Apply QuadTree On ASIFT Reduce Keypoints

1st Xiaoxin Zhou
dept. Computer Science
Ryerson University
Toronto, Canada
xiaoxin.zhou@ryerson.c

2nd Seit Sorra
dept. Computer Science
Ryerson University
Toronto, Canada
ssorra@ryerson.ca

***Abstract*—Feature matching is one of the most important tasks in the world of computer vision. It is critical in many applications such as 3D reconstruction, closed-loop control for Simultaneous localization and mapping (SLAM), camera calibration, and object detection. In this paper, we are comparing SIFT, Affine-SIFT, and Affine-SIFT QuadTree for feature matching processing time. This paper aims to reduce the number of keypoints and keep the distribution of the keypoints. The main contribution in this paper is to control the number of the Affine-SIFT keypoints and reduce its matching time.**

Source code for our experiments can be found at: https://github.com/zhoux121/843-Project

Keywords — component: SIFT, Affine-SIFT, feature matching.

I. INTRODUCTION

Feature matching has become an exciting area for researchers to study, and many papers have come out with many applications using feature matching. The task of feature matching involves matching feature descriptors of images. For some real-time applications such as SLAM, it is highly important to optimize the speed of feature matching while maintaining high-quality matching. Firstly, we will briefly introduce Scale Invariant Feature Transform (SIFT) and Affine-SIFT. SIFT involves five steps to calculate the SIFT keypoints and descriptors. 1) Scale-space extreme detection. 2) Keypoint localization 3) Orientation assignment. 4) Keypoint descriptor, and 5) keypoint matching [1]. SIFT is fully invariant with respect to only four parameters, namely zoom, rotation and translation. The Affine-SIFT is built on top of SIFT and provides two extra parameters: the angles defining the camera axis orientation [2]. This report compares the SIFT feature extraction and Affine-SIFT feature extraction based on their feature matching performance. Based on Guoshen Yu an Algorithm for Fully Affine Invariant Comparison [2], Affine-SIFT invariant to image rotation and can handle a transition tilt up to 36 degrees and higher, it has solved the descriptor tilt-matching problem. Hence, Affine-SIFT is a lot more robust and efficient than SIFT, and Affine-SIFT descriptors and keypoints format are the same as Lowe's SIFT. Our experiments showed that Affine-SIFT produced a lot more feature points with high accuracy. However, one drawback to that was that it took a significantly higher processing time. Provided this, we propose using a Quadtree that would reduce the number of keypoints generated by Affine-SIFT, which would improve processing time.

II. MATERIALS

For experiment, we are using two images from the Computer Vision Group website [3].

Figure 1: Sample images from fr1/xyz [3].
The hardware and system we are using is: 32 GB RAM, AMD Ryzen 7 1700X, GTX 1080 and Ubuntu 20.04.

III. INTRODUCTION TO SIFT AND ASIFT

A. SIFT: Scale Invariant Feature Transform (SIFT) is a common method used for feature detection on an image regardless of the scale of the image. The approach goes through four different stages to achieve the above [4]:

1) *Scale space extreme detection:*
For efficiency purposes, SIFT uses the difference of Gaussians to detect key-point locations in scale space.

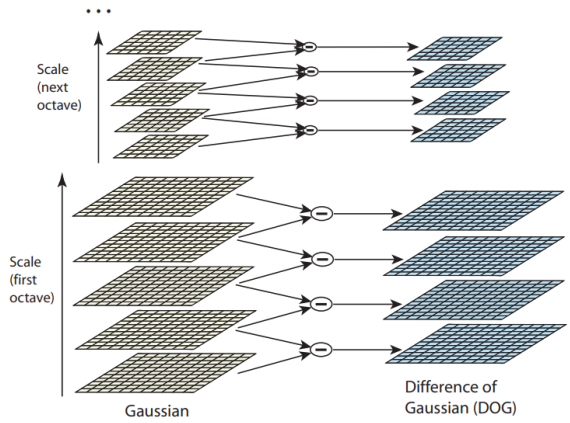

Figure 2: Difference of Gaussian per each octave.

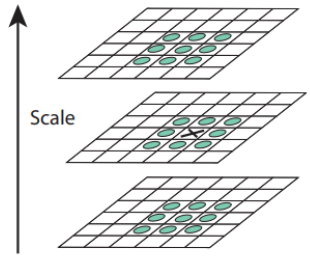

Figure 3: Local maxima/minima detection.

Local minima/maxima are then found by comparing each sample with the neighboring elements above and below as shown in Figure 3.

2) *Key-point localization:*

The process then uses a Taylor series expansion of scale-space centred around the sample point to get more accurate results.

3) *Orientation assignment:*

Then each feature is placed in one of 36 bins of 360-degree orientation.

4) *Keypoint descriptor*

Lastly, a keypoint descriptor is generated by looking at a 16x16 neighborhood of the selected keypoint and creating an 8-bin histogram of each 4x4 area ending with a 128-bin vector.

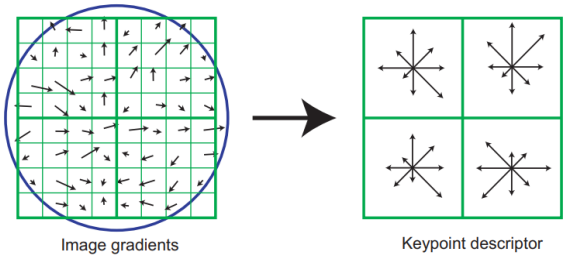

Figure 4: Keypoint descriptor.

Here is an example applying SIFT technique to detect features on an image using Open-CV:

Figure 5: SIFT feature detecting.

This technique is also used for object recognition by using a fast nearest neighbour algorithm and a Hough transform on a collection of features and is classified belonging to the same object within the image [6]. The performance of SIFT is invariant by different light conditions, object reflections, different view angles and 3D object structures [4].

ASIFT: AS SIFT algorithm, its performance is invariant to image scale and rotation. It provides a necessary condition for good matching. However, when the object tilts over 30 degrees, the matching rate of SIFT will drop. ASIFT switches the project camera model to an affine camera model and makes tilts. The Affine-SIFT solved the feature descriptor tilt matching based on feature extractions such as SIFT, SURF, and ORB.

The Projective Camera Model: U = SGAu.

A is a planer projective transform, G is an anti-aliasing gaussian filtering. S is the CCD sampling [3].

The Affine Camera Model: U = SGATu.

U is a digital image, and u is an ideal infinite resolution frontal view of the flat object. S is the standard sampling operator on a regular grid with mesh 1. G is a Gaussian convolution modeling the optical blur. T is a plane translation, and A is a planer protective map due to the camera motion or in simulating the camera motion [3].

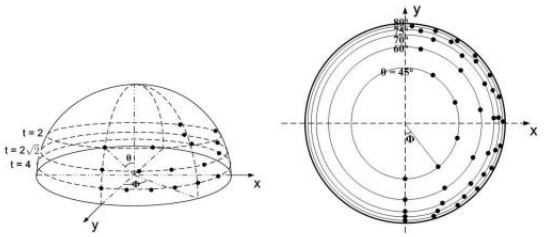

Figure 6: Sampling of parameters.

Assuming the camera is limited to a semi-circular ball, as shown in Figure 6, we can determine the spatial position of the camera through latitude and longitude. In the spatial position of the camera, ASIFT can simulate all possible replication distortions from images. After that, use a simulated image and apply the SIFT feature extraction.

IV. BRIEF INTRODUCTION TO QUADTREE

A QuadTree is a tree used to store the data on Two-dimensional efficiently. In QuadTree, each node has up to four children.

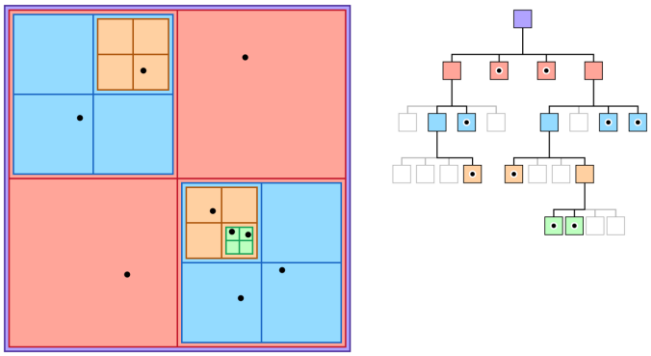

Figure 7: Spatial and logical arrangement of an example Quadtree [4].

Implementation of QuadTree is as such that it will continue splitting nodes into a subset of four children's nodes under some predetermined criteria. For this report, we use QuadTree to break up the image into subregions such that as long as the region contains three or more keypoints. To limit the growth of the tree, we limit the rank to up to five. Thereby, once the tree has reached a rank of 5, it will stop splitting.

Algorithm:

1) *Set Rectangle domain*
   Each domain will be split by the center point and divided into four squares.
2) *Call QuadTree*
   Use rectangle domain to split the region and insert point into the QuadTree.
   If the rank of quadtree is less than 6, keep splitting.
   Otherwise stop.
3) *After QuadTree's rank = 5*
   Choose one feature point for each subregion in the quadtree.
4) *Get descriptor for each keypoint in quadtree.*
5) *Return keypoint, descriptor.*

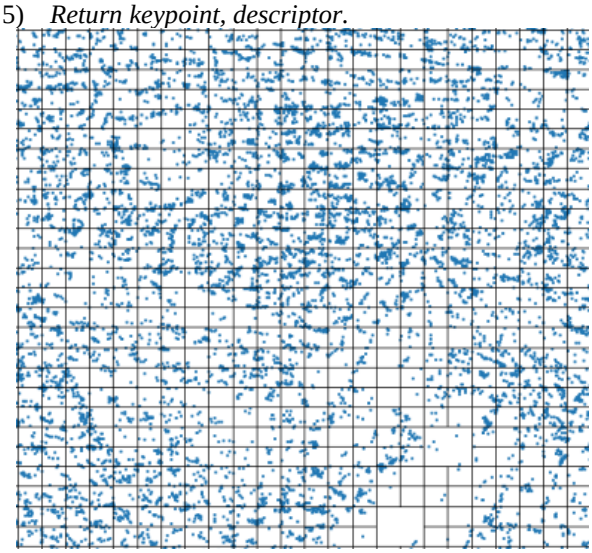

Figure 8: Before removing the keypoints from the QuadTree

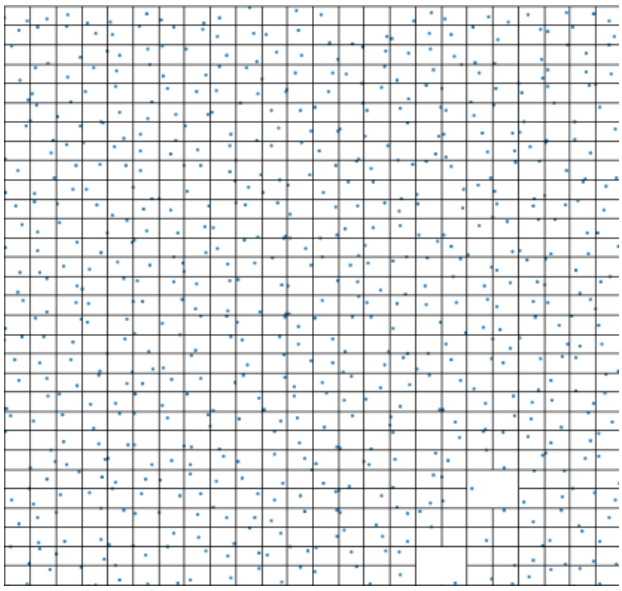

Figure 9: After removed the keypoints from the QuadTree

Example of QuadTree used to filter keypoints on an image:

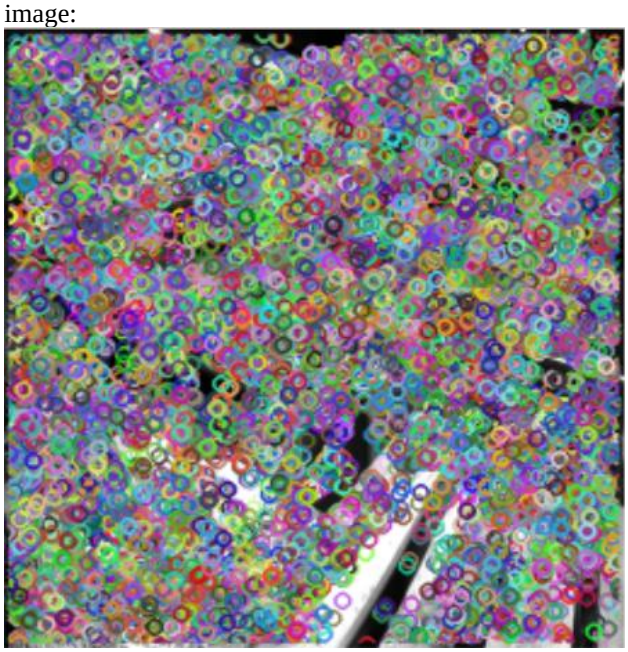

Figure 10: Before removing the keypoints from the image the number of keypoints shown in Figure 10 is 29,399.

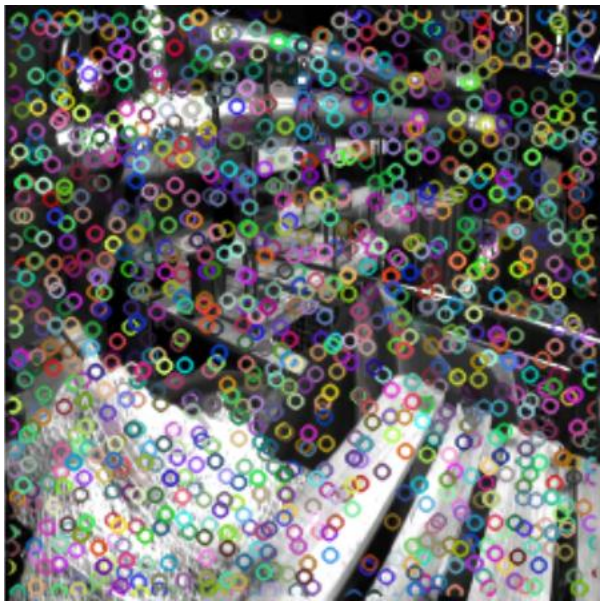

Figure 11: After removed the keypoints from the image The number of keypoints shown in Figure 11 is 939. The reducing rate for the keypoints is 96%.

V. EXPERIMENTAL RESULTS

For the feature matching, we are using Norm L1 from OpenCV [5]. The Norm L1 provides the fastest computing speed for the matching descriptor. The Norm L1 is the Manhattan distance for the sum of absolute value.

$$|\mathbf{x}|_1 = \sum_{r=1}^{n} |x_r|.$$

TABLE I. FOR THE FEATURE MATCHING, WE ARE USING NORM L1 FROM OPENCV [5]. THE NORM L1 PROVIDES THE FASTEST COMPUTING SPEED FOR THE MATCHING DESCRIPTOR. THE NORM L1 IS THE MANHATTAN DISTANCE FOR THE SUM OF ABSOLUTE VALUE.

	Image1	**Image2**	**Keypoints Matched**
SIFT	1985	1985	1106
ASIFT	29399	29941	16179
ASIFT(QuadTree)	939	847	393

TABLE II. NUMBER OF KEYPOINTS USING NORM L1 TO MATCH SIFT, ASIFT, AND ASIFT(QUADTREE) DESCRIPTORS.

	Matching Time (sec)
SIFT	0.35624599999999995
ASIFT	97.717684
ASIFT(QuadTree)	0.06117799999999818

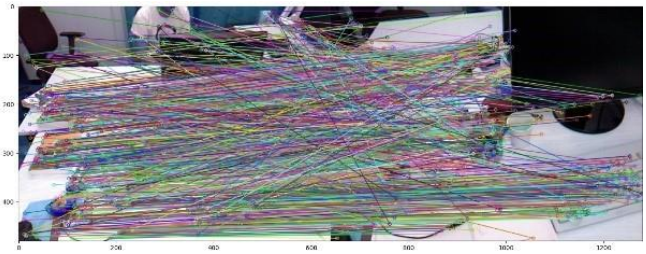

Figure 12: SIFT feature matching result.

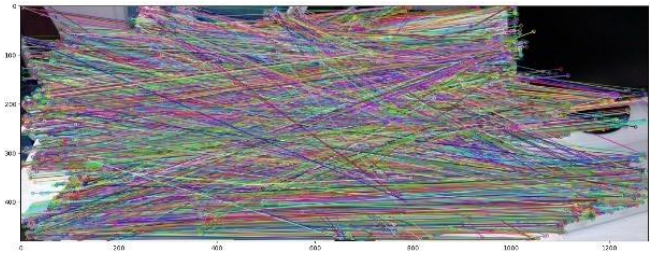

Figure 13: ASIFT feature matching result.

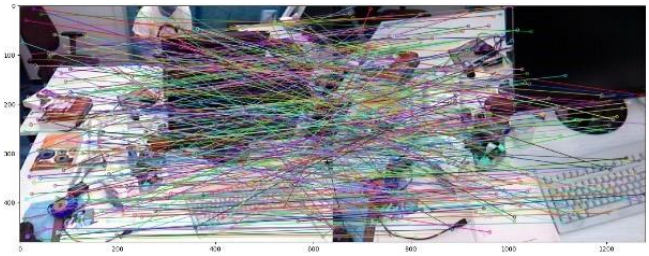

Figure 14: ASIFT(QuadTree) feature matching result.

A. *Feature Matching Accuracy Problem*

In our work we found out there are some accuracy problems for Norm L1. Hence, we choose the best ten feature matching results for a better visualization, but it still performs poorly.

Figure 15: SIFT feature matching result with ten best matching for Norm L1.

Figure 16: ASIFT feature matching result with ten best matching for Norm L1.

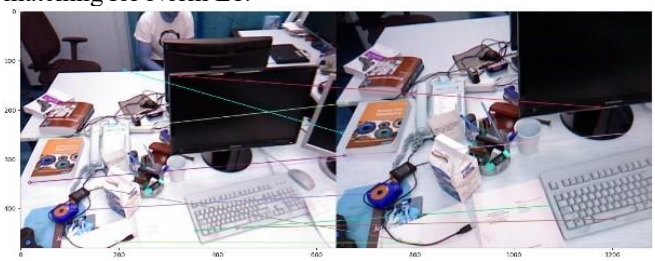

Figure 17: ASIFT(QuadTree) feature matching result with ten best matching for Norm L1.

Switch the Norm L1 to Norm L2, which is the Euclidean distance:

$$|\mathbf{x}| = \sqrt{\sum_{k=1}^{n} |x_k|^2},$$

TABLE III. PROCESSING TIMING FOR EACH FEATURE MATCHING USING NORM L1. PROCESSING TIMING FOR EACH FEATURE MATCHING USING NORM L1.

	Image1	Image2	Keypoints Matched
SIFT	1985	1985	1106
ASIFT	29399	29941	16179
ASIFT(QuadTree)	939	847	393

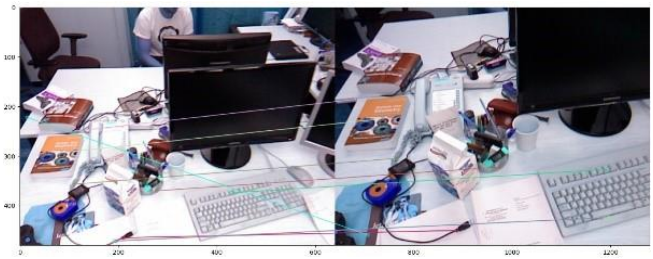

Figure 18: SIFT feature matching result with ten best matching for Norm L2.

Figure 19: ASIFT feature matching result with ten best matching for Norm L2.

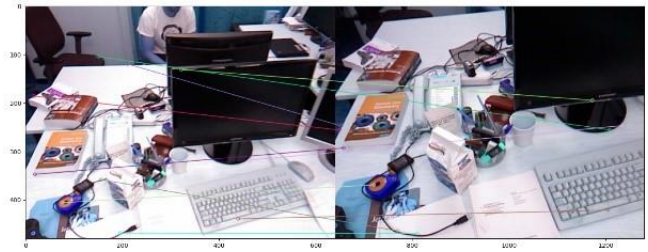

Figure 20: ASIFT(QuadTree) feature matching result with ten best matching for Norm L2.

Norm L2 did not yield improved results when applying the QuadTree, as shown in Figure 120. However, the SIFT and ASIFT provided a better result under Norm L2, as shown in Figures 18 and 19.

B. By using filter match and finding the homograph (RANSAC)

TABLE IV. PROCESSING TIMING FOR EACH FEATURE MATCHING

	Matching Time (sec)
ASIFT	99.24572700000002
ASIFT(QuadTree)	0.06417799999999829

Figure 21: ASIFT feature matching result with ten best matching for Norm L2.

Figure 22: ASIFT(QuadTree) feature matching result with ten best matching for filter matching. Based on the filter matching from SIFT paper [3], there still exist some wrong matching on the ASIFT and ASIFT(QuadTree). Therefore, filter matching does not aid our result.

C. By Using KnnMatch with distance less than 0.5.

TABLE V. PROCESSING TIMING FOR EACH FEATURE MATCHING USING KNNMATCH WITH DISTANCE LESS THAN 0.5.

	Matching Time (sec)
SIFT	0.36645700000000003
ASIFT	113.76952600000001
ASIFT(QuadTree)	0.07015900000000386

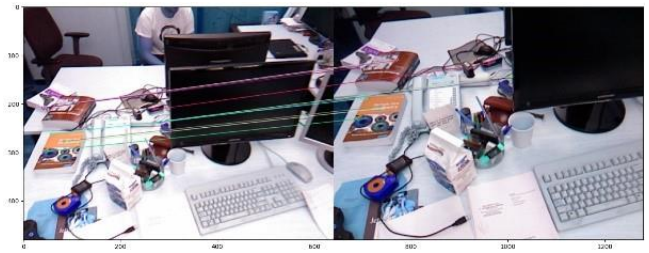

Figure 23: SIFT feature matching result with ten best matching for KnnMatch.

Figure 24: ASIFT feature matching result with ten best matching for KnnMatch.

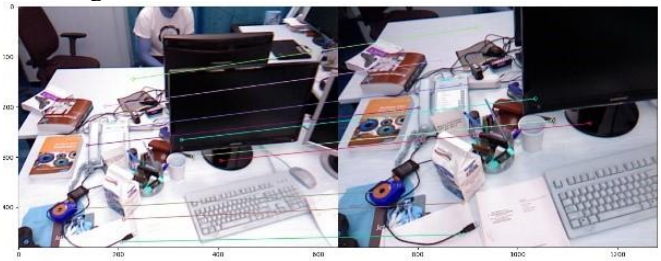

Figure 25: ASIFT(QuadTree) feature matching result with ten best matching for KnnMatch.

At the end of our experiment, we found out the KnnMatch from Open-CV implements the best matching.

VI. CONCLUSION

SIFT algorithm has proven to be very crucial in the advancement of feature matching. SIFT is a robust algorithm that offers feature matching regardless of the

image scale, zoom or translation. ASIFT built on top the SIFT algorithm by improving the number of features detected on an image while also performing well on different image angles and other affine transformations. The drawback of ASIFT was its performance, considering the huge amount of keypoints it detects. Some applications would be limited by the processing time of ASIFT, namely those working in real-time. In this paper, we developed a technique that would reduce the number of keypoints while still maintaining the quality of the feature points. Our initial experiments used the L1 and L2 norm for feature matching, which didn't produce outstanding results. The matching of the top ten highest quality keypoints was not being placed correctly. This report then used knnMatch, which in turn resulted in better matching of keypoints filtered by our QuadTree. As a result, we recommend going with knnMatch to get better matching results. Also, the work of this paper could be further improved by analyzing different set ups of the QuadTree, such as experimenting with a different rank limit or QuadTree max points to split a region. For our experiments, we use a rank limit of 5 and a maximum number of points set to 3.

ACKNOWLEDGEMENTS

We would like to acknowledge Dr. Guanghui (Richard) Wang for the supporting help throughout the development of this paper.

The workload of this report: Xiaoxin implemented QuadTree and applied QuadTree on ASIFT, experimented with filter matching and KnnMatch.
Sorra implemented SIFT and experimented with Norm L1 and Norm L2. Both of us maintain our project on GitHub and report writing. The workload is equal.

REFERENCES

[1] "OpenCV: Introduction to SIFT (Scale-Invariant Feature Transform)", Docs.opencv.org, 2021. [Online]. Available: https://docs.opencv.org/3.4/da/df5/tutorial_py_sift_intro.html. [Accessed: 28- Nov- 2021].

[2] G. Yu and J.-M. Morel, "ASIFT: An algorithm for fully affine invariant comparison," Image Processing On Line, vol. 1, pp. 11–38, 2011.

[3] "Informatik IX Chair of Computer Vision & Artificial Intelligence," Computer Vision Group - Dataset Download, 08-Mar-2016. [Online]. Available: https://vision.in.tum.de/data/datasets/rgbd-dataset/download. [Accessed: 30-Nov-2021].

[4] Mohsin, Zinah & Hasan, Reem Ibrahim & Ebis, Sif & Al-Wzwazy, Haider & Fadhel, Mohammed. (2019). Image Matching Performance Comparison Using SIFT, ASIFT &Moment Invariants. Journal of Engineering and Applied Sciences. 14. 10.36478/jeasci.2019.7656.7662.

[5] " Home," OpenCV, 16-Jun-2021. [Online]. Available: https://opencv.org/. [Accessed: 30-Nov-2021].

[6] Lowe, David G. "Distinctive image features from scale invariant key points" International journal of computer vision 60.2(2004):91-110