Video Stitching

Author:
Naravich Chutisilp (341752)
Yuxin Wang (338745)
Zhendong Li (328874)

Professor:
Sabine Süsstrunk

Supervisor:
Deblina Bhattacharjee
Ehsan Pajouheshgar

June 3, 2022

Abstract. In this project, we first stitch the images using SIFT detector to extract the keypoints from the images. We then use the nearest neighbor matching to match the keypoints. Finally, we use the RANSAC to find the homography matrix, which is used to transform one image and stitch it to the other. We expand this image stitching to stitch video frames together. However, we found that our implementation is not efficient enough to stitch frames from a video that has many number of images. Thus, we resort to the well implemented API from OpenCV which leverage the running time by using CPython. We create stitching video by mapping each frame to its approximated position in the panorama. We then use the background OpenCV's subtractor to provide background and foreground masks. That we can use to create a background and foreground movies. Lastly, we experiment with SuperPoint and SuperGlue to extract the keypoints and match them, instead of SIFT detector and nearest neighbour matching that we use previous. We find that the result of homography matrix from the result matches are similar. Hence we conclude that SIFT detector performs well enough in our case.

Keywords: Video stitching, interesting points, SIFT, homography transformation, background subtraction, SuperPoint, SuperGlue

1 Introduction

Video stitching is useful in the video surveillance, entertainment, virtual reality and other fields. There are several aspects in regarding this task. We are focusing on the video stitching from one camera. This task is similar to the image stitching from one camera which is widely used in the smart phones. This method lowers the cost of the video stitching process, which otherwise, in many cases utilizes a specialized tool. Once the video stitching is done. Background and foreground extraction can be made for extracting the features from the video.

Video stitching in this manner is similar to creating a panorama image. However, instead of placing images altogether onto one big canvas, each frame are placed onto the canvas one at a time. Consequently, a stitched video is obtained from combining these new frames together into one video. Therefore, the solution to making a stitched video is to use the traditional image stitching algorithm. This includes finding features and matching them together in order to acquire the transformation matrix for each frame. By doing so, one could estimate the camera parameters which can be used to place each frame onto the right position on the panorama canvas. To further improve the stitching result, image blending and exposure compensation must be done. Nonetheless, SIFT with nearest neighbor matcher is not good for all cases. Compared to the machine learning based algorithm, nearest neighbor matcher performance decreases when the angle between two images increases [12]. Furthermore, in pose estimation, machine learning based algorithm extracting the keypoint features from the images performs better than using SIFT to do the same task [12]. In order to reduce computational time and memory requirement, one may reduce the number of frame extracted from a video. By doing so, the angle between two consecutive frames increases. Thus, machine learning based algorithm may be useful in feature extraction and matching.

SuperGlue is a combination of machine learning and a traditional algorithm. It receives keypoints and descriptors as its input, and pass them through attentional graph neural network. This network encodes the keypoints and their visual descriptors into two vectors for each image. Matching score S_{ij} for keypoint ith from the first image, and jth from the second image is calculated by the inner product of the vectors. In order to build a matching matrix, Sinkhorn algorithm is performed on the matching matrix S to normalize each entry. As a result, each entry implies the confidence level. Each row of S is summed to 1, as well as each column [12]. This is different from nearest neighbor matching by which SuperGlue also utilizes the spatial information. However, its researcher only provides the pretrained model API. Therefore, we are not able to finetune the model for our use case.

We build a video stitching program, by first, using the traditional algorithms such as SIFT, and nearest neighbor matching. We use RANSAC to estimate the homography matrices, and all traditional algorithms to stitch the video. We

also extract the masks which indicate where each frame is on the stitched video canvas. We use the traditional background subtraction on the stitched video, and improve the result by applying the masks on the result. Background subtraction returns the masks indicating which pixels belong to the background. Thus, inverting this mask, we could also extract the foreground masks. Finally, we create a program that creates a background and foreground video from the stitched video. In order to evaluate the improvement in incorporating the machine learning based algorithm for feature extraction, and feature matching, we perform both quantitative, and qualitative analysis on the resulting homography matrices from the traditional algorithm against the machine learning based algorithm, using the root mean square error, and handcrafted difference metric, relative difference.

We assume that the homography matrices difference metrics and qualitative analysis could indicate the overall performance of the video stitching process. In this report, we will present the traditional algorithms, including SIFT detector, RANSAC homography estimation, image stitching using the homography matrix, image blending. We will present our final video stitching and the settings that we use. We will describe the background subtraction for making background and foreground videos. We will discuss SuperPoint and SuperPoint that would replace SIFT detector, and nearest neighbor matching (NN). Finally, we will discuss quantitatively and qualitatively the performance of the traditional algorithm (SIFT + NN) and the machine learning based algorithm (SuperPoint + SuperGlue).

2 Literature

2.1 SIFT detector

Image matching could become challenging if the images vary in multiple aspects, for example, the scale, rotation, other parameters like illumination and noise. The Scale Invariant Feature Transform (SIFT) is an approach to solve the issue. When detecting key-points in an image, SIFT transforms the data points to a high-dimensional vector relative to its local region [6]. This transformation helps to generate a highly distinctive descriptor of each keypoint so that is invariant to image scale and rotation and also robust to other parameters. The steps to generate the SIFT features can be concluded as the followings:

1. Scale-space extrema detection: searching over all scales and image locations by using a difference-of-Gaussian function to identify potential interest points that are invariant to scale and orientation.
2. Keypoint localization: fitting a detailed model to determine location and scale at each candidate location where keypoints are selected based on stability.
3. Orientation assignment: assigning one or more orientations to each keypoint location based on local image gradient directions so that all future operations

will be performed on image data that has been transformed relative to the assigned orientation, scale, and location.
4. Keypoint descriptor: measuring the local image gradients at the selected scale in the region around each keypoint, and the gradients are transformed into a representation that allows for significant levels of local shape distortion and change in illumination.

2.2 Nearest Neighbour Matching

When doing image matching by using the SIFT features, a nearest neighbour matching can be used to compare the features between images [6]. The nearest neighbor is considered as the the keypoint who has the minimum Euclidean distance for its descriptor vector. It is not effective to use a global threshold to determine "nearest" distance as there might be too many bad matches and some rather discriminate features are much better than other bad ones. Instead, comparing the distance between the closest neighbor and the second closest neighbor is used in this algorithm. Moreover, the Best-Bin-First algorithm will be used to do a more efficient exhaustive search on the neighbors[6].

2.3 RANSAC

Random sample consensus (RANSAC), is an iterative algorithm to estimate the parameters of a model by sampling a small set of data points from the original dataset with outliers[2]. The RANSAC algorithm is meaningful in finding the optimal solution while removing the outliers at the mean time. And during each iteration, it aims to use as small dataset as possible to estimate the model parameters.

RANSAC can be summarized as the following steps:

1. Randomly selecting s samples from the original dataset.
2. Fitting the model to the random sample.
3. Counting the number of inliers that fit the model within an error ϵ.
4. Repeating steps 1-3 for N times

2.4 Background subtraction algorithms

The first background subtraction we want to introduce is the Vibe[1]. The paper has three main contributions. First, it can be fast, as it only needs to use the first image to initialize. Second, it selects 20 pixels randomly from the first image to present the background model. Third, the random process can extend to the neighborhood pixels. Thus, include the spatial information in the model selection.

The second one is to use the adaptive Gaussian Mixture Model[14]. This algorithm attempts to describe the probability distribution of each pixel using the GMM model. If the value of the new pixel can be well described by the

probability density function, it belongs to the background. Otherwise, it is the foreground. In a word, this method mainly realizes the adaptation of the number of Gaussian models in GMM.

2.5 SuperPoint

Intersting points are ill-defined so it's a hard problem in computer vision to detect reliable and repeatable interesting points. However, SuperPoint [3] is a self-supervised framework to train the interesting points detection and descriptors as a promising input to SuperGlue. It performs well by creating a large dataset of pseudo-ground truth interest point locations in real images from a base detector and then it further trains the detector with those pseudo-ground truth points as labels, supervised by the interest point detector itself. The base detector, as the result from training a fully-convolutional neural network at first place, is named MagicPoint. So SuperPoint training can be summarized as: (1) Interest Point Pre-Training; (2) Interest Point Self-Labeling; (3) Joint Training.

2.6 SuperGlue

SuperGlue is a graph neural network [12] that receives keypoints and descriptors from two images as an input, and returns the matching matrix where an entry of the ith row and jth column indicates the confidence level that the ith keypoint of the first image and jth keypoint of the second image match. If a pair of keypoints does not match, the confidence value is 0. If a pair of keypoints does match, the confidence value can go up to 1.

3 Implementation

3.1 SIFT detector

We first try to re-implement SIFT detector following Lowe's [6]. To validate our program, we input our SIFT detector into the keypoint matching of the OpenCV. We implement our SIFT detector as follows.

1. Create Gaussian Pyramid Images.
2. Create a Difference of Gaussian Pyramid Images (DoG).
3. For each octave and the scale levels:
   (a) We compute the gradient of magnitude, and orientation of the gradient of that level.
   (b) We iterate through each pixel of that level, and find whether it is the extrema following the criteria of the paper. If that pixel is the extrema, we compute the descriptor.

However, our program runs inefficiently due to many iterations. Our differential and hessian calculations are not accurate. And we miss to properly do the interpolation. In addition, OpenCV has their own parameter and convention such

as using an integer to store all x, y, scale, and octave by doing bit-shifting. This leads to incompatibility which makes our program not working on the OpenCV. Eventually, we review the Python implementation from Russ Islam [5] to understand what we have missed, and use his program as our educational model of how the SIFT works inside the OpenCV.

We also learn that OpenCV has their own coding environment such that a naive program will not work. To work with OpenCV, we either have to use their output from their APIs, or do many trials and errors in order to learn how Python binds to their C++ class. Since OpenCV does not have a good document about their classes inside each API, this process is time consuming. Therefore, we would just use OpenCV API for our project.

3.2 RANSAC and Homography

Assume we already get the matching pairs based on the SIFT descriptors and the NN matching. Now we want to compute the homography matrix based on the matchings for the later stitching of two images. If we call the points of one image (x_d, y_d) and the points from another image (x_s, y_s), then the equation of transforming (x_d, y_d) to (x_s, y_s) using the homography matrix H, which is the matrix we want to solve, is as follows [8]:

$$\begin{bmatrix} x_d \\ y_d \\ 1 \end{bmatrix} \equiv \begin{bmatrix} \tilde{\mathrm{x}}_d \\ \tilde{\mathrm{y}}_d \\ \tilde{\mathrm{z}}_d \end{bmatrix} = \begin{bmatrix} h_{11} & h_{12} & h_{13} \\ h_{21} & h_{22} & h_{23} \\ h_{31} & h_{32} & h_{33} \end{bmatrix} \begin{bmatrix} \tilde{\mathrm{x}}_s \\ \tilde{\mathrm{y}}_s \\ 1 \end{bmatrix} \tag{1}$$

Then after some matrix computation, for one pair i of the matching points, we have:

$$\begin{bmatrix} x_s^{(i)} & y_s^{(i)} & 1 & 0 & 0 & 0 & -x_d^{(i)}x_s^{(i)} & -x_d^{(i)}y_s^{(i)} & -x_d^{(i)} \\ 0 & 0 & 0 & x_s^{(i)} & y_s^{(i)} & 1 & -y_d^{(i)}x_s^{(i)} & -y_d^{(i)}y_s^{(i)} & -y_d^{(i)} \end{bmatrix} \begin{bmatrix} h_{11} \\ h_{12} \\ h_{13} \\ h_{21} \\ h_{22} \\ h_{23} \\ h_{31} \\ h_{32} \\ h_{33} \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \end{bmatrix} \tag{2}$$

So for all matching pairs in size N, the first multiplier will grow into a large matrix in shape (2N x 9). In our implementation, we construct such a matrix to be the left hand side of a linear system and use the numpy library to solve it.

However, we could not just use the matching pairs from SIFT to solve the matrix directly. Because there is no guarantee that all the matchings are all good matchings. For example, in Figure 1, the matchings are good. But in Figure 2, there are two matchings which though considered as similar by SIFT, are not correct and we call them outliers.

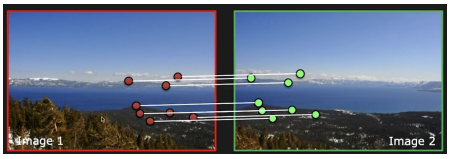

Fig. 1: Matchings with inliers

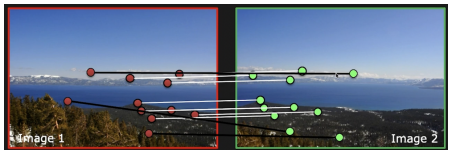

Fig. 2: Matchings with outliers (in black lines) [8]

Therefore, we would like to filter out those outliers and build a more precise model using only the largest number of inliers. The Ransac algorithm jumps in to help with this. Referred back to the H we talked before, we also need to fix the scale of it: $\sqrt{\sum(h_{ij})^2} = 1$ so that we wouldn't get a naive solution as all zeros. Hence the degree of freedom of H is actually 8. That means, only 4 point pairs are enough to get a unique solution. Then we just base on the steps as mentioned in the Literature section to implement the Ransac for our model.

1. Randomly select 4 matchings from the original dataset.
2. Calculate an H using the 4 random sample.
3. Use the H from step2 to transform all the data points in img1 and count the number M of inliers within an error ϵ. We compute the pixel distance between the transformed points and the points in img2 to compare with ϵ
4. Repeat steps 1-3 for N times
5. Choose the H that has the maximum M and also save this maximum set of inliers.

However, there is some issue with the H computed following the equation (2), so we refer to this implementation [4] to revise our code.

3.3 Image Stitching

Image stitching is the building block of the video stitching. We use OpenCV's SIFT detector and nearest neighbor keypoint matching with RANSAC to find the homography matrix. We use this homography matrix to transform one image to match the matched keypoints of another. Next, we stitch the these two images together.

Furthermore, we extend this simple algorithm to make a panorama image by the following algorithm.

1. Extract frames from a video. To help reduce the memory, we specify the parameter that we use to extract every other frame of that number.
2. Starting from the middle frame, we stitch the right image to it.
3. Then, we stitch the left image to this result. We call this our new middle image, and we repeat the process until all the frames are stitched.

Nevertheless, when two images are stitched together using this simple algorithm, the result has a lot of black blank part. The stitched image size increases

drastically when few images are stitched together. As a result, the memory of the computer could not handle to stitch the all frames from a video by this particular algorithm (see Figure 3).

Additionally, we review the image stitching implemented by OpenCV. OpenCV solves this problem by calculating the camera parameters from the matched image pairs. OpenCV also refines the camera parameters using its bundle adjuster. As a result, the image warper in OpenCV leverages these camera parameters. Consequently, the warped images are more accurate. The detailed description about OpenCV image stitching will be discussed in the Section 3.5.

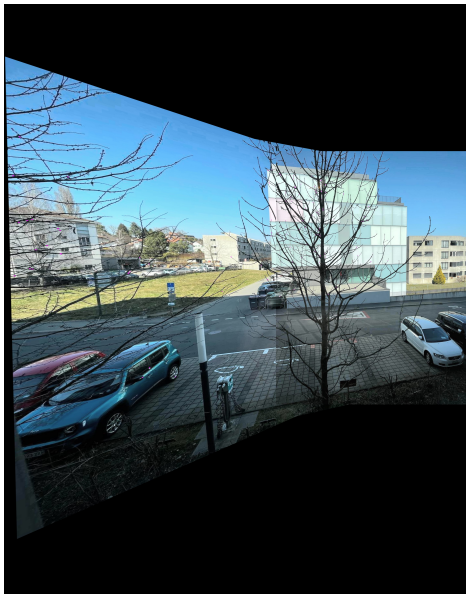

Fig. 3: Result stitched image from the simple stitching algorithm - with only two images stitched together, the image has more blank area which for more images, the memory grows significantly

3.4 Blending

To seamlessly stitch the two images together, we tried Laplacian Blending. One obvious thing is that when we downsample and then upsample the image, the seams of the image are smoother due to interpolation. But we also lose image details, so we need to add the original image details to the upsampled image. This is the core point of the Laplacian Blending algorithm. The specific details are as follows:

1. Create the mask; the mask's boundary is the seam of two images. Then use the mask to combine two images into one image.
2. Do gaussian blurring before they are downsampled.
3. Downsample the image to a certain scale step by step (until it is enlarged back to the original size after splicing, the border becomes smooth). At the same time, get the Laplacian pyramid.
4. Upsample the image and add the Laplacian pyramid level by level.
5. Go back to step 4 and loop until it returns to the original size.

The whole process is as Figure 4.

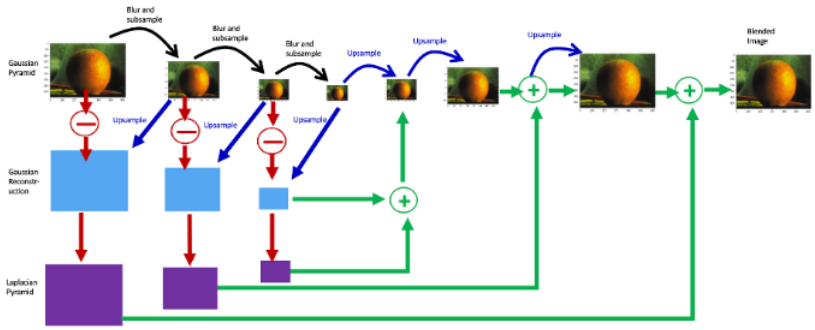

Fig. 4: Laplacian Blending ref.[13]

Fig. 5: original image

Fig. 6: After Laplacian Blending image

We have wrote the code about laplacian blending and tried to use Laplacian Blending to process our images Figure 5 and Figure 6. We can clearly see that the shadows at the seam of the two images are blended out. But at the same time, the high frequency information of the picture is lost.

3.5 OpenCV Panorama API for video stitching

OpenCV provides a very comprehensive image stitching API that we could modify and adjust according to our project requirements under the OpenCV's environments [10].

For our settings, we use SIFT detector for finding keypoints. Nearest neighbor and RANSAC for finding matches for homography matrix. We set bundle adjuster to ray [9]. This adjustment helps refine the camera parameters by optimizing the total distance of the rays passing through a camera center. The blending is set to *multiband* [11]. This algorithm differs from the Laplacian blending. It computes the pyramid of images by applying several bandpass filters to the two images. This yields image pairs of different frequency components. This is because the image splining using a weighted summation works best when two images have no more than one octave difference in their frequency components. Each sub-band is then splined by weighted summation, and the splined images of all sub-bands are combined together [11]. To find the seam between a pair of images, we set the seam finder to dp_color. From our knowledge, it is the OpenCV implementation of seam finding using dynamic programming to optimize the custom cost function. We use spherical warper to map the images onto the unit sphere centered at the origin. The exposure compensator is set to GAIN_BLOCK which adjusts the image block to remove exposure related artifacts [7].

The OpenCV's image stitching algorithm is as follows.

1. Extract frames from a video. To help reduce the memory, we specify the parameter that we use to extract every other frame of that number. This part is our implementation as OpenCV only needs the directory of the images to be stitched.
2. For each image, compute features using the SIFT detector.
3. From the features, match image pairs using nearest neighbour matching. This process matches all the possible pair of image and keep only the pairs with highest confidence.
4. From the features and matches, we estimate the camera parameters for each image.
5. From the features and matches, refine the camera parameters, using Ray bundle adjuster.
6. Do the wave correction in the horizontal axis in order to make the horizontal line in the image horizontal.
7. Using the camera parameters, warp the image with a spherical warper. This process provides both warped images and their top left corners which are the positions in the panorama.

8. Feed all the warped images to the exposure compensator, block gain. This prepares the compensator for later use.
9. Put all the warped images to the seam finder. This returns a mask of the seam for each image. This will be used later with the blender.
10. Compensate the exposure for each image.
11. Pass the image to the blender along with the seam mask that is being dilated with 3x3 rectangle kernel.
12. Along with the seam mask, the image is put into the blender which blends the image, and also do the images stitching onto the panorama canvas.
13. Using the corners, the timelapser puts the image onto the correct position of the panorama canvas. In this case, we don't need the blender since we only put one image onto the canvas for each frame. Since each frame is a warped image on the black panorama canvas. We need to modify some code in order to make our background and foreground extraction work better.
14. We add the code to create a GIF image from the resulting timelapse frames. Thus, this gives us a stitched videos in a GIF format. The result frames from the video are save into a specified directory. We use this frames of the stitched video to do the background and foreground extraction. This is will discussed in the following section.

Some frames from stitched videos are shown in Figure 9, 10, and 13. Each frame are correctly placed on the video canvas to create a stitched video.

3.6 Background Subtraction for foreground and background video

We try the both Vibe and the adaptive Gaussian Mixture Model mentioned in the Section 2.4. We qualitatively investigate these two algorithms. From our test, Vibe [14] performs worse than GMM. This may be due to the fact that Vibe initializes with random color pixels from the image to represent the background. According to the paper, this might cause the "ghost" effect where the foreground pixels are classified as background, and persist to be so. This can be seen in the Figure 7, compared against GMM's Figure 8. We use the opening operation with the 5×5 kernel for both algorithms. So that some small noise can be filtered. We multiply the detected video with the mask video; it has the same size as the detected video. We use this mask to avoid detecting the new boundary, due to the movement of the camera from left to right, as the moving object. You can see that the image on the left edge is clean after applying the mask.

In our experiments, the GMM algorithm is integrated into OpenCV and the Vibe algorithm we use is not a library algorithm of OpenCV. However, Vibe is theoretically faster than GMM. But the actual running effect of the GMM algorithm is better than the Vibe algorithm in terms of speed and effect. Hence, we choose the GMM algorithm in the end.

Some frames from background/foreground extraction videos are shown in Figure 11, 12, 14 and 15.

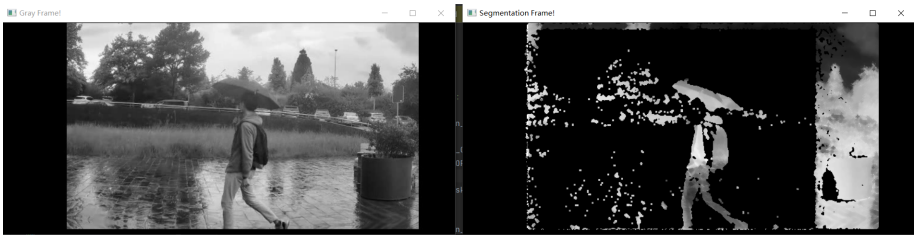

Fig. 7: the result by using vibe algorithm

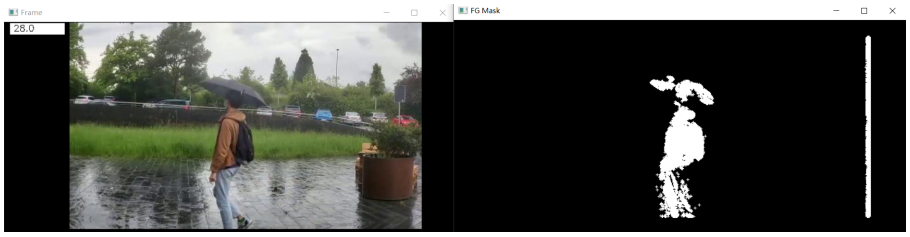

Fig. 8: the result by using GMM algorithm

Fig. 9: Frame 4 of the stitched video car

Fig. 10: Frame 108 of the stitched video car

Fig. 11: a background frame of the stitched video car

Fig. 12: a foreground frame of the stitched video car

Fig. 13: One frame of the stitched video RLC

Fig. 14: A background frame of the stitched video RLC

Fig. 15: A foreground frame of the stitched video RLC

3.7 SuperPoint and SuperGlue

Just as we mentioned in the Section 2.5, SuperPoint is a self-supervised deep neural network that receives an image as an input and returns keypoints and their descriptors as an output. And this output is passed to SuperGlue as input. This machine learning based matching algorithm can match two images that are captured in a highly different perspective. Thus, it could potentially replace our traditional algorithm in finding keypoints and matches, and improve the overall stitching result.

To experiment, we incorporate SuperPoint and SuperGlue into our image stitching implementation in the Section 3.3. This is done by converting the keypoints and descriptors from the SuperPoint, and the matching matrix from the SuperGlue into two Numpy array of $(N, 1, 3)$ where N is the number of matches. Specifically, we add the third dimension to the 2D keypoints from the SuperPoints, which is always 1, implying that these keypoints are in the 1st octave. The keypoints of the first and the second image are paired according to the match pairs from the SuperGlue, which we filter by the confidence values and the default threshold provided by SuperGlue. These Numpy arrays are compatible for OpenCV's findHomography API. Finally, we compare the homography matrices from this API using the the keypoints and matching pairs from SIFT detector and nearest neighbor matcher with RANSAC, and those matrices from the SuperPoint and SuperGlue.

3.8 Results

Quantitative Analysis
We compute the mean average of the element-wise root mean square errors (RMSE) and their variances of all the frames from videos captured in different environments. In order to reflect the actual difference, we also introduce the relative differences (RDiff) in the Equation 3 and their variances. To illustrate, 200 and 202 are very much different in RMSE. However, they are fairly similar in RDiff.

$$\frac{|H_1 - H_2|}{\sum |H2|} \tag{3}$$

The videos that we captured include both indoor, and outdoor, with no moving objects to several moving objects videos. The video *lake_duck* was captured at the back of the campus, beside the lac leman with the ducks swimming across the screen. The *lake* video was captured at the same place without the ducks, but the reflections on the water surface were there. The *car* video was captured beside the road behind the Rolex Learning Center with the cars passing by. The video *les* was captured at the Les Etudiantines where there were no people. The video *people* was captured on the campus where people were walking by. The video *square* was captured square near the Rolex Learning Center, where there were no moving object. *RLC* was captured inside the Rolex Learning Center where students were sitting. *RLC*2 was captured inside the Rolex Learning Center where there were less people. *RLC*3 was captured inside the Rolex Learning Center where students were walking pass the camera. Lastly, *RLC*360 was captured inside the Rolex Learning Center which we captured a 360 degree view.

The results are summarized in the Figure 16. Homography matrices calculated from the SIFT detector and nearest neighbor matching, and those from SuperPoint and SuperGlue are similar in the mean of RMSE's. Their variances are also low. Only the homography matrices from the lake video that are different in RMSE. However, RDiff shows that these differences are due to the high values in entries of the homography matrix, but relatively these matrices are similar. These differences are lower for the videos captured by the drone, which rotates the camera at a constant angular velocity (see Figure 17).

Qualitative Analysis
To see the result qualitatively, we run the same algorithm to stitch two images of the 1st frame and the 30th, 60th, or 80th frame. We predicted the middle angle image by cropping the middle part of the stitched image. This could be done because the videos that we use were captured by drone which moves at the constant angular velocity. For the middle angle prediction of the two images that are more than 80 frames apart, we found that both traditional and machine learning based algorithms could not provide enough matching pairs for OpenCV's findHomography API.

The qualitative results comply with the quantitative result. Since the homography matrix difference in terms of RMSE and RDiff are low, the stitched image are similar. Thus, the middle angle predictions from both algorithms are

	RMSE	Variance RMSE	RDiff	Variance Rdiff
lake_duck	0.488	0.331	0.047	0.003
lake	6.126	54.664	0.498	0.28
car	0.367	0.113	0.019	0
les	0.301	0.098	0.04	0.003
people	0.196	0.02	0.016	0
square	0.281	0.042	0.024	0
RLC	0.514	0.545	0.034	0.002
RLC2	0.841	0.663	0.103	0.007
RLC3	0.225	0.019	0.01	0
RLC360	0.626	0.842	0.039	0.003

Fig. 16: Metrics of difference in homography matrices calculated by keypoints, descriptors and match pairs from OpenCV's SIFT detector and nearest neighbor matching, and SuperPoint and SuperGlue

	RMSE	Variance RMSE	RDiff	Variance Rdiff
video1	0.140	0.004	0.008	0.000
video2	0.126	0.004	0.012	0.000
video3	0.136	0.004	0.021	0.000

Fig. 17: Metrics of difference in homography matrices calculated by keypoints, descriptors and match pairs from OpenCV's SIFT detector and nearest neighbor matching, and SuperPoint and SuperGlue

the similar. Middle angle prediction for the two images that are 30 frames apart are close to the real middle angle image, which is the 15th frame. Nonetheless, the middle angle predictions for the two images the are 60 frames, and 80 frames apart are not close to their middle angle images. This can be seen from the road of the middle angle image as a reference point. The result is summarized in the Figure 18.

4 Conclusions

We "stitch" the traditional algorithms together to get the video stitching program that is also able to return the mask information which improve the background and foreground extraction. We investigate further the potential machine learning based algorithms to replace SIFT and Nearest Neighbor. Nonetheless, according the quantitative and qualitative analysis, machine learning based algorithm for finding the keypoints, descriptors and matches do not improve the video stitching result. This is due to the nature of the video acquirement in our particular use case. The video is captured in a gradual manner where the camera slowly rotates from one angle to the final angle. This manner does not leverage the ability of the machine learning based algorithm that could match the keypoints from a pair of images that are in the significantly different point of views. Furthermore, SuperGlue's ability might works only the two images capturing the same object at the highly different point of views. This is not the case for our use case. We rotate the camera around where we stand, rather than around the object.

(a) 30 frames apart - traditional (b) 30 frames apart - machine learning based (c) 30 frames apart - target

(d) 60 frames apart - traditional (e) 60 frames apart - machine learning based (f) 60 frames apart - target

(g) 80 frames apart - traditional (h) 80 frames apart - machine learning based (i) 80 frames apart - target

Fig. 18: Middle angle prediction from traditional and machine learning based algorithm

References

1. Barnich, O., Van Droogenbroeck, M.: Vibe: a powerful random technique to estimate the background in video sequences. In: 2009 IEEE international conference on acoustics, speech and signal processing. pp. 945–948. IEEE (2009)
2. Derpanis, K.G.: Overview of the ransac algorithm. Image Rochester NY **4**(1), 2–3 (2010)
3. DeTone, D., Malisiewicz, T., Rabinovich, A.: Superpoint: Self-supervised interest point detection and description. In: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR) Workshops (June 2018). https://doi.org/https://arxiv.org/abs/1712.07629v4
4. Hughes, J.: Homography estimation in python (2015), https://github.com/hughesj919/HomographyEstimation/blob/master/Homography.py
5. Islam, R.: Pythonsift (2021), https://github.com/rmislam/PythonSIFT
6. Lowe, D.: Distinctive image features from scale-invariant keypoints (2004). https://doi.org/https://doi.org/10.1023/B:VISI.0000029664.99615.94
7. Matthew Uyttendaele, Ashley Eden, R.S.: Eliminating ghosting and exposure artifacts in image mosaics. pp. 509–516 (2001)
8. Nayar, S.: Computing homography (2021), https://youtu.be/l_qjO4cM74o
9. OpenCV: Bundle adjuster ray class reference (2022), https://hal.inria.fr/inria-00548290
10. OpenCV: High level stitching api (stitcher class) (2022). https://doi.org/https://docs.opencv.org/4.x/d8/d19/tutorial_stitcher.html
11. Peter J. Burt, E.H.A.: A multiresolution spline with application to image mosaics. ACM Transactions on Graphics **2**, 217–236 (1983). https://doi.org/https://doi.org/10.1145/245.247
12. Sarlin, P., DeTone, D., Malisiewicz, T., Rabinovich, A.: Superglue: Learning feature matching with graph neural networks. CoRR **abs/1911.11763** (2019), http://arxiv.org/abs/1911.11763
13. Zhao, M.: laplacian blending (2020), https://becominghuman.ai/image-blending-using-laplacian-pyramids-2f8e9982077f
14. Zivkovic, Z.: Improved adaptive gaussian mixture model for background subtraction. In: Proceedings of the 17th International Conference on Pattern Recognition, 2004. ICPR 2004. vol. 2, pp. 28–31. IEEE (2004)