Implementation of the Improved sGLOH2 Descriptor in OpenCV

1st Justin Boyer
School of Science, Technology Engineering & Mathmatics
University Of Washington
Bothell, United States of America
jboyer4@uw.edu

2nd Frank Sossi
School of Science, Technology Engineering & Mathmatics
University Of Washington
Bothell, United States of America
fsossi@uw.edu

***Abstract*—This project aims to incorporate the sGLOH (scale-invariant Gradient Location and Orientation Histogram) descriptor into the OpenCV environment, employing C++ as our coding language. Furthermore, this project will implement and evaluate the sGLOH descriptor and its performance in image search compared to SIFT (Scale-Invariant Feature Transform) approach, which constitutes our benchmark for assessment.**

***Index Terms*—Keypoint matching, SIFT, sGLOH, RFDs, LIOP, MIOP, MROGH, CNN descriptors, rotation invariant descriptors, histogram binarization, and cascade matching.**

I. Introduction

In the paper Rethinking the sGLOH Descriptor [2], Fabio Bellavia and Carlo Colombo reassess and enhance the sGLOH descriptor, improving its robustness, speed, and dimension. The revised sGLOH incorporates more quantized rotations, resulting in better matching accuracy. The authors also introduce a novel fast-matching scheme and a new technique to convert the descriptor to binary representation. Experimental results indicate that the improved sGLOH descriptor outperforms the state-of-the-art in image matching and object recognition tasks [2]. The paper aims to enhance the sGLOH descriptor, a histogram-based key point descriptor that associates multiple quantized rotations of the key point patch without re-computation. The authors improve sGLOH's robustness, speed, and descriptor dimension, by introducing a revised version that embeds more quantized rotations for better matching. The project aims to implement the improved sGLOH descriptor and benchmark it against the SIFT descriptor.

II. Problem Statement

The current version of OpenCV 4 does not have an implementation for the sGLOH 2 descriptor [8]. This project is intended to provide an implementation of this novel descriptor to allow it to be used in the OpenCV library to expand the capabilities of this open-source computer vision resource for future developers.

III. Objectives

The principle objective of this project is to successfully implement the sGLOH2 descriptor so that it can integrate with OpenCV. Also, we plan to benchmark the implementation to get an initial comparison to the standard SIFT descriptor to inform further development [4].

IV. Methodology

A. Implement the sGLOH descriptor

1) **Compute the gradient orientation and magnitude at each pixel in the key-point patch.**
   - This step involves calculating the gradient of the image at each pixel in the patch, which gives us both the magnitude and orientation of the gradient vector.
   - The gradient orientation is typically divided into a small number of discrete bins (e.g., 8 or 16) to reduce the dimension of the descriptor.
2) **Divide the patch into sub-regions and compute a histogram of gradient orientations and magnitudes for each sub-region.**
   - The key point patch is divided into a set of overlapping sub-regions, each typically rectangular or circular.
   - For each sub-region, we need to compute a histogram of gradient orientations and magnitudes by accumulating the gradients of all pixels within that sub-region.
3) **Apply a rotation-invariant transformation to each sub-region histogram to obtain a set of rotation-invariant descriptors.**
   - To make our descriptor invariant to rotations, we need to apply a transformation that maps similar histograms with different orientations to similar feature vectors.
   - In order to do this we will use circular harmonic functions (CHFs) as shown in the paper [3], which are complex-valued functions invariant to rotations about their center point. By applying CHFs to our histograms, we can obtain a set of rotation-invariant descriptors for each sub-region.
4) **Concatenate all rotation-invariant descriptors to form the final sGLOH descriptor.**
   - To produce the final descriptor, we concatenate all of our rotation-invariant descriptors into a single feature vector representing our key point.

5) **Calculate sGLOH2 descriptor.**
   - For each patch, perform a pi/m rotation where m is the number of orientation bins used in step 1.
   - Compute the sGLOH descriptor for the rotated patch repeating steps 1-4
   - Concatenate the original sGLOH descriptor and the sGLOH descriptor from the rotated patch

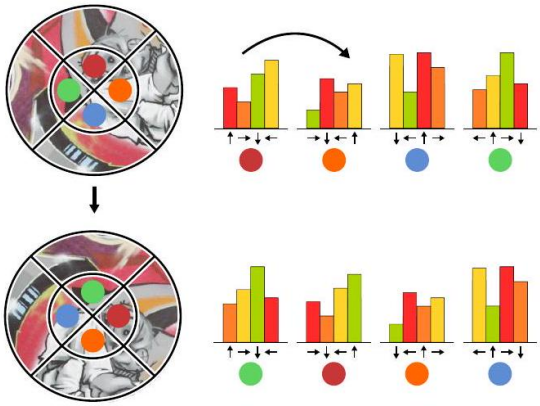

Fig. 1. SGLOH descriptor [2]

B. Determine optimum key point selection method

After the descriptor is tested and valid in controlled tests, we will then implement the key point locating functions based on one or more of the following methods:

1) **SIFT (Scale-Invariant Feature Transform [4]):** As one of the most favored key point detection algorithms, SIFT's robustness against scale and rotation changes makes it effective for a wide array of applications. It uncovers key points by identifying local peaks of the Difference of Gaussians (DoG) function at differing scales.
2) **SURF (Speeded-Up Robust Features):** A swifter alternative to SIFT, SURF employs an approximation of the Hessian matrix for key point detection. As a result, it retains robustness to scale and rotation changes while generally outpacing SIFT.
3) **FAST (Features from Accelerated Segment Test) [6]:** FAST, a real-time corner detection algorithm, discerns key points by examining the intensity of a compact circular region around a pixel. Its speed makes it particularly fitting for real-time applications.
4) **ORB (Oriented FAST and Rotated BRIEF) [5]:** ORB fuses the FAST key point detector with the BRIEF (Binary Robust Independent Elementary Features) descriptor. Offering scale and rotation invariance, ORB surpasses SIFT and SURF in speed, rendering it apt for real-time applications and large-scale image search tasks.
5) **AKAZE (Accelerated-KAZE): [8]** A key point detection algorithm with scale and rotation invariance, AKAZE unites the KAZE detector with a swifter processing strategy. It performs non-linear scale-space filtering and relies on a tweaked version of the Hessian matrix for key point detection.
6) **BRISK (Binary Robust Invariant Scalable Key points): [8]** BRISK merges a corner detection method with a scale-space approach, proving robust against scale, rotation, and illumination shifts. Moreover, it is well-suited for real-time applications, thanks to its speed.

V. Development

The primary focus of our project was the implementation of the SGLOH2 descriptor. This task was highly integrated and complex, necessitating a collaborative approach to coding. As a result, we adopted a pair programming methodology, which allowed us to maintain a cohesive understanding of the code and ensure that both team members were on the same page.

A significant portion of our planning phase was dedicated to understanding the mathematics behind the SGLOH descriptor. This was a challenging task, as the mathematical concepts involved were complex. However, we were able to gain a solid understanding of the underlying principles, which greatly facilitated our subsequent coding efforts.

Our initial implementation of the SGLOH2 descriptor was rather complicated and only partially successful. Therefore, we decided to revisit some of our base assumptions and simplify our approach. Instead of trying to implement the mathematics directly, we focused on capturing the intent of the descriptor in its simplest terms. To do this, we break down the primary purpose of the descriptor, which was to get a directional gradient histogram for the regions shown in Figure [1] and to combine them into a vector descriptor such that by rotating each 16 value block by four positions we could simulate the rotation of the image patch. This resulted in a cleaner and more efficient implementation that performed considerably better than our initial attempts.

The OpenCV library greatly simplified our efforts. Its robust set of features and functions allowed us to focus on the core logic of our implementation rather than getting bogged down in low-level details. The cleaner implementation enabled us to make significant progress and rapidly iterate on our design.

One of the main challenges we faced was key point selection. Our original implementation used the SIFT detector based on a gradient of intensity values. However, this approach yielded only partial results, even after we had refined the inner workings of the SGLOH2 descriptor. We tried various key point detectors like AKAZE and SURF, but unfortunately, none of them provided us with the desired results.

Our breakthrough came when we used the ORB (Oriented FAST and Rotated BRIEF) detector [5]. ORB is a fast and robust local feature detector based on the FAST key point detector and the BRIEF descriptor. It is computationally efficient and performs well on several benchmarks.

The ORB algorithm utilizes the FAST method to detect important points by identifying corners in gray scale images.

Additionally, the intensity centroid method is employed to add a rotation component to these key points. This approach significantly enhances the effectiveness of sGLOH2 for image searches without the need for supplementary techniques like GMM, KNN, or machine learning.

We found that the ORB detector worked differently for key point selection and represented an improvement over the method described in the original paper, which was based on a Gaussian difference. This approach had hampered the performance of the descriptor, and switching to ORB resulted in a significant improvement in our results. We hypothesized that the corner selection resulted in descriptors better at separating images than the descriptors resulting from other key point selection methods and is an area for further research for understanding.

VI. Implementation of the sGLOH Descriptor: A Detailed Analysis

Implementing the sGLOH descriptor, as seen in the provided C++ code, involves several key steps and methods contributing to its functionality.

A. The sGLOH2 Class

The **sGLOH2** class is the main class in this implementation. This class has several methods explained below and one private variable `region_masks` that stores the pre-computed masks for each region based on the defined size of the patch.

B. sGLOH2 Class Constructor

The **sGLOH2** class constructor initializes the region_masks vector, which stores masks for each ring and sector of a patch. The constructor creates these masks by iterating over each pixel in the patch, calculating its angle and distance from the center, and checking if it falls within the current ring and sector. If it does, the corresponding value in the mask is set to 1. This pre-computation of masks is done in the constructor so that it is only done once saving computation time.

C. The Compute Method

The **compute** method is the central function within the class. It inputs an image and a set of key points and computes the sGLOH descriptor for each key point. The method is implemented as follows:

1) An ORB detector is created using the OpenCV library. This detector is then used to detect key points in the input image [5].
2) The method then removes key points that are too close to the edge of the image. This is done to avoid issues with patch extraction for these key points.
3) The method then enters a parallel loop that iterates over all the key points. For each key point, it does the following:
    a) A patch of the image around the key point is extracted. The PATCH_SIZE constant determines the size of the patch.
    b) A copy of the patch is made and rotated to align with the orientation of the key point. This uses a rotation matrix created with the OpenCV function `getRotationMatrix2D`.
    c) The sGLOH descriptor is computed for the rotated patch using the `compute_sGLOH` method.
    d) The descriptor is then normalized using the OpenCV function `normalize`.
    e) The normalized descriptor is added to a local list of descriptors.
4) After the parallel loop, the local lists of descriptors are concatenated into a global list. This is done in a thread-safe manner using a mutex.
5) Finally, the global list of descriptors is returned.

This method effectively computes the sGLOH descriptor for each key point in the input image, considering each key point's orientation. In addition, using a parallel loop allows the method to compute descriptors for multiple key points simultaneously, which can significantly speed up the process for images with many critical points.

D. The Compute_sGLOH Method

This method takes a patch of an image as input and computes the sGLOH descriptor for the patch at multiple orientations. The method is implemented as follows:

1) The method first computes the sGLOH descriptor for the original patch using the `compute_sGLOH_single` method.
2) It then computes the center of the patch.
3) It sets the rotation angle step to 90 degrees.
4) It then enters a loop that iterates three times. For each iteration, it does the following:
    a) It computes the rotation matrix for the current angle using the OpenCV function `getRotationMatrix2D`.
    b) It rotates the patch using the OpenCV function `warpAffine`.
    c) It computes the sGLOH descriptor for the rotated patch using the `compute_sGLOH_single` method.
    d) It concatenates the descriptor for the rotated patch with the original descriptor.
5) After the loop, the descriptor is normalized using the OpenCV function `normalize`.
6) Finally, the normalized descriptor is returned.

E. The Compute_sGLOH_single Method

This method takes a patch of an image as input and computes the sGLOH descriptor for the patch. The method is implemented as follows:

1) The method first initializes the descriptor as a zero matrix of size 1 by M * N * M.
2) It then enters a nested loop that iterates over all sectors in all rings. For each sector in each ring, it does the following:

a) It retrieves the pre-computed mask for the region.
b) It computes the histogram for the region using the `computeHistogram` method.
c) It places the histogram in the descriptor at the correct position.

3) After the nested loop, the descriptor is normalized using the OpenCV function `normalize`.
4) Finally, the normalized descriptor is returned.

F. The ComputeHistogram Method

This method takes a patch of an image, a and the pre-computed mask, as input and computes a histogram of gradient orientations for the patch. The method is implemented as follows:

1) The method first computes the patch gradient in the x and y directions using the OpenCV function `Sobel`. This results in two matrices, `grad_x` and `grad_y`, which hold the x and y components of the gradient at each pixel in the patch.
2) The method then computes the magnitude and orientation of the gradient at each pixel using the OpenCV function `cartToPolar`. This results in two matrices, `magnitude` and `orientation`, which hold the magnitude and orientation of the gradient at each pixel, respectively.
3) The orientation is then converted from radians to degrees by multiplying it with the factor 180.0/CV_PI.
4) A histogram of size 'M' is initialized with zeros using the OpenCV function `Mat::zeros`.
5) The method then enters a loop that iterates over all the pixels in the orientation matrix. For each pixel, it does the following:
   a) If the corresponding pixel in the mask is non-zero, it calculates the bin number for the orientation at the pixel by rounding the orientation to the nearest integer after scaling it by the factor 'M/360.0'. The bin number is then modulo 'M' to ensure it falls within the histogram range.
   b) The magnitude at the pixel is then added to the corresponding bin in the histogram.
6) After the loop, the histogram is reshaped to a single-row matrix using the OpenCV function `reshape`.
7) Finally, the reshaped histogram is returned.

This method effectively computes a histogram of gradient orientations for a patch of an image, taking into account the mask.

G. The Distance Method

This method computes the minimum distance between two sGLOH descriptors. The method is implemented as follows:

1) The method first initializes the minimum distance (float) to the maximum possible value and the best rotation to 0.
2) It then enters a loop that iterates M times. For each iteration, it does the following:
   a) It performs a cyclic shift on the second descriptor.
   b) It computes the L1 norm between the first and the shifted second descriptors.
   c) If the computed distance is smaller than the current minimum distance, it updates the minimum distance and the best rotation.
3) After the loop, it adds a global rotation constraint. If the absolute difference between the best rotation and M/2 is greater than a rotation threshold, it returns the maximum possible value.
4) Finally, it returns the minimum distance. The lower the distance, the better the match.

H. The CyclicShift Method

This method performs a cyclic shift on a given sGLOH descriptor. The method is implemented as follows:

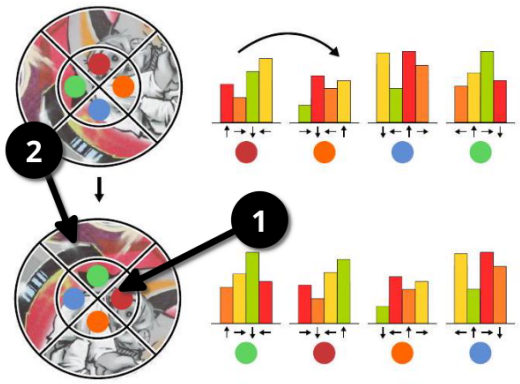

Fig. 2. SGLOH descriptor [2]

1) The method first clones the input descriptor to create a new matrix that will be modified and returned.
2) It then computes the size of the blocks. Each block corresponds to a ring of the image #1 and #2 in Figure [2].
3) It enters a loop that iterates once. For each iteration, it does the following:
   a) It extracts the i-th block from the descriptor.
   b) It converts the block to a vector to facilitate the rotation operation.
   c) It rotates the block vector by k (in this case 4) positions.
   d) It converts the rotated vector back to a Mat and replaces the i-th block in the shifted descriptor.
4) Finally, it returns the descriptor with cyclically shifted blocks.

VII. Evaluation

Algorithm performance was evaluated in 3 main categories where sGlOH and SIFT were compared on their ability to identify matches from an image library.

First, we used a synthetic rotation test modeled after the tests performed in "Rethinking the sGLOH descriptor" [2]. In this case, a successful match would be copies of the image that had been rotated or flipped from a larger set of images. sGLOH is designed to be more resilient to rotations and does succeed here. Three of the top five results were true matches, while SIFT could only successfully recall one rotated copy and no flipped copies.

Second, we used a set of images of rotated items from the Amsterdam Library of Object Images (ALOI) [7]. This image set consists of one thousand objects taken in a controlled environment where each object was rotated on a table allowing us to test the rotational invariance of the descriptor relative to SIFT.

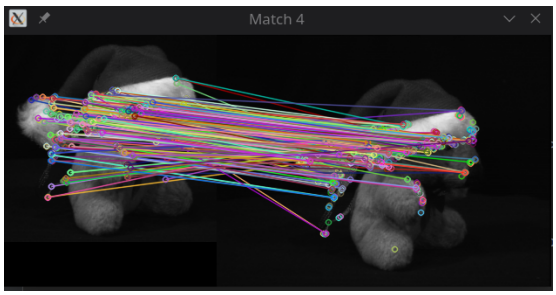

Fig. 3. Result with rotated bear [7]

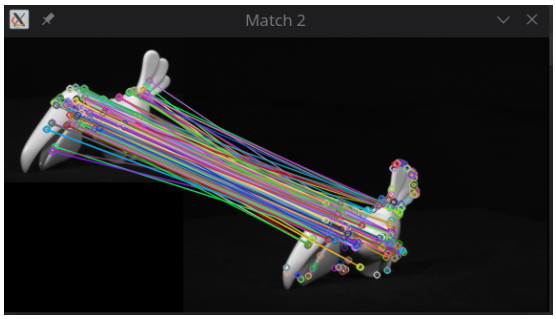

Fig. 4. Result With rotated toucan [7]

As expected, these tests are where sGLOH2 descriptors really shine. Using the teddy bear (Fig 3) as the query resulted in the top 5 matching images from the image library being rotated teddy bears. Conversely, SIFT found only one of the teddy bear images in the library.

Similarly, other images from this library also performed well. In all 4 tests from this set (teddy bear, jar, toucan, brush) the sGLOH2 descriptor found true matches for 4-5 of the top query results. On the other hand, SIFT typically found only the exact match or an image with a small degree of rotation (i.e., 5 degrees).

Found image	Matches
Tucan 245	362
Tucan 250	350
Tucan 255	327
Bear 65	326
Tucan 260	323

TABLE I
Matches found

Image Name	**sGLOH2 Correct Matches**	**SIFT Correct Matches**
Beagle	1/10	0/10
Bear	8/10	1/10
Bombay	5/10	2/10
Brush	4/10	0/10
Cabin	3/10	2/10
Jar	6/10	1/10
Toucan	5/10	1/10
Waterfall	1/10	0/10

TABLE II
Image Descriptor Comparison Results

To match images, we utilized sGLOH2 and SIFT. The classes `ImageComparator_sGLOH` and `ImageComaparator_Sift` used brute force matching techniques to compare the two descriptors' usefulness in image search. The "Rethinking the sGLOH descriptor" findings indicate that sGLOH can effectively perform this task solely through key points [2].

We used query images of a dog, two cats, and a waterfall to test image matching. We searched against an image library, including pictures of the same animal (or waterfall). The library also contained images of other cats, dogs, waterfalls, and objects.

As expected, SIFT did not perform well in this task. It typically failed to find matches within our thresholds. On the other hand, sGLOH2 did frequently manage to find accurate matches, as shown in Table II. For example, when testing against the black cat "Bombay," the descriptors could be used to find 5 of the 11 images of that cat in its top 10 matches. This is impressive, considering the images' wide variety of poses and backgrounds.

It could have been more successful with the Beagle and waterfall queries, but it only found the closest match.

On average, our implementation of sGLOH2 takes about 4x as long as SIFT. However, this does not consider any run time improvements from "Rethinking the sGLOH descriptor," which could likely be leveraged to reduce that gap [2].

VIII. Follow up Research

This project has opened up several avenues for further research and development. Here are some potential extensions:

1) **Fast Matching Algorithm:** The implementation of the "Fast matching Algorithm" described in section 4.2 of reference [2] could potentially improve the performance of the descriptor. This algorithm could provide a more efficient way to match images.
2) **Filtering Distances:** Another area of interest is the real-time filtering of partially computed distances. This could lead to more efficient computations and faster processing times.
3) **Binarization of Descriptors:** Binarizing the sGLOW descriptors based on their histograms, as described in section 4.3 of reference [2], could provide a more compact and efficient representation of the descriptors.
4) **Generalization:** The techniques used in this project could potentially be generalized to other histogram-based descriptors. This could broaden the applicability of the methods and provide insights into their performance across different types of descriptors.
5) **Training Learning Models:** The sGLOH2 descriptor could potentially be used for training machine learning models. This could open up new possibilities for automated image recognition and classification tasks.

IX. Conclusion

Our project has demonstrated the effectiveness of the sGLOH2 descriptors in matching key points from images. Using ORB for generating key points proved to be the most effective in our testing, outperforming other methods such as SIFT, AKAZE, and SURF.

The sGLOH2 descriptors, while taking approximately four times as long as SIFT for key point generation and matching, showed superior robustness to rotations. This was particularly evident in our tests involving images where the focus item was rotated. In addition, the sGLOH2 descriptors consistently matched key points between these images, demonstrating their potential for applications requiring rotation-invariant image matching.

However, it's important to note that while our implementation of sGLOH2 has shown promising results, there are still areas for improvement and further research. Nevertheless, the potential for enhancing the usability of this descriptor is significant, and other iterations and experimentation with the sGLOH2 implementation could yield even more robust results.

Future research could explore adding sGO or sCO algorithms to the sGLOH2 descriptor calculation. Furthermore, implementing the "Fast matching Algorithm" described in section 4.2 of our reference [2] could potentially improve the performance of the descriptor. Other areas for exploration include real-time filtering of partially computed distances and the binarization of the sGLOH descriptors based on their histograms, as described in section 4.3 of our reference [2].

In conclusion, the sGLOH2 descriptor implemented in this project shows promise for image processing, particularly in scenarios requiring robust rotation-invariant image matching. With further research and optimization, this descriptor would make a valuable addition to the OpenCV Library.

X. Related Projects

During the initial research into this topic, the following projects related to this project were located.

1) We used the sGLOH_demo source code provided by Professor Bellavia to implement his work.
2) Implementing sGLOH in OpenCV Ardalan Ahanchi, Michael Waite This project partially implemented the sGLOH descriptor but not the sGLOH 2 described in principle reference for this project [2].

References

[1] Bellavia, Fabio, Domenico Tegolo, and Emanuele Trucco. "Improving SIFT-based descriptors stability to rotations." 2010 20th International Conference on Pattern Recognition. IEEE, 2010.

[2] Bellavia, Fabio, and Carlo Colombo. "Rethinking the sGLOH descriptor." IEEE Transactions on Pattern Analysis and Machine Intelligence 40.4 (2017): 931-944.

[3] Colonnese, Stefania, et al. "Fast image interpolation using circular harmonic functions." 2010 2nd European Workshop on Visual Information Processing (EUVIP). IEEE, 2010.

[4] Lowe, David G. "Distinctive image features from scale-invariant key points." International journal of computer vision 60 (2004): 91-110.

[5] Rublee, Ethan, et al. "ORB: An efficient alternative to SIFT or SURF." 2011 International conference on computer vision. Ieee, 2011.

[6] Rosten, Edward, and Tom Drummond. "Machine learning for high-speed corner detection." Computer Vision–ECCV 2006: 9th European Conference on Computer Vision, Graz, Austria, May 7-13, 2006. Proceedings, Part I 9. Springer Berlin Heidelberg, 2006.

[7] ALOI Grey Image Set. ALOI, University of Amsterdam, 27 May 2023, https://aloi.science.uva.nl

[8] OpenCV Documentation. OpenCV, version 4.7.0, OpenCV, https://docs.opencv.org/4.7.0/