Technical University of Munich

Group 17

Computer Vision Challenge

Satellite Image Change Detector

Team member: Zach Ren Lee, Shunxi Xu, Han Liang, Zhendong Li, Yuhang Zhou

Semester: So 2021

Submission date: July 10, 2021

1 Application Structure

In this section, a detailed insight to our application to detect landscape changes will be given. Fig. 1 depicts the structure of our application, which consists of two main blocks, the GUI and image processing block. In the following, the image processing block will be first discussed to give a better overview of how the changes are detected. Then, the GUI and features of the application will be introduced.

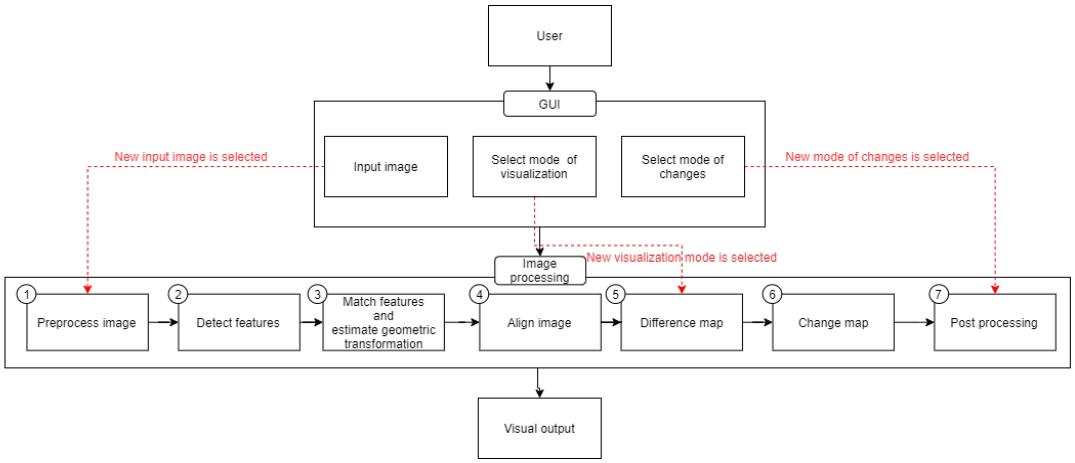


Figure 1: Application structure.

2 Image processing

The image processing block is the core block of the application which runs the change detection algorithm to detect changes between the input images. Inputs to this block are two images to be compared. Hence given a folder of multiple images, this block will be called iteratively and compare all the consecutive image pairs. At the end of this block, the change map between the two input images will be used to generate a visual output in the GUI.

2.1 Preprocess image

First and foremost, the 'GOOGLE' watermark, which is assumed to be at the bottom part of the image, is removed from both the input images. This can avoid matching the wrong correspondence point pairs of the input images. Then, the images are converted to gray scale images as a preparation to detect feature in the next step. The gray scale images are then normalized and standardized using the following equation

$$I = \frac{1}{\sigma(I)}(I - \bar{I}), \tag{1}$$

where I is the original gray scale image, $\bar{I}$ is the mean intensity, and $\sigma(I)$ is the intensity standard deviation.

2.2 Feature detection and match features

The KAZE features of the gray scale images are then extracted using the MATLAB built in detector. The KAZE detector has a slight advantage over SURF and SIFT in term of performance [2]. After that, the extracted KAZE features are matched as shown in Fig. 2.

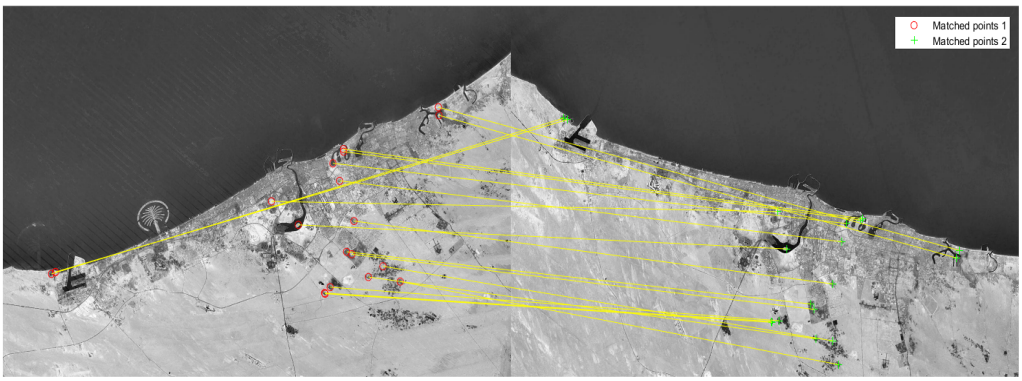


Figure 2: Correspondence point pairs of the input images.

2.3 Estimate geometric transformation

Given the correspondence point pairs, the geometric transformation from image 1 to image 2, which is described by the homography matrix H, can be estimated using the MATLAB built in function $estimateGeometricTransform2D()$. To remove outliers in the correspondence point pairs, this function uses the M-estimator sample consensus (MSAC) algorithm, which is an improved variant of the random sample consensus (RANSAC) algorithm [4]. At this point, it is assumed that the image is captured at a high altitude and therefore everything in the images lies on a plane. Thus, an extra affine constraint is added in the estimation and the function gives an affine homography matrix as the output. Note that an affine homography matrix has the following form

$$H = \begin{pmatrix} * & * & * \\ * & * & * \\ 0 & 0 & 1 \end{pmatrix} \tag{2}$$

2.4 Align image

Using the affine homography estimated in the previous step, the input images can now be aligned using the relation $\boldsymbol{x'} = H\boldsymbol{x}$, where $\boldsymbol{x'}$ is the homogeneous coordinate of image 1 in the frame of image 2 and $\boldsymbol{x}$ is the original homogeneous coordinate of image 1. An example is shown in Fig. 3

(a) Image 1 (b) Aligned image 1 (c) Image 2

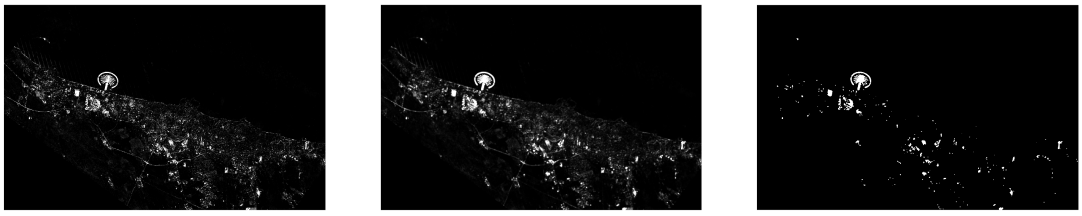
Figure 3: Image 1 is aligned with image 2 using the estimated homography H.

2.5 Difference map

To find the changes between the two input images, a difference map $D(y, x)$ is generated by computing the squared difference of the aligned images (Fig. 4a). The equation is as follows:

$$D(y, x) = (I_1'(y, x) - I_2(y, x))^2, \tag{3}$$

where $I_1'(y, x)$ is the gray scale of the aligned image 1 and $I_2(y, x)$ is the gray scale of image 2. The difference map is then filtered using a Gaussian kernel to remove some noises (Fig. 4b).



(a) (b) (c)

Figure 4: (a) Difference map, (b) low-pass filtered difference map and (c) change map.

2.6 Change map

To determine the changes, an unsupervised change detection technique, which uses the principal component analysis (PCA) and k-means clustering [3], is implemented. The difference map $D(y, x)$ of size $H \times W$ from the previous step is first padded with h zeros and partitioned into $h \times h$ (in our program $h = 2$) non-overlapping blocks. All the blocks are then stretched into columns vectors and stored inside the matrix $\boldsymbol{X}_d = [\boldsymbol{x}_{d1} \;\; \boldsymbol{x}_{d2} \;\; ... \;\; \boldsymbol{x}_{dM}]$, where $\boldsymbol{x}_{di}, 1 \leq i \leq M$ is the column vector representation of i-th block, and $M = \lfloor (H \times W)/(h \times h) \rfloor$. The matrix $\boldsymbol{X}_d$ is then subtracted by its average column vector $\boldsymbol{\Psi} = \frac{1}{M} \sum_{i=1}^{M} \boldsymbol{x}_{di}$ to obtain

$$\boldsymbol{\Delta} = \boldsymbol{X}_d - \boldsymbol{\Psi}, \tag{4}$$

where the subtraction is done column-wise. Next, an eigenvector space is created using PCA on the matrix $\boldsymbol{\Delta}$. The PCA technique is used to extract the most important information from $\boldsymbol{\Delta}$, reduce the dimension and thus simplify the description of the data [1]. The principal components can be found by computing the eigenvectors of covariance matrix $\boldsymbol{C} = \frac{1}{M} \sum_{i=1}^{M} \boldsymbol{\Delta}\boldsymbol{\Delta}^T$. The eigenvectors correspond to the largest S eigenvalues are then selected (in our program $S = 3$) to create a new feature vector space. The matrix $\boldsymbol{\Delta}$ is then projected to the new feature vector

$$\boldsymbol{V} = \boldsymbol{\Delta}^T [\boldsymbol{e_1} \;\; \boldsymbol{e_2} \;\; \boldsymbol{e_3}], \tag{5}$$

where $\boldsymbol{e_i}$ is the eigenvector corresponds to the largest 3 eigenvalues.

Subsequently, the feature vector space is then clustered into two clusters using the k-means clustering algorithm, where $k = 2$. The binary change map $\boldsymbol{CM} = \{cm(i, j) | 1 \leq i \leq H, 1 \leq j \leq W\}$ can be determined using the following equation

$$cm(i, j) = \begin{cases} 1, & \text{if } ||\boldsymbol{v}(i, j) - \boldsymbol{w_1}||^2 \leq ||\boldsymbol{v}(i, j) - \boldsymbol{w_0}||^2 \\ 0, & \text{otherwise} \end{cases}, \tag{6}$$

where $\boldsymbol{v}(i, j)$ is the new feature vector (column of $\boldsymbol{V}$) of the corresponding pixel coordinate, $\boldsymbol{w_1}$ and $\boldsymbol{w_0}$ are the mean feature vectors for the classes '1' (represents changes) and '0' (represents no change). Fig. 4c shows an example of the change map. White region represents area of changes and black region represent unchanged area.

2.7 Post processing

All the previous steps are the fundamental and necessary steps to detect the changes between the two input images. Now, according to the user input from the GUI, the images can be segmented and the region of changes can be separated into large and little change.

2.7.1 Segmentation

In the part of segmentation, the images are segmented based on the pixel colors using K-means clustering. According to the generated cluster labels, each segment on the original image can be highlighted.

The k-means clustering is an algorithm that tries to minimize the distance of the points in a cluster with their centroids. The algorithm can be realized by following steps:

- Step 1: Choose the number of clusters k
- Step 2: Select k random points from the data as centroids $\overrightarrow{w_k}$
- Step 3: Assign all the points to the closest cluster centroid

$$c(n) = \arg\min_k \ \|\overrightarrow{x_n} - \overrightarrow{w_k}\|_2$$

- Step 4: Recompute the centroids of newly formed clusters (S_k)

$$\overrightarrow{w_k} = \frac{1}{|S_k|} \sum_{i=1}^{|S_k|} \overrightarrow{x_i^k}$$

- Step 5: Repeat steps 3 and 4 until
    1. Centroids of newly formed clusters do not change
    2. Points remain in the same cluster
    3. Maximum number of iterations are reached.

Fortunately, the MATLAB build-in function *imsegkmeans*() can be directly used. The only improved to be done is to relabel the clusters to a desired order since the cluster labels are randomly assigned.

Fig. 7 shows the segmentation of image 1 into 2 clusters. In each figures, the respective cluster retained its intensity while the other regions are darkened.

2.7.2 Size of region of changes

To determine the size of region of changes, the connected pixels in change map are grouped together using the MATLAB function *bwconncomp*. Large and little region of changes are classified by setting a threshold to the number of connecting pixels in each group. In our program the threshold is set to 100 pixels. Thus, if a region of change has more than 100 connecting pixels, it is classified as large changes (with label '2') and vice versa. At the output, the change map with label of '0' represents no change, '1' represents small region of changes, and '2' represents large region of changes, is passed to the GUI and used in the visualization part.

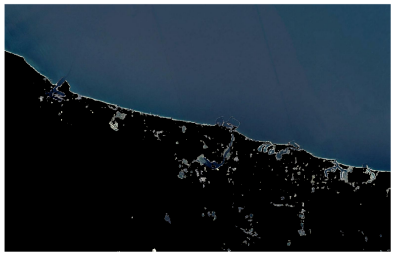


Figure 5: Segment 1

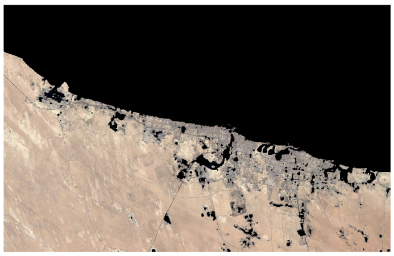


Figure 6: Segment 2

Figure 7: Segmentation of image 1 into 2 clusters.

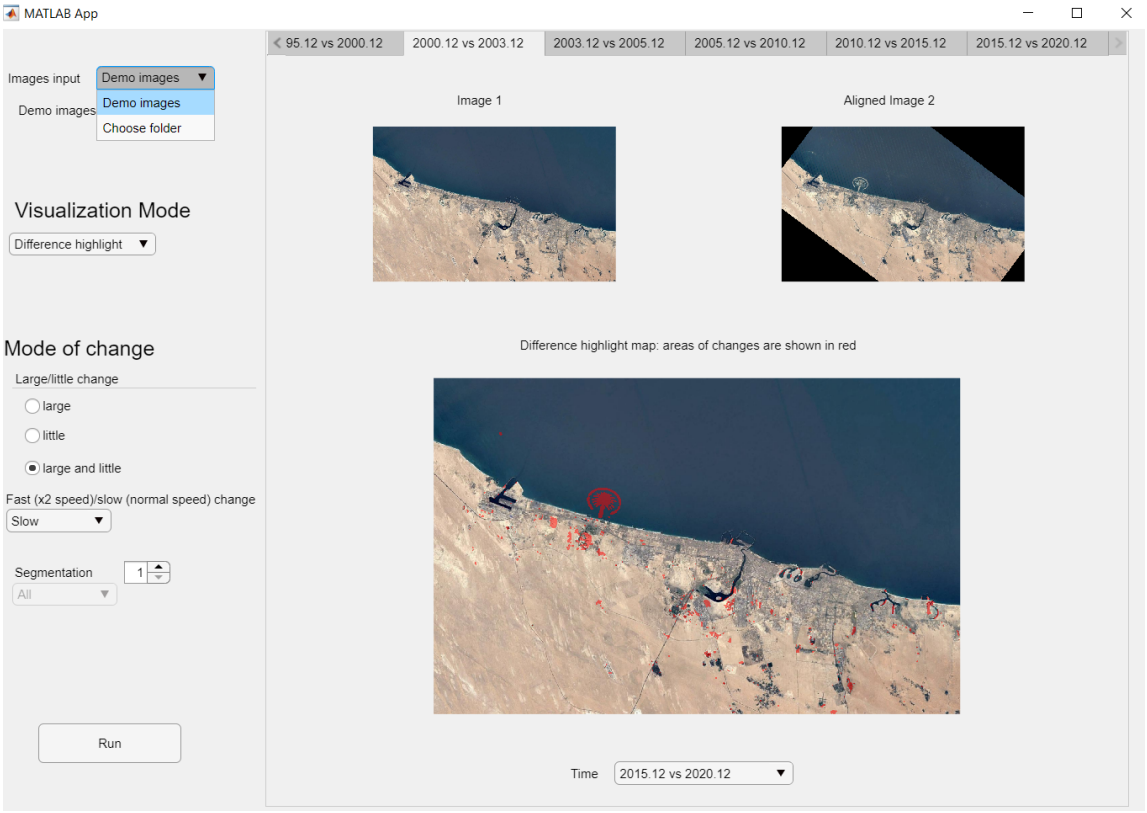


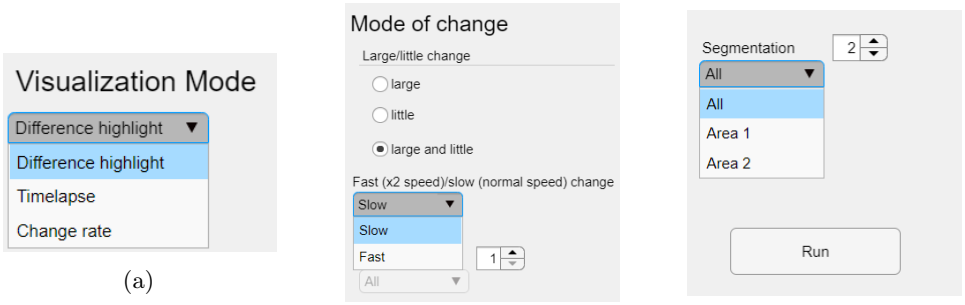
Figure 8: GUI design.

3 Graphical User Interface (GUI)

Fig. 8 shows the design of our GUI. The left panel of the GUI is the control panel, where the user can make several options such as input method, visualization mode and the desired mode of change. The right panel is reserved for the visualization output such as images, plots and GIFs.

3.1 Input method

Users can either use the demo images included in the application folder or manually select a folder by choosing the option 'Choose folder' under images input on the top left part of the GUI.



(a) (b) (c)

Figure 9: (a) Visualisation mode, (b) Mode of change and (c) Segmentation option.

3.2 Visualisation mode

There are 3 visualization modes to be chosen (Fig. 9a). Difference highlight mode compares every image pair in the input folder and marks the region of changes in red, which can be seen in Fig. 8.

In timelapse mode, the change of landscape over time is progressively demonstrated. This can be done by selecting a reference image in the input folder and align all other images to the reference image using the estimated homography and slowly display the next image (ideal case) or interpolating the next changes in pixels that are given by the change map. The latter helps the user to focus primarily on changes in the landscape detected by our program and avoid the influence of different brightness and illumination. As an example some frames of the generated GIF is shown in the left column of Fig. 10. There are 2 options in the timelapse visualization (Fig. 11, which allows the user to compare the change detection and the performance of our program. To reduce the computation time, the estimated homographies from the image processing block are reused to align the images with the reference image. To minimize errors in the alignment, the image in the middle of the input folder is selected to be the reference image.

The last visualization mode shows the change rate of the consecutive image pairs by computing the ratio of the number of changed pixels in the change map to the total number of pixels. An example can be seen in Fig. 12.

3.3 Mode of change

Users are given the freedom to choose the visualization of a desired type of changes by selecting the options available as shown in Fig. 9b. The first option (large/little change) uses the change map from Section. 2.7.2 to display the changes according to the user's choice. Fig. 8 shows an example of the default setting (large/little) and Fig. 13 illustrate the result with large and little change option.

Next, there is a fast/slow option which determines the changes at different speed. The fast option basically skips one image in the folder every iteration when computing the change map, while the slow option goes through every image in the input folder. For the fast option, the folder has to contain at least 3 images.

Last option available is the segmentation option, where the user has to define how many segments the image should be segmented. The user can then select the desired segment to be observed as shown in Fig. 7. An example of with 2 segments in the visualization mode timelapse is given in the middle and right column of Fig. 10.

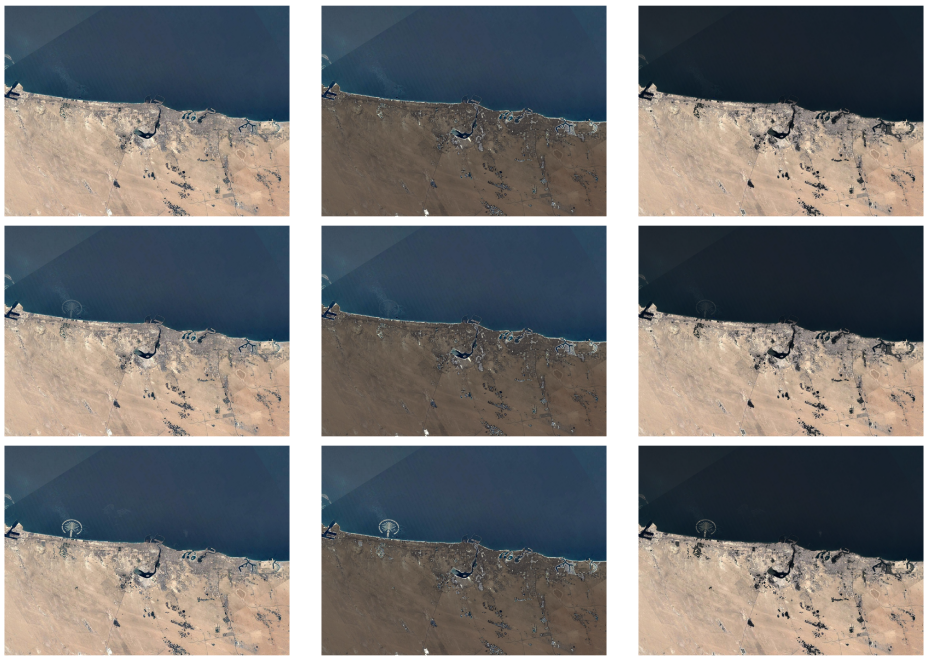


Figure 10: The three columns from left to right are frames without and with segmentation effect. From top to bottom, the images are changing progressively (based on change map option).

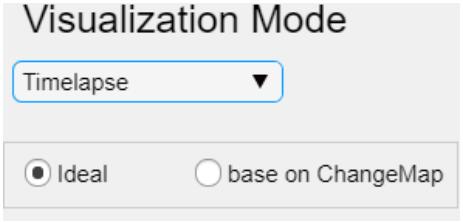


Figure 11: Options in timelapse visualization mode.

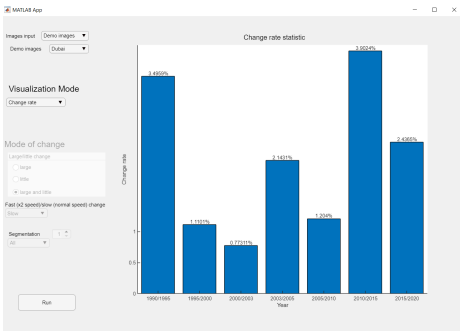
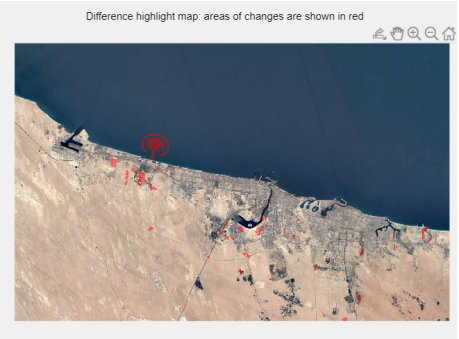


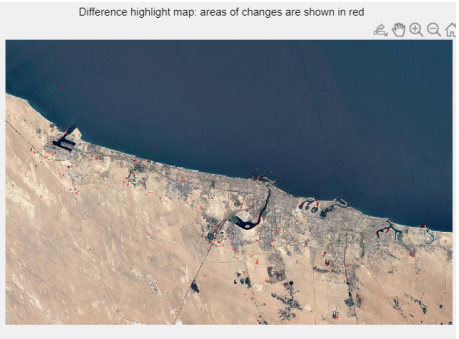
Figure 12: Change rate of the Dubai image set.

3.4 GUI flow

To make the application computationally efficient, the application caches the computed data such as estimated homographies and change maps, and reuse these data when possible. Red dotted lines in Fig. 1 shows the steps that will be re-initiated when corresponding new changes are made by the user in the GUI.



(a) Large changes.



(b) Little changes.

Figure 13: Difference highlight mode with large or little changes.

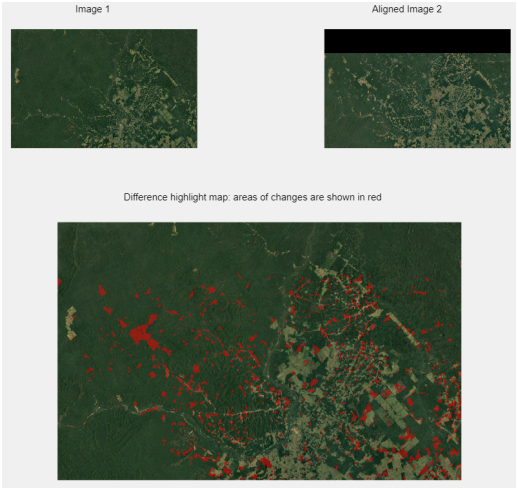
4 Results and discussion

The change detection application developed by our team works well only under following conditions:

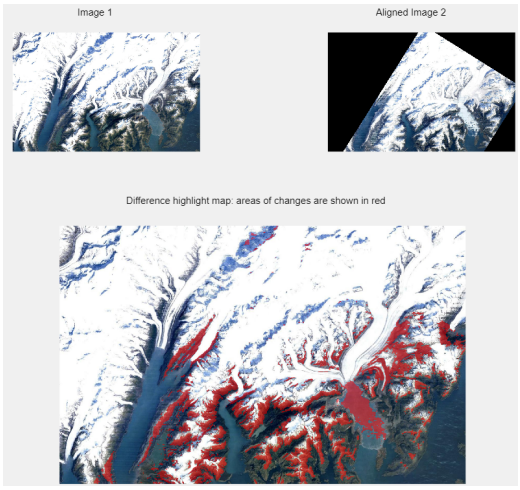
- Input images are taken at a high altitude
- Input images have even illumination
- Input images have a large homogeneous surface

This can be seen in Fig.14, which displays the results of our application on the demo image sets. Most images in the Brazilian rainforest, Columbia glacier, Kuwait and Dubai image set fulfills these conditions, thus the performance is way better compared to the Frauenkirche and Theresienwiese data set. The Frauenkirche image set has a lot of shadows and most importantly, the camera altitude is not high enough, thus the assumption made in Section 2.3 is violated. This means that objects in the image do not lie in a plane, which makes alignment impossible with the estimated affine homography. This effect can be observed at the tower part of Fig. 14e. The Theresienwiese image set (city images) do no perform well, because most part of the image are non-homogenoous surface (houses). When the alignment is slightly off, the difference map will has a large value in this housing region and then classified as change area in the change map. Other than that, in the middle part Fig. 14f, it can be seen that all the newly built buildings are not detected. A possible reason for this is because the roof and the ground are similar in color. This leads to a relative small difference value in difference map compared to the housing region. Consequently, the new buildings in middle part are not classified as region of changes and most of the changes are in the housing area.

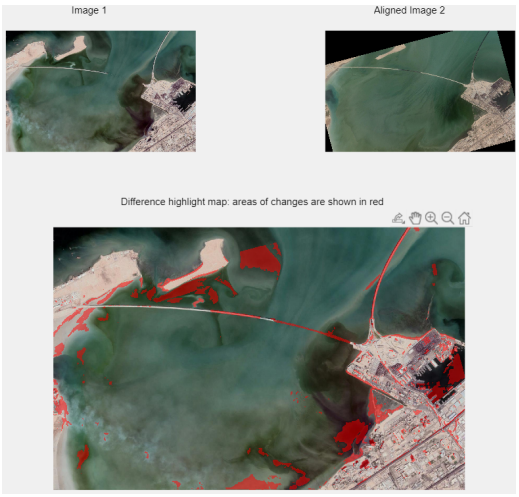
In short, when the input image does not fulfill the above conditions, the our program will most likely has a bad performance. This includes the city images (Frauenkirche image set and Theresienwiese image set) and seaside images with highly uneven illumination (such as some images in Kuwait image set).



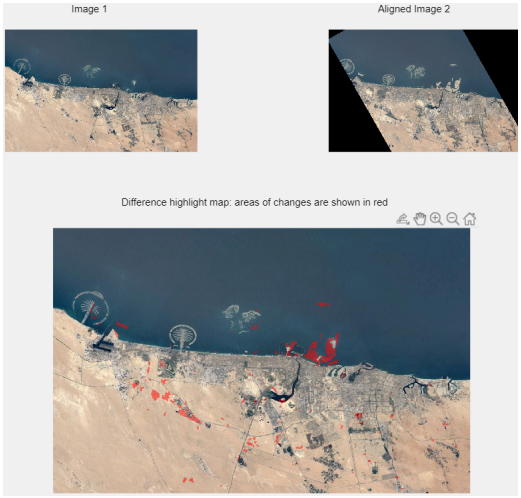
(a) Brazilian rain forest.



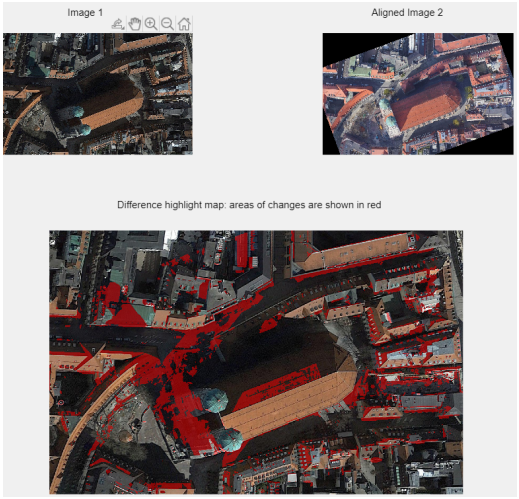
(b) Columbia glacier.



(c) Kuwait.



(d) Dubai.



(e) Frauenkirche.

(f) Theresienwiese.

Figure 14: Change detection results on different sets of images.

References

[1] Hervé Abdi and Lynne J Williams. "Principal component analysis". In: *Wiley interdisciplinary reviews: computational statistics* 2.4 (2010), pp. 433–459.

[2] Pablo Fernández Alcantarilla, Adrien Bartoli, and Andrew J Davison. "KAZE features". In: *European conference on computer vision*. Springer. 2012, pp. 214–227.

[3] Turgay Celik. "Unsupervised change detection in satellite images using principal component analysis and k-means clustering". In: *IEEE Geoscience and Remote Sensing Letters* 6.4 (2009), pp. 772–776.

[4] Philip HS Torr and Andrew Zisserman. "MLESAC: A new robust estimator with application to estimating image geometry". In: *Computer vision and image understanding* 78.1 (2000), pp. 138–156.