Panorama Creation with Subsequent Object Removal

Erica Warriner 101002942
Daniil Kulik 101138752

Carleton University

Abstract. The ultimate goal of this project was to create a tool that could produce a panorama from 4 images and perform subsequent object removal and inpainting. The user must be able to upload several images of a scene into the program, create a panorama, and manually select a contour of the object to be removed from the scene. The removed object spot is then inpainted with a suitable background. The image stitching task is challenging because 4 images are used rather than 2, requiring more time and complexity in the solution. The inpainting task is challenging because the object removal function must be implemented with an interactive desktop application that can adequately display contours, while also preserving the panorama image.

1 Introduction

Image stitching is the creation of a panorama image by combining multiple images of the same scene. The first step in image stitching is to use feature detection to find keypoints in each of the images. Next, the keypoints are represented by descriptors, which are then used to match some overlapping points between the images. Once the overlapping points are known, a homography matrix is calculated and applied to the right-hand image to warp its perspective. Finally, the images are concatenated to form the final panorama result.

Image stitching is a good candidate for a vision application because it makes use of key computer vision concepts like feature detection and homography to solve the problem of capturing an entire scene with only one camera. Despite being conceptually simple, image stitching can become challenging in some circumstances. First, a panorama can only be created if there is sufficient overlap between the different image views. Another common issue with panorama creation is a lack of texture in the image. Without texture, the image will not have enough distinguishable features to successfully match the image points together. Furthermore, natural variation in lighting and colour between the images can make it difficult to stitch them together seamlessly.

Object removal is a good candidate for a computer vision application for both research and practical reasons. From the research perspective, we investigate how well we can restore linear structures and textures in the removed region. From the practical point of view, it can be directly used to remove unwanted objects from a scene. There are several approaches to this problem. Some of them try to synthesize the removed texture from a small source, while others sample the source image for patches. In this work we explore the second approach, which uses an exemplar-based technique (Criminisi et al., 2004). The proposed approach generates a new texture by sampling the source image and copying pixel values to the target region.

Object removal in a visually plausible way is a challenging task because it is hard to replicate consistent texture in real-life photographs. While the original method proposed by Criminisi et al. (2004) gives decent results, it is computationally expensive because it searches the whole image for suitable patches. We proposed a new method to find suitable patches by defining a patch window – an area around the inpainted central pixel where we look for possible patches. The intuition behind this method was a simple observation that similar patches are usually located near to the inpainted region. Moreover, we re-define the loss function from the original Euclidean distance and sum of squares to a mean squared error.

2 Background

There are many software packages and algorithms that can be used to complete each step of the image stitching process. Tareen and Saleem (2018) provide a comparative analysis of some of the most popular algorithms for feature detection, descriptor generation, and matching, as well as outlier rejection. To detect image features or keypoints and produce corresponding descriptors, tools such as SIFT (Lowe, 1999) and AKAZE (Alcantarilla et al., 2013) are commonly used. Depending on the chosen feature detection method, the L2-Norm (for SIFT and SURF) or Hamming Distance (for AKAZE) is used to calculate the level of similarity between keypoints of the two images (Tareen & Saleem, 2018). Next, there are several different feature matching algorithms that can be applied, including the brute force method, threshold-based matching, and nearest-neighbour (Brown et al., 2005). To ensure that the keypoint matches correctly reflect the overlapping features between the images, an outlier detection method like RANSAC (Fischer & Bolles, 1981), MSAC (Torr & Zisserman, 2000), or PROSAC (Chum & Matas, 2005) is used, discarding any outliers that are found. Finally, OpenCV's findHomography and warpPerspective functions transform the right-hand image to match the projective space of the left-hand image, and the two images are concatenated.

The object removal approach is based on the work done by Criminisi et al. (2004). It employs an exemplar-based inpainting of the target region. Our implementation does not rely on software packages except for packages that provide math operations like convolution and matrix multiplication. The main software packages for object removal are OpenCV and numpy. The OpenCV library provides a prototyping GUI that we use to generate removal masks directly from the source image.

3 Approach

To produce a panorama from 4+ images, a for-loop iterates through each of the source images. On each iteration, the current image is stitched to the consecutive right-hand image, and the resulting panorama is then used as the current image for the next iteration, until all source images have been stitched together. This implementation is different from a traditional image stitching application because it produces a panorama from more than 2 images. Furthermore, this approach is unique because several feature detection and outlier detection algorithms were tested to choose the method that produced the optimal result.

The SIFT detector was used for finding keypoints and descriptors. SIFT works by convolving the image with Gaussian filters at varying scales, starting with $\sigma = \sqrt{2}$ (Lowe, 1999):

$$g(x) = \frac{1}{\sqrt{2\pi\sigma}} e^{\frac{-x^2}{2\sigma^2}}$$

The keypoints are then identified as the maxima and minima of the Difference of Gaussians at multiple pyramid levels. At each keypoint, descriptors are generated by extracting the image gradient magnitude and orientation (Lowe, 1999):

$$\text{Magnitude}(i,j) = \sqrt{\left(A_{ij} - A_{i+1,j}\right)^2 + \left(A_{ij} - A_{i,j+1}\right)^2}$$

$$\text{Orientation}(i,j) = atan2(A_{ij} - A_{i+1,j,}\ A_{i,j+1} - A_{ij})$$

Next, a brute-force matching algorithm was applied to the keypoints using the L2-norm distance metric (Li & Jain, 2009):

$$\text{L2-norm} = \sqrt{|x_1 + x_2|^2}, \text{ for keypoints } x_1, x_2$$

This algorithm finds matching keypoints between the two images by iterating through the list of keypoints in both images, and pairing the points with the minimum L2-norm distance.

Finally, if there are at least 4 keypoint matches, the homography matrix is computed using the PROSAC method and OpenCV's findHomography and warpPerspective functions. PROSAC stands for Progressive Sample Consensus and is an improvement on the traditional RANSAC method for detecting and discarding outlier points. The RANSAC algorithm begins with a random selection of keypoint matches, which are then used to calculate the 8 degrees of freedom $(h_{11}, h_{12}, h_{13}, \ldots, h_{32})$ in the homography matrix H (Derpanis, 2005):

$$H\begin{bmatrix}x\\y\\1\end{bmatrix}=\begin{bmatrix}h_{11}&h_{12}&h_{13}\\h_{21}&h_{22}&h_{23}\\h_{31}&h_{32}&h_{33}\end{bmatrix}\begin{bmatrix}x\\y\\1\end{bmatrix}$$

Once the homography matrix is obtained, RANSAC checks each keypoint pair to see if the difference between the left-hand keypoint and the right-hand keypoint (transformed by H) exceed an error threshold $\in = 3$. The keypoint pairs whose difference does not exceed $\in$ are counted as inliers. The RANSAC algorithm continues until it finds the 4 keypoint pairs that have a maximum number of inliers (Derpanis, 2005). PROSAC is similar to the RANSAC algorithm, but a semi-random selection of keypoints is chosen at each iteration, from progressively larger sets from the original set of matches (Chum & Matas, 2005). OpenCV's findHomography function calculates a perspective transformation matrix between the planes of the two images, employing the selected outlier detection method (PROSAC).

Next, the right-hand image is warped using OpenCV's warpPerspective function, which applies a perspective transformation to it so that it can match the plane of the left-hand image. The transformed points $dst(x, y)$ are calculated via the following formula, where src are the original points and H is the homography matrix:

$$dst(x,y) = src\left(\frac{H_{11}x + H_{12}y + H_{13}}{H_{31}x + H_{32}y + H_{33}}, \frac{H_{21}x + H_{22}y + H_{23}}{H_{31}x + H_{32}y + H_{33}}\right)$$

Finally, the warped right-hand image is stitched to the left-hand image by a simple overlay.

The object removal is based on the exemplar-based inpainting technique. The algorithm iteratively inpaints the target region with the most suitable pixel values that it finds in the source image. One iteration has the following steps:

1. Detect target region fill border, $mask * Laplacian\ filter$, where $mask$ is the target region
2. Update priorities
   a. Compute confidence values:
      $confidence_i = \sum_{j \in fill\ border}(patch_j / patch\ area_j)$
   b. Compute data term:
      $\sqrt{normal\ gradient\ in\ x\ direction^2 + normal\ gradient\ in\ y\ direction^2}$
   c. Compute priority with $confidence * data * fill\ border$, where $*$ means element-wise multiplication
3. Take the position with the highest priority $max(priority)$
4. Find appropriate pixel values for a patch such that $MSE = \sum \frac{(source_{xy} - target_{xy})}{N}$ is minimal
5. Set images from the source patch to the target patch

This algorithm repeats until all values in the target region are assigned with values from the source.

4 Results

The image stitching application combines and transforms 4 images from the same scene (see Figure 1) into a single panorama image (see Figure 2).

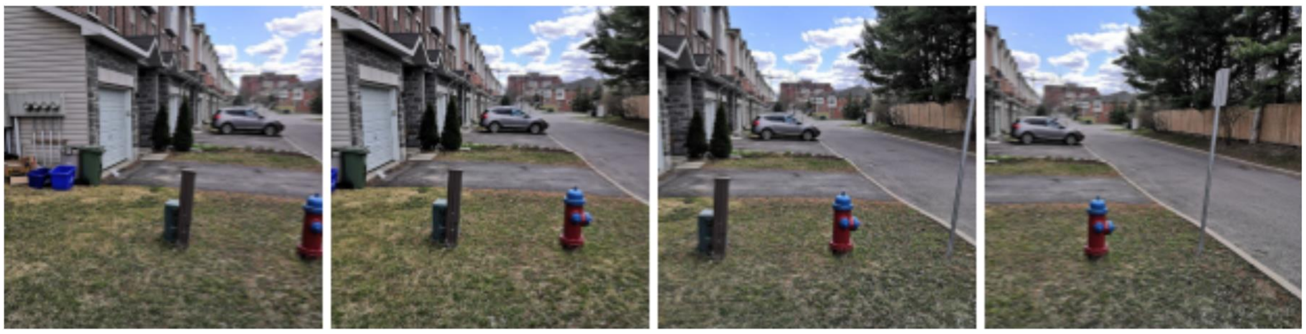

Figure 1. A grid of 4 images that were taken from different viewpoints of the same scene.

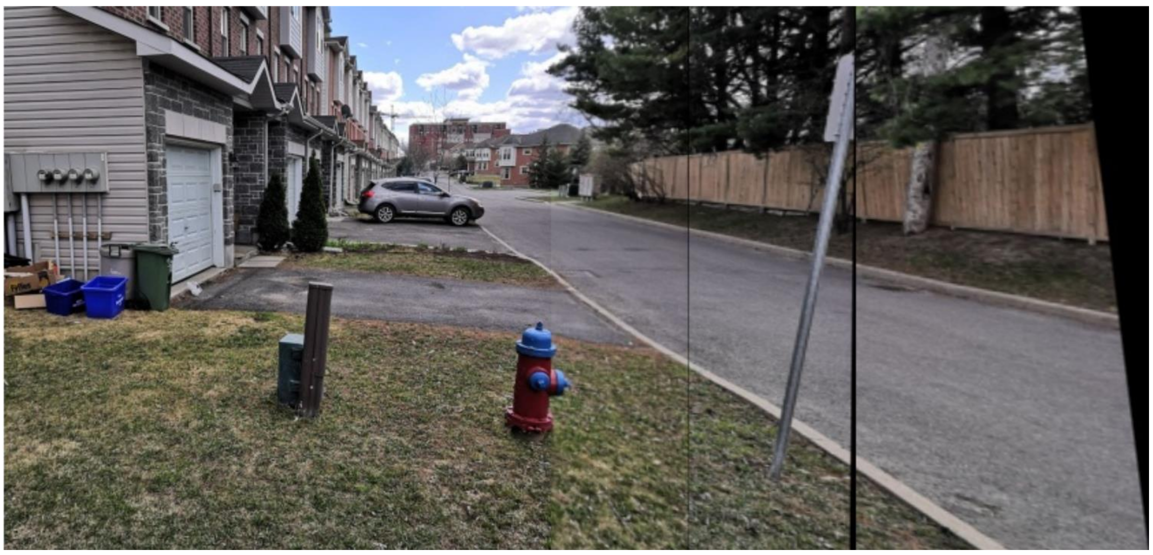

Figure 2. The panorama image result from the image stitching code, applied to the 4 images in Figure 1.

Note that the resulting panorama image has some anomalies (see Figure 3). The traffic sign on the side of the road appears to be transformed or aligned incorrectly. There are also two areas (circled in red) that have a black line between the left and right images, likely due to an error in the overlay method.

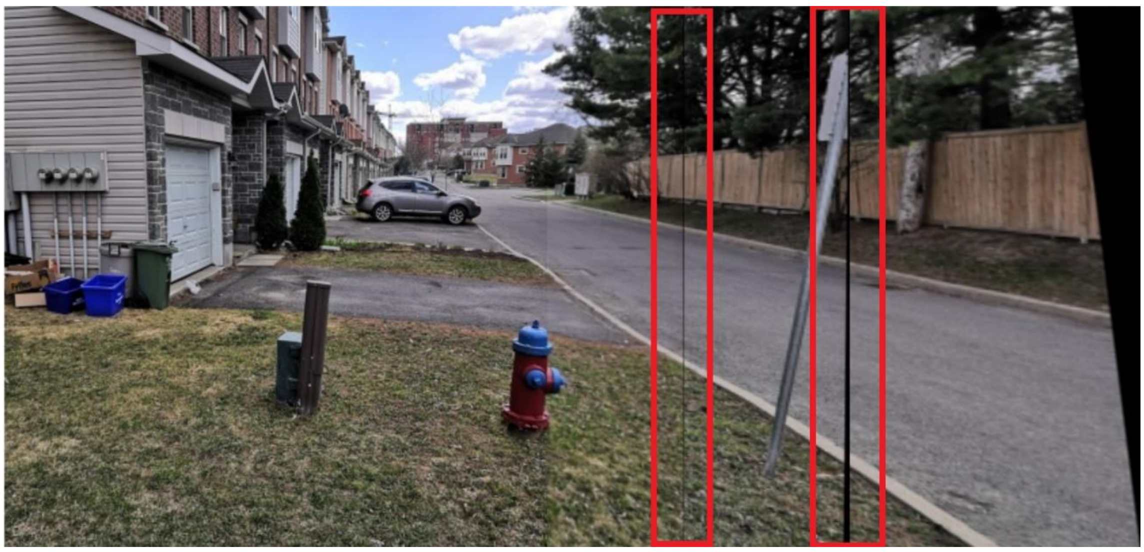
Figure 3. The panorama image result, with highlighted anomalies.

Figure 4 demonstrates intermediate results for object removal.

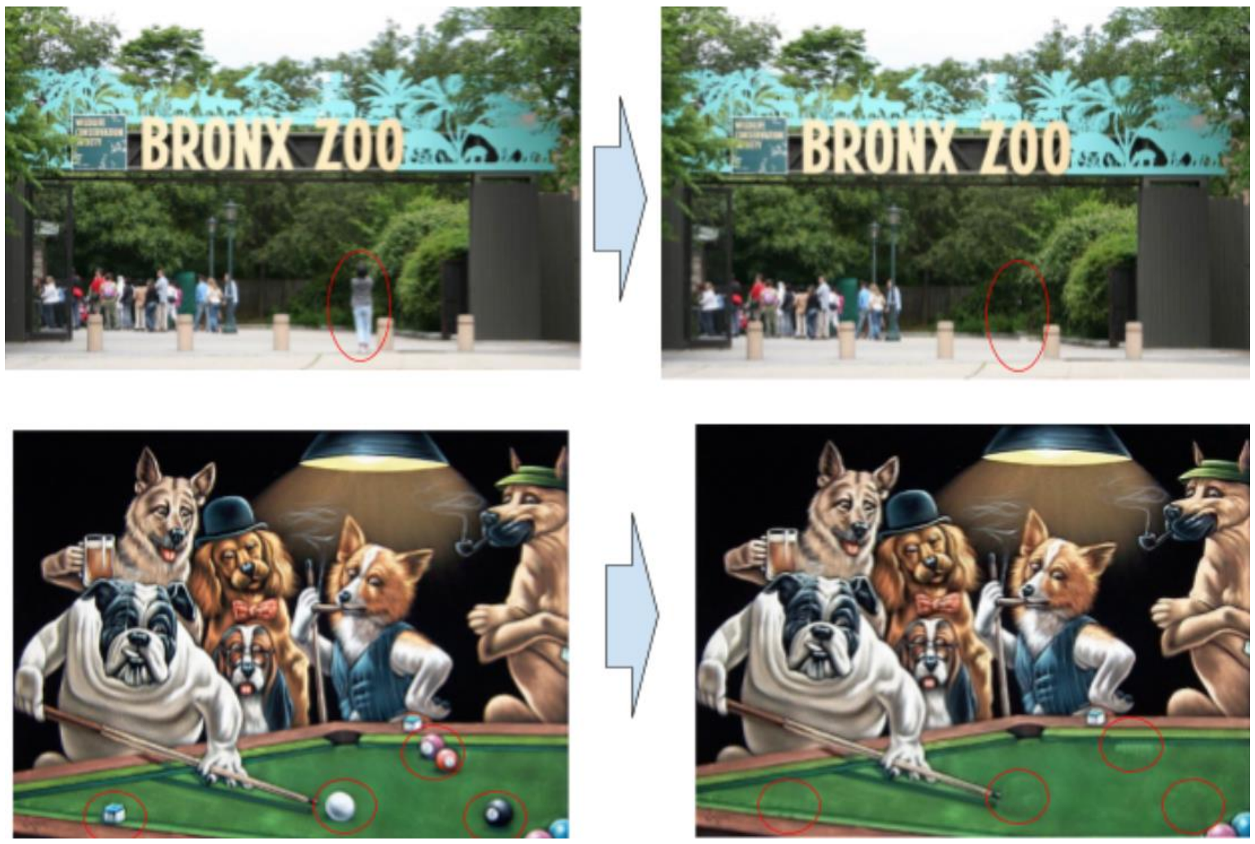

Figure 4. Object removal examples.

The object removal results depend on several factors that affect both the running time and inpainting quality. The factors are (1) object size relative to the scene and (2) patch size. First, large object removal significantly increases the running time with a low patch size but gives better results, while using a larger patch size has a faster runtime but worse results.

For future work, other overlay methods should be considered for stitching the final image, as well as different parameters, such as the error threshold, for the outlier and keypoint detection algorithms. Moreover, we introduced the patch window method to reduce time complexity, but it still takes significant time to remove an object from a panorama image. For example, the runtime to produce Figure 5 was five minutes, while Figure 6 had a runtime of about one hour. In addition, efficiency of the exemplar-based inpainting relies on a correctly generated mask, so in the future, a more advanced mask generation than square selection should be employed.

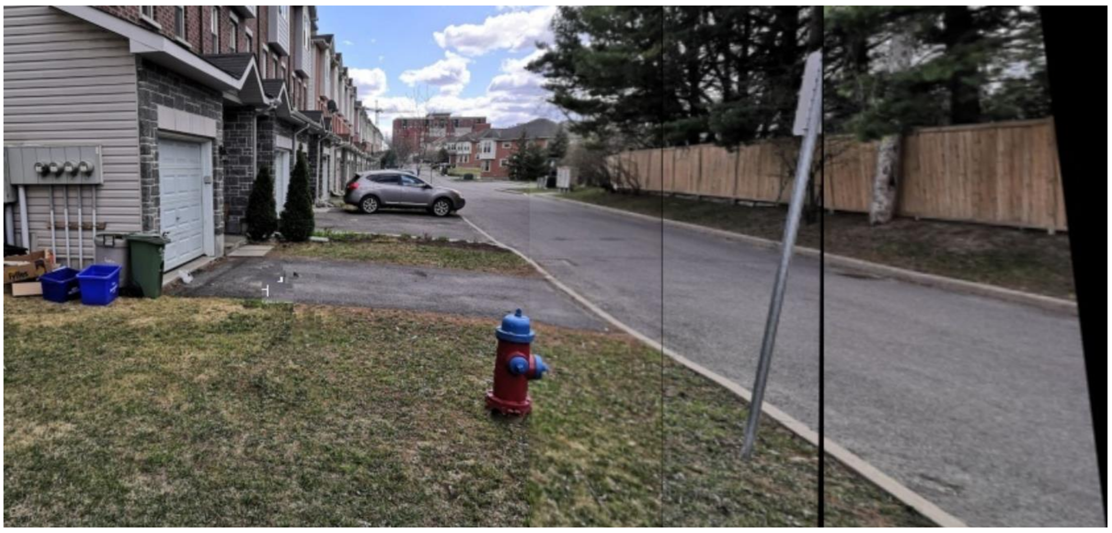

Figure 5. The panorama image with the electrical box removed.

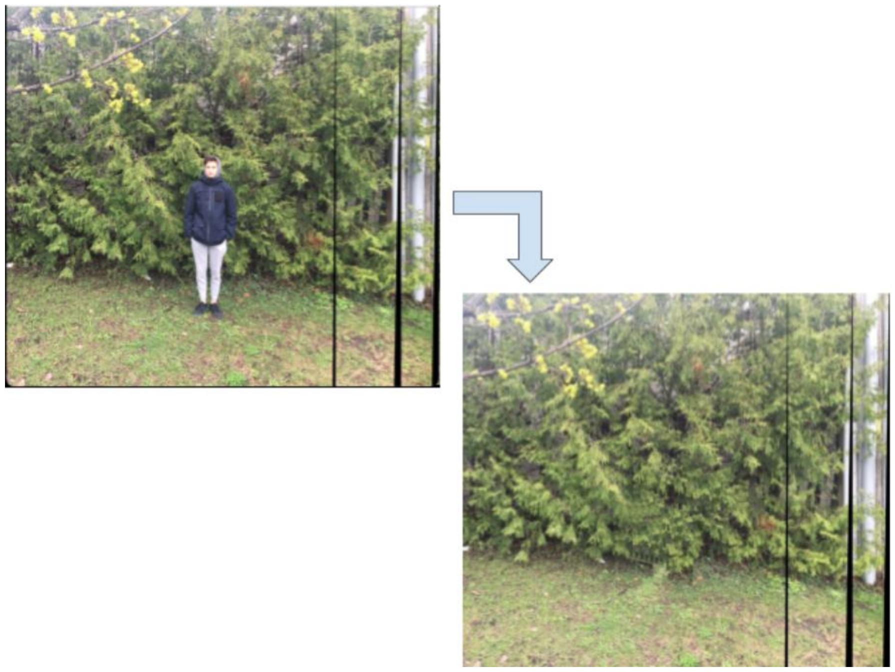

Figure 6. Object removal example.

5 List of Work

Erica	**Daniil**
Wrote code for detecting interest points and extracting descriptors	Wrote code for isophote-driven image-sampling process
Wrote code for matching descriptors between 2 images	Wrote code for region-filling
Wrote code for computing homography	Wrote code for computing patch priorities
Wrote code for aligning images using homography	Wrote code for propagating texture and structure information
Modified code to use 4+ source images	Modified code to handle parallax effect
Tested code for correct and seamless panorama creation	Tested code for correct and seamless object removal and patching
Prepared demonstration/presentation	Wrote code for a simple desktop app that employs both features
	Completed app testing

6 GitHub Page

Please visit https://github.com/d0rnkernsky/COMP4102-Project for the project's source code, written in Python.

7 References

Alcantarilla, P., Nuevo, J., Bartoli, A. (2013). Fast explicit diffusion for accelerated features in nonlinear scale spaces. Proceedings of the British Machine Vision Conference 2013. doi:10.5244/c.27.13

Brown, M., Szeliski, R., & Winder, S. (2005). Multi-Image matching USING MULTI-SCALE oriented patches. 2005 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'05). doi:10.1109/cvpr.2005.235

Chum, O., & Matas, J. (2005). Matching with Prosac — Progressive Sample Consensus. 2005 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'05). doi:10.1109/cvpr.2005.221

Criminisi, A., Pérez, P., & Toyama, K. (2004). Region filling and object removal by exemplar-based image inpainting. IEEE Transactions on Image Processing. https://doi.org/10.1109/TIP.2004.833105

Derpanis, K. (2005). Overview of the RANSAC Algorithm.

Fischler, M. A., & Bolles, R. C. (1981). Random sample consensus. Communications of the ACM, 24(6), 381-395. doi:10.1145/358669.358692

Introduction. (n.d.). Retrieved April 10, 2021, from https://docs.opencv.org/master/d9/dab/tutorial_homography.html

Li, S. Z., & Jain, A. K. (2009). *Encyclopedia of biometrics*. New York: Springer.

Lowe, D. (1999). Object recognition from local scale-invariant features. Proceedings of the Seventh IEEE International Conference on Computer Vision. doi:10.1109/iccv.1999.790410

Tareen, S. A., & Saleem, Z. (2018). A comparative analysis of SIFT, Surf, kaze, akaze, ORB, and brisk. 2018 International Conference on Computing, Mathematics and Engineering Technologies (iCoMET). doi:10.1109/icomet.2018.8346440

Torr, P., & Zisserman, A. (2000). MLESAC: A New robust estimator with application to estimating Image Geometry. Computer Vision and Image Understanding, 78(1), 138-156. doi:10.1006/cviu.1999.0832