure from motion showed to be successful, although there were problems at implementing the method with homogeneous coordinates, due to a high number of linearly dependent rows in the matrix that represents the motion maps, making unfeasible the factorization. The calculus of the camera centers had to be done from non-homogeneous coordinates, by taking the Cartesian product of the orientations of the camera, yielding the vectors of the projections of the image in each frame, which show reliably the position of the "cameras".

Regarding the relation of the number of points with respect to the number of frames, the rule of thumb suggested in the paper where the double of the number of frames should remain lower than the number of points was easily accomplished, since a normal video of a few seconds does not have much more frames than the number of interest points that, for example, SIFT detects. It is clear that there is a trade-off,

because too few points with even fewer frames will yield a poor reconstruction. This was shown when using the other descriptors different from SIFT, which obtained few points from which nothing reliable can be reconstructed.

REFERENCES

[1] D. G. Lowe, "Object recognition from local scale-invariant features," in *Proceedings of the International Conference on Computer Vision-Volume 2 - Volume 2*, ser. ICCV '99. Washington, DC, USA: IEEE Computer Society, 1999, pp. 1150–. [Online]. Available: http://dl.acm.org/citation.cfm?id=850924.851523

[2] J. Shi and C. Tomasi, "Good features to track," Ithaca, NY, USA, Tech. Rep., 1993.

[3] E. Rublee, V. Rabaud, K. Konolige, and G. Bradski, "Orb: An efficient alternative to sift or surf," in *2011 International Conference on Computer Vision*, Nov 2011, pp. 2564–2571.

[4] C. Tomasi and T. Kanade, "Detection and tracking of point features," International Journal of Computer Vision, Tech. Rep., 1991.

[5] ——, "Shape and motion from image streams under orthography: A factorization method," *Int. J. Comput. Vision*, vol. 9, no. 2, pp. 137–154, Nov. 1992. [Online]. Available: http://dx.doi.org/10.1007/BF00129684