Computer Vision Challenge - Group 30

R. Jacumet, M. Schneider, O. Inak, A. Misik
Chair for Data Processing, Technical University of Munich
<r.jacumet, ga27jik, onat.inak, adam.misik>@tum.de

***Abstract*—Human impact on the Earth's ecosystem has increased exponentially over the last century. Various indicators can be used to analyze the impact, such as air pollution levels, species extinction rates, or average surface temperature changes. This report proposes a satellite imagery-based approach for the impact analysis. Given a series of distorted images representing a location on Earth at different times, the approach visualizes the changes between the images. There are several tasks to be solved for such a visualization - the preprocessing and matching of images, the computation of transformations needed for perspective alignment, and a quantification of the changes, which is mainly done by pixel differences. In addition, a heuristic for automatic segmentation of image regions is proposed. The methodology used for matching and perspective alignment, a SURF feature extractor combined with a transformation optimization algorithm, solves the task robustly and with high efficiency. The k-Means based heuristic used for semantic segmentation shows potential, but only for coarse regions, such as sea and land. Superpixels were used to provide a more meaningful representation of the detected image changes.**

***Keywords*—Computer Vision, SURF, MSAC, Correspondence Estimation, Graph optimization, k-Means, Semantic Segmentation**

I. Introduction

The extent of human impact on the Earth's ecosystem has reached immense proportions in the recent century. Negative consequences such as climate change, water shortages and man-made disasters have become more acute and have reached global proportions. Analysis of human influence is a rather complex problem given the proportions involved. One possible holistic approach is a satellite image-based analysis of ecosystem changes at different locations on Earth.
In this work, we develop a MATLAB-based tool, allowing the user to robustly detect and visualize changes occurring on Earth, using satellite images taken at different times. After uploading the respective recordings of interest, the images are matched to each other in terms of size, intensity, colors and perspective. Therefore we deploy various preprocessing steps and a feature-based correspondence between temporally ordered image pairs is established. The speeded up robust features (SURF) feature detector presented in [1] is chosen for this task due to its high robustness to image perturbation. A graph theory-based algorithm is deployed, allowing for efficient calculation of needed transformations. The whole matching process is later needed for detection of actual changes that happened at the places shown and not just differences in the recordings as e.g. lighting, rotations or image quality. Having the matched pictures, the user can choose between various forms of change-visualization and select parameters via a graphical user interface (GUI). This includes highlighting changes based on the their magnitude, the size of the differing areas or the speed at which a change appeared. Using an image segmentation approach, the user can additionally decide in which regions the changes should be detected.
This Paper is structured as follows. Section II introduces the methodology for matching the satellite images, region segmentation, and the subsequent detection of different kind of changes, Section III gives an evaluation of the developed algorithms. After the discussion, a conclusion is given in Section IV.

II. Methodology

The methodology section is split up into two core modules, Module 1 dealing with the preprocessing and subsequent matching of the pictures, Module 2 containing functionalities for the detection and visualization of changes between different images. Furthermore, a semantic segmentation approach is proposed for the detection of regions in the satellite images.

A. Module 1 - Preprocessing and Matching of Images

In general, the satellite images uploaded by the user vary in intensity, color and most important for recognizing changes: perspective. Therefore Module 1 has to match the pictures to each other, where the main work stems from finding transformations that rotate and translate images to be the taken from the same perspective as a so called reference image. Figure 1 shows an overview of Module 1. Taking in three pictures varying in rotation, translation and slightly in intensity, the algorithms used in Module 1 robustly transform the images to match the reference image, here chosen as the middle one.

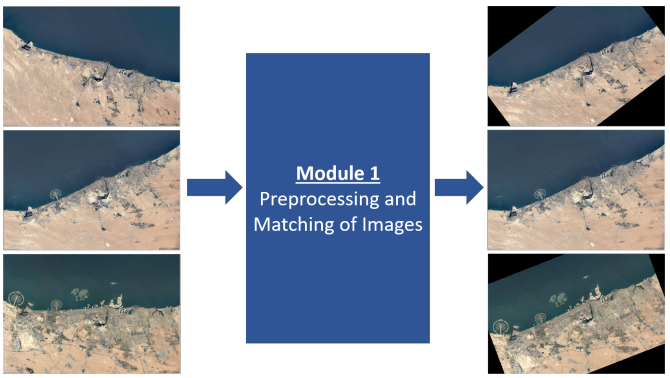

Fig. 1. **Overview Module 1:** Module 1 takes in multiple pictures, here the three on the left side, having different intensity and perspective in general. It then outputs pictures that are now matched to the reference image, here the middle one.

When preprocessing the images, we tried 2-D image flat-field correction, color matching and techniques for different intensities such as histogram matching or equalization. Due to the great variance of landscapes and locations that can be uploaded, our best performing algorithm did not use any of these methods in Module 1. The high matching qualities stem from the optimized algorithms explained in the next paragraphs.

The central part of Module 1 is the estimation of transformation matrices in order to rotate and translate images to match a reference image, allowing for change detection and visualization afterwards. For this, two images are taken, where one is the reference image that the second, so called moving picture, should be adapted to. Module 1 implements the SURF feature detector, which was chosen due to its high robustness and efficiency when dealing with images showing the perturbations we face in this problem [1]. Features from both images are extracted and matched to each other using the sum-of-squared differences (SSD) metric. Transformations are estimated by a function already implemented in MATLAB. The estimation is done by a function already implemented in MATLAB. It uses the MSAC algorithm [2] to increase the robustness of the estimation. The MSAC works similar to RANSAC, but uses a more sophisticated criteria than only the number of inliers to evaluate a possible solution. The returned transformation rotates and translates the moving image to match the reference one.

For visualizing the changes, especially for the timelapse of differences, it is of great importance to filter out wrong transformations, since this would lead to images shown from a different perspective than the reference image. As a result, Module 2 is not able to detect and show relevant changes. In some cases, SURF and MSAC fail to find correct matching features and therefore the estimated transform is wrong, being the reason for employing an extra validity check. An example for a failed transformation is given in fig. 2. A failed transformation does not correspond to the actual transformation between the images and does not lead to a meaningful image, from which differences could be extracted.

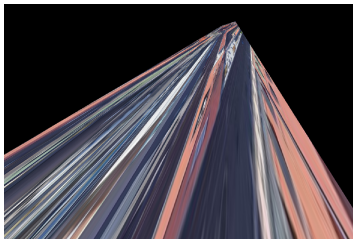

Fig. 2. **Failed Transformation:** This is an example of a false transformation. It subsequently produces a distorted image.

One way to distinguish between a valid and a false transformation is by inspecting the structure of the transformation matrix. The estimated transformation matrix T (the transposed of a homography matrix) corresponds to the transformation of the image points between the two images. It thus holds:

$$x_2 \sim x_1 T \,, \tag{1}$$

with x_1 being the homogeneous pixel coordinate of an image point in image one and x_2 being the corresponding homogeneous pixel coordinate of an image point in image two, both given as row vectors. In this code the transformation matrix is used to align image two with respect to image one, via interpolation based on the transformation defined by T.

The basic idea for determining the validity of a given transformation is that satellite images are taken nearly perpendicular to the surface of the earth. As a result most of the rotation of the ground plane between two images must take place along the z-axis of the camera frame ('in the image plane'). Strong rotation along x and y are an indication that the estimated transformation is wrong, as this is unlikely for a satlite image. This insight is visualized in Figure 3.

The exact criteria to be evaluated would therefore be that T is proportional to rotational matrix of a rotation along the z axis, added with a translation (along x,y and z) in homogeneous coordinates. Assuming same height of the satellite (no translation along z-axis) within one set of images this gives that

$$T \sim \begin{bmatrix} \cos\alpha & -\sin\alpha & 0 \\ \sin\alpha & \cos\alpha & 0 \\ t_1 & t_2 & 1 \end{bmatrix} \tag{2}$$

has to be satisfied for some angle α of 'in plane rotation'. A check of this criteria has been implemented in a fast way, that also allows for some deviation in out of plane rotations and satellite height to not unnecessarily discard a suitable transformation.

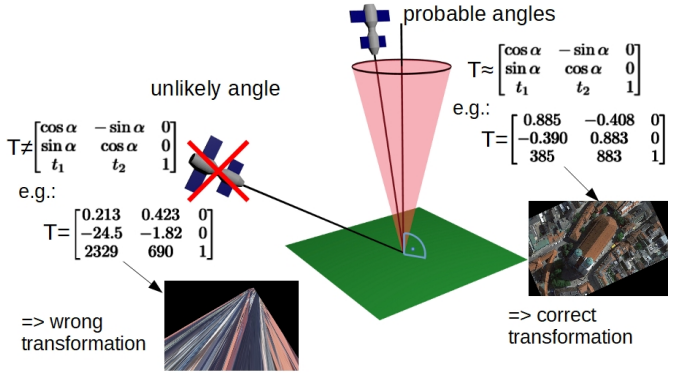

Fig. 3. **Visualization:** Satellite orientations and corresponding transformation matrices. This relationship helps to detect false transformations.

In some cases no correct transformation from an image to the reference image is found. In order to still map the image to the reference image, a chain of transformations is used, since it might be possible to map it to another image, which in turn has a valid transformation to the reference image. The resulting chain of transformations corresponds to a multiplication of matrices in homogeneous coordinates. Assume you have image

3 as the reference image and a transformation $T_{2,3}$ from image 2 to image 3 has already been found , but no valid transformation from image 1 to image 3 could be estimated. If you can now find a correct transformation $T_{1,2}$ from image 1 to image 2, you can also map image 1 to image 3 via a two step transformation. The resulting homogeneous pixel coordinates x_3 of an image point x_1 of image 1 projected into image 3 are then given by:

$$x_3 \sim x_2 T_{2,3} \sim x_1 T_{1,2} T_{2,3} , \quad (3)$$

with x_2 being the corresponding pixel coordinate of x_1, transformed into image 2.

Given the ability to check and link transformations, in most cases a valid transformation over one or multiple steps can be obtained. You could for example simply estimate the transformations between all possible pairs of images and then for each image pick a valid path to the reference image. It is computationally expensive to estimate a transformation between two images. Therefore the estimation should be done as seldom as possible. Furthermore a long chain of transformations can result in a more imprecise linked transformation, as small estimation errors propagate and accumulate. For that reason a graph search has been implemented, that aims at estimating as little transformations as possible, while also returning the transformation based on the shortest chain of transformations for each image to the reference image.

The first step of the graph search is to select the reference image as the middle image of the sequence. The reasoning behind it is that normally, the closer two images are in time, the less elements have changed in the image. As a result more image features can be matched correctly and it is more likely to find a transformation. Choosing the middle image as reference image makes it likely to find a transformation from any of the other images already in the first step.

The algorithm first tries to link every image directly to the reference image. It calls the function for estimating the transformation and saves the result if a valid transformation is returned. From there on it iteratively tries to find connections over an increasing amount of steps. In every new iteration it tries to link any so far unconnected images to the images linked in the prior iteration. When doing so, the previously linked images are always sorted so that the closest images (with respect to the currently selected unconnected image) are tested first, further reducing the amount of conducted transformation estimations. Trying to link images corresponds to estimating a transformation from the unconnected image to the other image and evaluating its validity. If a valid transformation is found for a currently unconnected image, the transformation is saved and the next unconnected image is tried. The algorithm stops when having connected all images or when no new images could be connected during the current step. The first case means, that all images have been connected successfully and the transformations of all images to the reference image are returned. The latter corresponds to the case, that not all images have been connected, but no additional connections can be found. In this case only the obtained transformations are returned and the not connected images are ignored in the rest of the program. The described procedure is depicted in Figure 4.

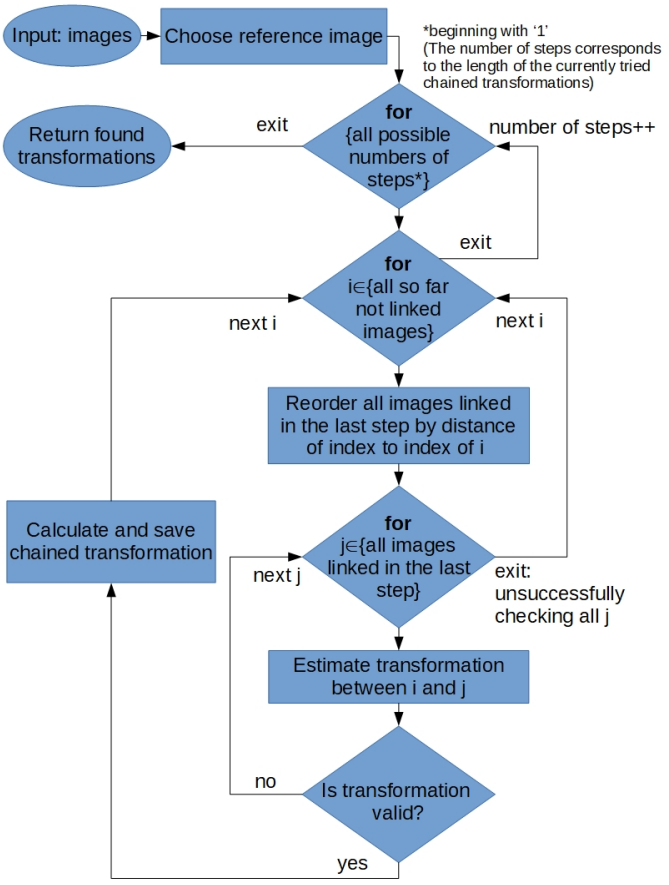

Fig. 4. **Flow chart:** Basic structure of the used graph search algorithm.

An example for the working of the graph algorithm is shown in Figure 5.

B. Image Segmentation

An interesting feature for the analysis of human influence is the inspection of certain image regions. This can be used to analyze how humans have changed certain parts of the ecosystem - be it landscapes or rivers. To be able to implement such a feature, the task of semantic segmentation needs to be solved.

For a semantic segmentation of image regions, an adaptive k-Means clustering of grayscale images is chosen, inspired by the work presented in [3]. The core idea behind this heuristic is that the number k of initial cluster centers is controlled by the user - the more semantic regions are selected for the segmentation task (cities, countryside, sea, rivers), the more cluster centers are needed. An assumption in this heuristic is that each semantic region is characterized by an average pixel intensity. While sea regions are characterized by lower average

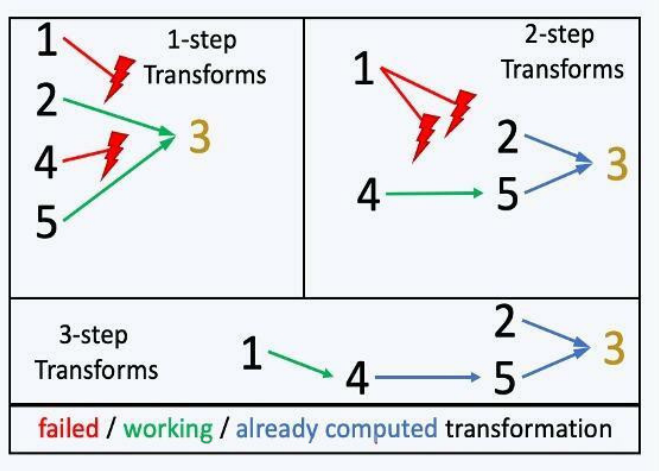

Fig. 5. **Example:** Image '3' is the reference image. In the first step a transformation from image '3' and '5' to image '3' could be found. Image '1' and '4' could not be connected. In the next step for each remaining image '1' and '4' a two step transformation over '2' and '5' was tried. As image '1' still could not be connected, a three step transformation over '4' was tested and was successful. If it failed, image '1' could not have been transformed w.r.t. the reference image '3' and would have been ignored for the rest of the program.

pixel intensities, cities have higher intensities. Through a series of experiments with different combinations of regions, a sorted pixel intensity ranking is created for the regions of cities, land, sea, and special landscapes (e.g., glaciers). By creating such a ranking of regions based on their pixel intensity, one can directly assess which cluster (characterized by a center pixel intensity) belongs to which region. For example, if the user wants to examine only land regions, the k-Means algorithm is initialized with $k = 2$. The cluster with the higher pixel intensity center is selected as the representative region for land. This assumption holds for several datasets, such as Dubai, Columbia Glacier, or Brazilian Rainforest.

Based on the regions selected by the user, a binary mask is created through which the relevant image regions are cropped.

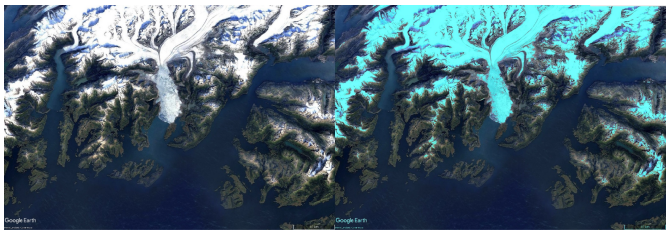

Fig. 6. **Segmentation Example:** Example semantic region segmentation for glacier regions in images of the 'Columbia Glacier' dataset.

C. Module 2 - Change analysis and Visualization

After the satellite images have been brought to a common perspective, the goal is now to analyse the changes occurring over the time in the images. The functionalities of Module 2 can be segregated into four visualization options:

- Historical timelapse
- Comparison timelapse
- Highlights
- Two image comparison

For all the different options mentioned above, a preprocessing of the transformed images regarding the corresponding folder is done. In Module 1, we tried to align all the images in the selected folder by finding reasonable transformations, but we were not able to find an applicable transformation for all the images, because some images were hardly rotated or shared just small areas regarding the reference image. Therefore, it was needed to ignore all the images, which could not be transformed regarding the selected reference image and to use just the images, for which we could find an applicable transformation. In the following, we investigate the different visualization options, which are implemented for the user, in further detail.

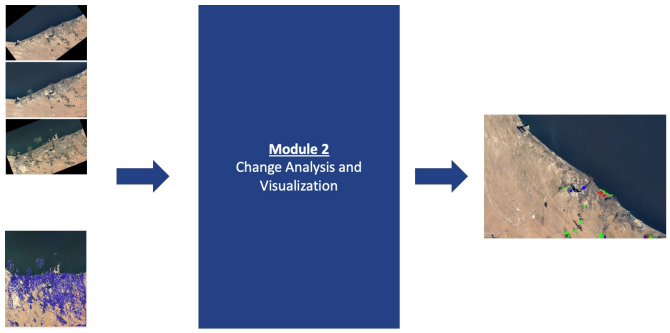

Fig. 7. **Overview Module 2:** The input of Module 2 consists of perspectively matched images and a segmentation mask, through which only selected regions are analyzed. The functions within Module 2 then create a change image, which optionally can have color-coded difference pixels.

1) Historical Timelapse: In this section, our main goal is to visualize the differences between images in a historical timelapse. The oldest image from the folder is chosen as the reference image and the differences between the reference image and other images are visualized in a timelapse plot. Thus, the user has the opportunity to see how the corresponding landmark has changed over time.

While selecting the oldest image in the folder as the reference image, the other images in the same folder have to be reconstructed regarding this new reference image. For this purpose, we saved the transformations of all images to the initial reference image (middle image of sequence) in Module 1. With these transformations we can simply map the images to the perspective of the new reference image. For each image its stored transformation to the reference image is concatenated with the inverse of the stored transformation from the desired new reference image to the old one. The concatenation is computed according to equation 3. The calculated transforms are then applied to each image correspondingly. As a result, all the images are aligned regarding the oldest image and ready for the further operations.

Since we have all the images aligned regarding the oldest image, we can calculate the differences between the images

in a for-loop. In each for-loop, the oldest image is selected as the reference image and the other images are selected as moving images one by one. In this section, rather than calculating the differences between each pixels, a different approach with segmentation is used to detect the areas, which have significantly changed over time. For this, the landscape is divided into superpixels, which are selected adaptively to the different surface types and changing gradients in the corresponding moving image, as seen in Figure 8. From now on, the moving images are not considered pixel vise but rather with superpixels, which dynamically adapt themselves to different surface types in the corresponding image. It is beneficial for us to use superpixels to differ big areas that have significantly changed over time from the others.

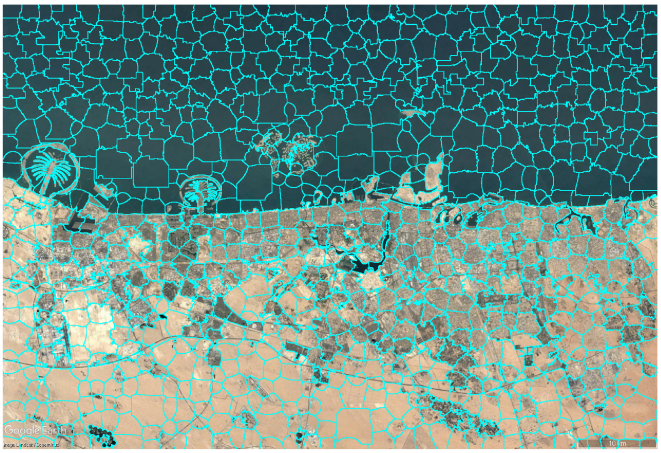

Fig. 8. **Superpixels:** Image from Dubai divided into 1000 superpixels (cyan).

Now, each superpixel is investigated individually and the mean of the total change to the corresponding superpixel is calculated for each superpixel. We do not want to set a static threshold for visualizing the changing areas, since this threshold would change for every different landmark. Therefore, we defined ourselves an adaptive dynamic threshold by finding the biggest mean of the total change in a superpixel in the corresponding moving image. In order to differ the areas, in which we can observe big changes, from the areas, in which we can observe small changes, we defined a dynamic threshold regarding the above stated biggest mean of the total change in a superpixel. All the areas which show big changes like the calculated biggest mean, are marked as red. Areas with intermediate changes are marked as blue and areas with small changes over time are marked as green in the final plot. The user also has the opportunity to set this dynamic threshold in the GUI and see the historical timelapse accordingly. In Figure 9, one step from the visualization of the historical timelapse is depicted. On the left-hand side we can see the oldest image in the folder (an image of Dubai in 1990) as the background image and on the right-hand side an image of the Dubai from the year 2015. The areas marked with red are the areas, the biggest changes over time have observed, with blue the areas with intermediate changes and with green small changes. From this comparison, we can conclude that islands are built in Dubai and the city has significantly expanded.

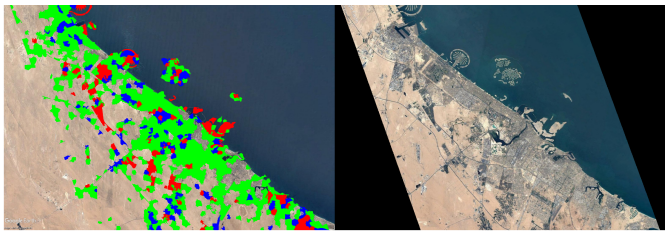

Fig. 9. **Historical Timelapse in Dubai:** The background image on the left side is from the year 1990 (oldest image in the Dubai folder) and the image on the right side is taken in 2015. (red: big changes, blue: intermediate changes, green: small changes).

2) Comparison timelapse: This section principally uses the same approach to visualize the differences between images in a timelapse as the Section II-C1 (Historical Timelapse) with only one difference. In this section, the images are not compared regarding the oldest image, but rather with their previous image, in order to show differences that occur in a short timelapse. For that reason, the reference image is changed in every iteration of the for-loop regarding the corresponding moving image in the current step and the moving image is inverse transformed regarding the reference image (in this case the previous image of the moving image). An example of the comparison timelapse is shown in Figure 10.

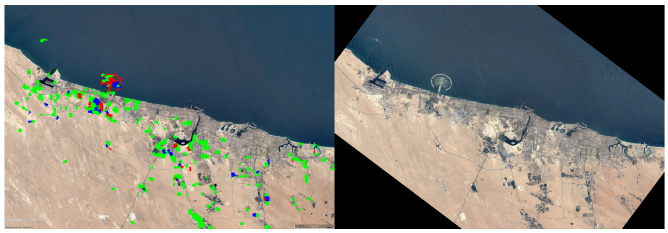

Fig. 10. **Comparison Timelapse in Dubai:** Background image on the left-hand side from the year 2000 and the image on the right-hand side from the year 2003 (red: big changes, blue: intermediate changes, green: small changes).

3) Highlights: In this type of visualization, pixels, where the most change appeared over all pictures uploaded, are marked. Therefore, each image is compared to its successor and the absolute differences in pixels are stored. All these differences between subsequent pictures are accumulated and afterwards sorted based on their magnitude. The user can select, which percentile should be marked, i.e. the percentage of pixels where the most change took place over all the pictures. This differs from the other types of visualizations, where we always mark changes happening between two time instances. With the highlight visualization option, the user receives information which cannot be seen by solely comparing two images, even in a timelapse. This can be understood at the example of the Dubai pictures: If we only compare the first and the last image of the Dubai folder with each other and filter out changes with the highest magnitude, we would mark the Palm Islands, which were built sometime between the two images. If we use the Highlights visualization as can be seen in Figure 11,

especially city areas are marked. Here we have smaller changes that accumulate due to these areas being in constant change.

Fig. 11. **Highlight visualization for Dubai:** The top 2% of pixels where most changes appeared over all the pictures are marked in red.

4) Two image comparison: We also wanted to give the user to compare just two images, which can be directly selected from the user, with each other. First, we select the first(older) image, which is chosen by the user, as the reference image. Then, the second chosen image, as the moving image, is transformed regarding the first image. After that, depending on the pre-defined threshold by the user, the chosen images are compared pixel-vise and the differences are visualized by red pixels on the reference image. As seen in Figure 12, two images from the Dubai folder are selected by the user in this case: image of Dubai in 1990 and image of Dubai from 2020. The first image from the year 1990 is selected as the reference image and depicted on the left-hand side as the background image. The second image from the year 2020 (image on the right-hand side) is selected as the moving image and transformed regarding the reference image, so that the reference and moving image are perfectly alligned. Then, the images are compared pixel-vise regarding a user-defined threshold and the differences are shown on the left-hand side on the reference image. Essentially, with the help of visualization mode, changes can be inspected for a specified time range.

Fig. 12. **Two image comparison for Dubai:** The reference image on the left-hand side from the year 1990 and the moving image on the right-hand side from the year 2020. The differences between reference image and moving image are plotted in red on the reference image on the left-hand side.

5) Change parameters: The first parameter that can be set after selecting the visualization mode is an intensity threshold for the changes (absolute pixel differences). This threshold is in the range from 0 to 100, and can be set with a slider element. If set, out all changes whose intensity is below the set threshold are filtered out. The intensity threshold is available for each visualization mode.

Additionally the user has the option to filter the obtained changes by the area (regions) they are covering. It is possible to filter out changes that lie in regions with only sparse changes, leaving just changes in regions that underwent large-scale changes. So it is possible to extract big regional changes. This can for example be useful in cases of urban sprawling. Here it is helpful to not only highlight new settlements as changes, but also differentiate how large different settlements grew and which sprawls grew the strongest. Also it helps the user to determine where the most alterations occurred in the landscape. Given a slider the user can set the minimum size of regions to be considered. Any changes that do not lie in a changed region of at least this size will be filtered out. The bigger the user sets the size, the coarser the regions will be determined and the more small regions of change will be ignored. This option has been realized as follows. The differences of the two images are lowpass filtered with a Gaussian filter. Then a fixed threshold is applied to the filtered image. The resulting mask from this step corresponds roughly to the regions of filter size that underwent changes. The size of the region can be adjusted by varying the filter width. To avoid narrowing of the regions due to the filtering and thresholding step, the mask is expanded along its boarders. This is done by convoluting it a second time with a Gaussian filter and applying a very small threshold. The mask resulting from this step is depicted in fig. 13 for two different region sizes. This mask is then used to mask the differences from before. The remaining differences, all lying in these regions, are then displayed to the user.

Another parameter, which can be changed by the user, is the visualization of big/intermediate/small changes. The user has the opportunity to see the areas with big, intermediate and small changes over time individually by ticking the corresponding box in the GUI, shown in Section II-D. Since the big, intermediate and small changes are defined relatively to the mean of the biggest change in a superpixel (please refer to the Section II-C1 for more detail), they can be easily labelled through the inspection of the mean. This parameter is only available for the historical and comparison timelapse.

The last separate change parameter is a slider element, used only for the highlight visualization mode. The slider allows the user to set what percentage of cumulative pixel differences should be displayed in the highlight image (e.g. show only top 2% of the difference pixels).

D. GUI

The functional blocks described above are integrated into a graphical user interface shown in Figure 14. The use of the graphical user interface can be described with a happy path, i.e. a list of steps required to accomplish the task of change visualization in satellite images:

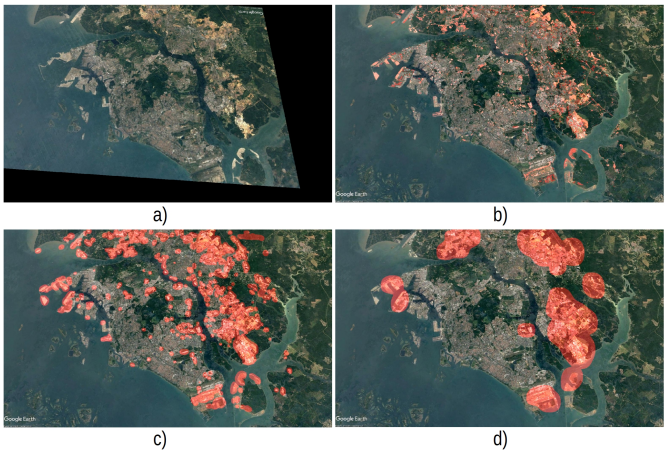

Fig. 13. **Example for regions of changes for Singapur in the years 2006 and 2018:** Images a) (2006) and b) (2018) show the two compared images, with the differences already overlayed in b). Images c) and d) are examples of regions of change for different sizes. In image c) even small regions are found, whereas for image d) only large regions are considered. The here shown regions will then be used as masks for the actual differences.

- Upload image folder
- (optional) Visualize uploaded images
- (optional) Select image regions used for semantic segmentation
- Select visualization mode
- Set change and visualization parameters
- Run visualization option
- (optional) Show change image in fullscreen format
- (optional) Export reconstructed images to files

For efficient usability, the image matching and reconstruction functions are invoked immediately after the images are provided, giving the user the impression that the images are still being loaded. The change analysis and visualization functions used afterwards are comparably efficient and thus do not interfere with the flow of use of the graphical user interface. In the end, the user gets the opportunity to export the matched (or reconstructed) images to files, which can be used for individual analysis.

After the happy path has been finished, the user is given the option to restart the change visualization with new images via a reset button.

III. Evaluation

In this section, we analyze the performance of the previously explained modules. Module 1 is evaluated on the amount of correctly transformed images, the respective computation times, as well as how many transformations had to be calculated. The performance of the k-Means algorithm used for image segmentation is inspected by computational time with varying k. Module 2 is evaluated in terms of the computational time of each visualization option. Example change visualizations of our approach can be found at the end of this section.

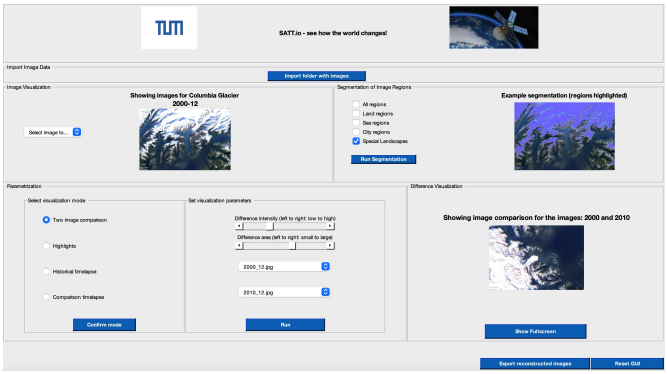

Fig. 14. **GUI:** A screenshot of the developed graphical user interface.

A. Evaluation - Module 1

Since the goal of Module 1 is to return all images matched to a reference picture, we evaluate the performance mainly on how many images we were able to correctly transform. Additionally, the computation time and number of calculated transformations is analyzed in order to show the advantages of the developed graph based algorithm.

For the evaluation, 80 images of nine different locations on Earth were used, ranging from cities like Dubai, Garching or Singapur to areas of different nature types, such as a Brazilian rainforest or the Columbia Glacier in Canada. The calculations for Module 1 where performed using a standard laptop with an i7 CPU.

Out of the 80 images, Module 1 transformed 77 $(96,25\%)$ to their respective reference image, where 76 (95%) were correct. The three images Module 1 was not able to transform, all belong to the Brazilian rainforest, whereas the only wrong transformation that slipped through was found in a dataset showing the metropolis Garching. In total, 119 transforms have been computed, meaning that on average we were able to transform an image to match the reference image in only 1.5 steps, where this number varied drastically between different datasets. As an example, for the images showing Dubai all 1-step transformations to the reference image worked, whereas the Frauenkirche dataset lead to the computation of 19 transformations for 10 images.

Looking at these numbers, we can conclude that the developed graph-based algorithm is working and needed to efficiently compute only the necessary transformations. Since 119 transformations were calculated for 80 images, we know that solely using 1-step transformations would not work to match a high amount of images to the respective reference image. On the other side, taking the naive approach and calculating all transformations from every image to every other image in order to allow for many matches would need $n(n-1)$ computed transformations, where n denotes the number of images showing the same location. Compared to this naive approach, the developed algorithm cuts down the number of calculated transforms to only 18.4%, saving time while still achieving the same extensive matching performance as the approach of checking all combinations of transforms.

Calculating a single transformation, so extracting, matching and validating features, as well as estimating the transform itself with further checking the correctness as described in the previous section, takes only 625ms on average.
The function, checking the validity of the computed transformation introduces a trade-off. If we tune the algorithm to rigorously reject all transformations that are not certainly correct, more transformations have to be calculated, i.e. longer chains of concatenated transformation matrices, leading to longer program run times. If no transformation fulfilling the demanding requirements can be found, some images can't be matched to the reference image, having negative impact for visualizations like the timelapse, where these images mustn't be displayed. On the other hand, if we tune the algorithm to be too undemanding, wrong transformations, i.e. transformations leading to a bad adaption to the reference image, will slip through more often. This would allow for more images to be matched, but more of them incorrect. Showing a user wrongly matched images, i.e. images where also the changes are useless or even worse: misleading, is worse than recognizing a transformation is incorrect and hence not showing it. Therefore we decided for a stricter validation process, increasing the number of calculations and making it not possible to find a transformation for one image in the Brazilian Rainforest dataset, but greatly reducing the number of undetected incorrect transformations to only one, as compared to a less strict checking procedure.
Overall, Module 1 is able robustly match given images to a reference image due to the transformation validation process. The graph based algorithm significantly decreases the time needed for Module 1, by decreasing the amount of calculated transformations, while still performing an extensive search for possible transformation concatenations. The high robustness of accepted transformations comes at the cost of not finding any transformations, and therefore not matching, a small percentage of the pictures.

B. Evaluation - Image Segmentation

To evaluate the computational complexity of the proposed semantic segmentation approach utilizing k-Means, the Dubai, Kuwait, Columbia Glacier, Brazilian Rainforest, and Frauenkirche datasets are evaluated with $k = 2, 3$ (i.e., with 2 or 3 regions selected). The results of the experiment are depicted in Table I. Again, the experiments run on a standard laptop with an i7 CPU.

TABLE I. RESULTS OF THE K-MEANS SEGMENTATION RUN TIME WITH VARYING K.

k	Dubai	Kuwait	Columbia Glacier	Brazilian Rainforest	Frauenkirche
2	19.38 s	24.35 s	27.53 s	15.88 s	30.89 s
3	28.27 s	36.46 s	47.55 s	24.93 s	60.80 s

The results in Table I highlight the sensitivity of the k parameter. In the case of the Frauenkirche dataset, an increase from $k = 2$ to $k = 3$ resulted in double the computational run time. A further increase to $k > 3$ thus could lead to non-linearly increasing run time, with only minor improvements in the segmentation. Therefore, it is not recommended to use the used segmentation algorithm with $k > 3$. This insight was also taken into account in the development of the graphical user interface, where a maximum of 3 regions can be selected.

C. Evaluation - Module 2

In Module 2, we were able to visualize the differences regarding every possible option mentioned in II-C for all the images in the selected folder. For most images, the changes can be found and visualized very well. In few cases, differences in images are considered as changes although the landscape maintained its state. These wrongly detected changes are usually shadows, clouds or especially differences due to 3-dimensional elements in the scene combined with an altered perspective. An example for these 3-dimensional elements with a change in perspective can be found in the Frauenkirche dataset, where buildings rise from the ground plane. The effect is portrayed in fig. 15.

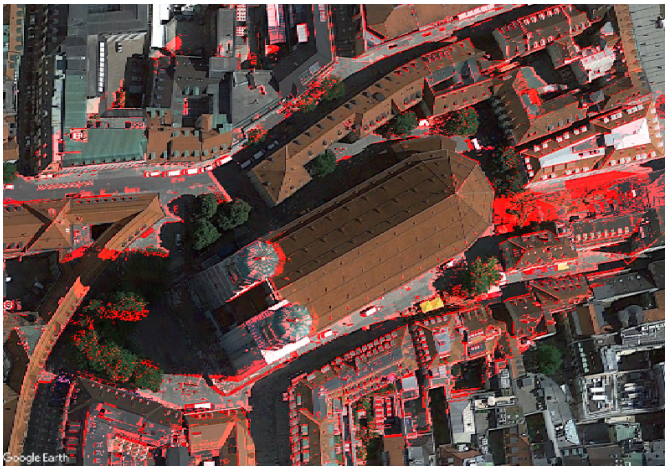

Fig. 15. **Altered perspective on Frauenkirche:** Here the buildings generate differences due to the change of perspective, although they themselves did not change. Most notably are the two towers. In this image the abundance of these wrongly highlighted changes veil the highlighted actual change (construction site on the right image border, slightly above the middle).

Apart from these exceptions, the visualization works as supposed and therefore, we evaluate the performance of Module 2 mainly by the computational time needed to apply the individual options. In the following, the mean computational time of the different options are shown. These results are obtained considering 9 different folders and obtaining the corresponding computational time for each option and calculating the average of the results. For these calculations, i7 9700K as the CPU and RTX 2070S as the GPU are used:

- Historical Timelapse (for 3000 superpixels): 36.97s
- Comparison Timelapse (for 3000 superpixels): 36.08s
- Highlights : 0.96s
- Two Image Comparison: 0.87s

Historical timelapse and comparison timelapse have a significantly longer computational time in comparison to the other options. There are 2 reasons for that. First one is that image

comparison is done regarding all the images in the selected folder in a for-loop and not just two images, which is the case for the option two image comparison. The second reason is that the computation time is highly dependent on the number of superpixels. The moving image is divided into superpixels and larger the superpixel number is, longer the computation time, because each superpixel is investigated individually in a for-loop.

For our GUI implementation, we used a superpixel number of 1000. We investigated numerous image comparison visualization with different settings for the number of superpixels and decided that the number 1000 was the optimal one for quality of the illustration with the capability of showing detailed segmentation between different small areas in a landscape but also achieving a reasonable computation time. If we have used rather 250, the computation time would be four times lower, but the quality of the illustration and change segmentation would be also lower, since the corresponding moving image is divided into larger independent pieces. On the other hand, if we had used 3000, this would lead to a higher quality change segmentation, but would also significantly increase the runtime, leading to usability problems.

D. Example Visualizations

In this section, three example visualizations of scenes with meaningful insights are given. The first visualization is a highlights image from the Dubai dataset. The top 10% difference pixels accumulated over the images are shown in Figure 16.

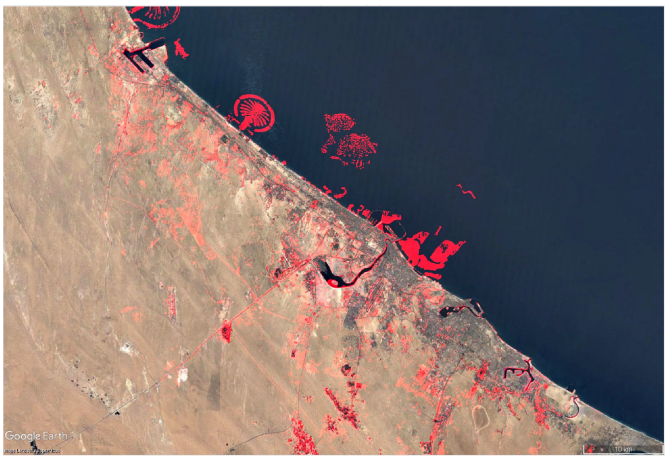

Fig. 16. **Highlights image for Dubai**

Some interesting patterns become visible, such as the emergence of the infamous Palm Islands and the World Islands (located east of the Palm Islands), indicating the rapid urbanization of the city of Dubai.

Another interesting insight comes from a historical timelapse of the Columbia Glacier dataset. The accumulation of superpixels around the main branch of the glacier shows how much the branch has melted over the course of 20 years (Figure 17).

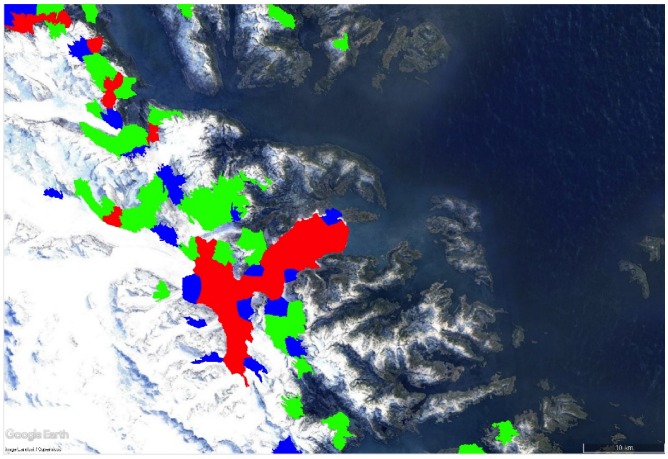

Fig. 17. **Historical Timelapse for Columbia Glacier**

The last example is a two image comparison of images from the Brazilian Rainforest dataset (Figure 18). The images were selected from 2000 and 2020 to highlight the significant changes in the ecosystem in the rainforest region during the 20-year period. It can be seen that in 2000 (left image) the studied region was predominantly covered with a rainforest area. In 2020 (right image), areas created by human activities (e.g., deforestation and settlement construction) cover most of the image. The marked differences in the right image thus indicate forests that have been replaced by human activities.

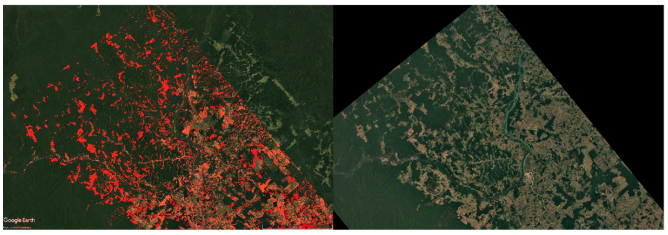

Fig. 18. **Two image comparison for Brazilian rainforest**

IV. Discussion and Conclusion

Module 1 takes care of reorienting and aligning the images. As depicted in the evaluation, it has been tuned for robustness and speed. It performs well in a sense that it transforms the images to the same perspective, with only few transformations not being found and even fewer transformations being wrong, while finishing the task in a time reasonable for the user. To further improve the reliability and calculation time, the amount of robust features, extracted in the first step, needs to be increased. The more features are matched correctly, the more likely a valid transformation between images can be found and on average less steps are necessary to obtain a chained transformation. Although the SURF feature detector is fairly robust and fast, a combination with a different feature extractor might possibly enhance the performance of Module 1. Additional improvement might be made by further optimizing

the parameters of the used SURF and MSAC algorithm. These two aspects could be a starting point for future work.

Another open problem for Module 1 is the Google logo located in the lower left corner of each satellite image. Methods that would automatically detect and crop out this distracting element could be explored in further work.

The proposed segmentation heuristic provides a computationally efficient and low-complexity approach for semantic segmentation of image regions. However, the established ranking of regions was adapted to the dataset, which reduces the generalization to unseen datasets. Thus, the heuristic would fail the segmentation task if the ranking of average pixel intensities were different for city, land, sea, and special landscape regions. Furthermore, in the case of some datasets (such as for Kuwait, see Figure 19), no reasonable city region segmentation was found. These insights points to the low robustness of the approach. Future work will examine a segmentation approach with higher generalization and robustness, inspired by recent findings from the machine learning domain [4].

Fig. 19. **Failed Segmentation Example:** Example failed region segmentation for city regions in images of the 'Kuwait' dataset.

Module 2 works efficiently and successfully by comparing images with each other and visualizing the differences in a clear way with numerous different illustration options. The historical and comparison timelapse could be computationally optimized further in a future work, so that the duration of the visualization based on superpixels can be decreased. The only downside of the visualization was again the Google logo, that is located on the left-down corner of every sample data/satellite image and already presents a problem in Module 1.

This report proposes an efficient approach for automated satellite image analysis that can be used to study ecosystem changes anywhere on Earth. For image matching and perspective normalization, a combination of the SURF feature detector and a graph-based optimization algorithm required to compute the Euclidean transformations between images has been shown to be particularly effective. For the image segmentation task, a k-Means-based heuristic was developed that can be used in a crude sense for region detection. In the change visualization task, superpixels were used to provide a more meaningful representation of the computed differences.

REFERENCES

[1] H. Bay, T. Tuytelaars, and L. V. Gool, "Surf: Speeded up robust features," in *Computer Vision – ECCV 2006*, A. Leonardis, H. Bischof, and A. Pinz, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2006, pp. 404–417.

[2] P. Torr and A. Zisserman, "Mlesac: A new robust estimator with application to estimating image geometry," *Computer Vision and Image Understanding*, vol. 78, no. 1, pp. 138–156, 2000. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S1077314299908329

[3] M.Barthakur and K. Sarma, "Semantic segmentation using k-means clustering and deep learning in satellite image," in *2019 2nd International Conference on Innovations in Electronics, Signal Processing and Communication (IESC)*, 2019, pp. 192–196.

[4] F.Lateef and Y. Ruichek, "Survey on semantic segmentation using deep learning techniques," *Neurocomputing*, vol. 338, pp. 321–348, 2019. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S092523121930181X