COIFv6: Concentric Oval Intensity Features Version 6

Daniel Puckowski
May 2024

Abstract

In this paper, I present an update to the novel quasi-rotation invariant interest point descriptor COIF (Concentric Oval Intensity Features). The descriptor is straightforward to implement and feature matching can be time efficient. COIF may be used to detect rotated images and may be used for image stitching in panorama applications. COIF demonstrates the feasibility of using luminance histograms for feature matching.

Figure 1. Shape of a COIF descriptor.

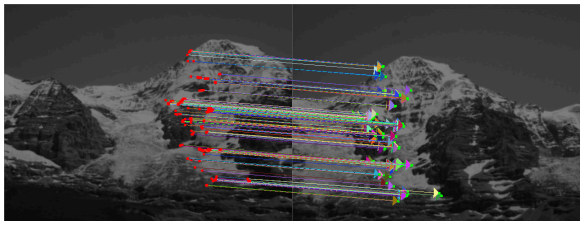

Figure 2. Typical matching result using COIF on real-world images with default settings. Many matches are detected with few incorrect matches.

1. Introduction

Feature matching—the process of finding matching or similar regions between two images of the same scene or object—is a common computer vision task. Feature matching may be a step for object recognition, a step for image stitching, and a step for pattern tracking. Keypoint detectors and descriptors such as SIFT, SURF, and ORB are commonly and successfully used for this task [4] [5] [6]. But the ORB binary descriptor, for example, is not easy to implement. To implement ORB, one needs to implement procedures to produce FAST features filtered by the Harris measure for a scale pyramid of an image, compute the orientation of a FAST feature by determining the intensity centroid, compute BRIEF descriptors for image patches from a set of binary intensity tests, perform a greedy search for a set of uncorrelated tests with means near 0.5 to generate rBRIEF descriptors, then implement Locality Sensitive Hashing (LSH) to perform a nearest neighbor search [4]. By contrast, COIF is meant to be easy to implement and is meant to be easy to optimize so that it is time efficient and so that feature matching may be performed in real time without the need for GPU acceleration on low-power devices.

2. Related work

Keypoints

The Moravec corner detection algorithm, introduced by Hans P. Moravec in 1977, is one of the earliest corner detection algorithms [1]. The Moravec algorithm defines a corner as a point with low self-similarity.

Descriptors

Traditionally, image retrieval is based on the representation of the image content through features thought to be relevant for the image description. Luminance, color, edge strength, and textural features are commonly used. Vertan and Boujemaa use

fuzzy color histograms and their corresponding fuzzy distances for the retrieval of color images within various databases [2]. Vertan and Boujemaa use fuzzy distances due to the imprecision of the pixel color values.

Tola, Lepetit, and Fua developed a local descriptor, DAISY, which depends on histograms of gradients like SIFT and GLOH but uses a Gaussian weighting and circularly symmetrical kernel [3]. Tola, Lepetit, and Fua compute 200-length descriptors for every pixel in an 800x600 image in less than 5 seconds. DAISY consists of a vector made of values from the convolved orientation maps located on concentric circles centered on the location, and where the amount of Gaussian smoothing is proportional to the radii of the circles [3].

Luo, Xue, and Tian proposed a novel method based on making use of both SIFT features and the local intensity histograms on the feature points in order to achieve more robust image matching [7]. Luo, Xue, and Tian demonstrate that many false matches can be rejected by the proposed method.

3. Motivation and efficacy

COIFv6, like its predecessors, is license free and may be used for any purpose. COIF is primarily intended for use in image stitching applications. COIF uses the Moravec corner detection algorithm because of the algorithm's simplicity combined with good enough time efficiency. COIFv6 can be implemented in under 2,000 lines of Java code, with a reference implementation consisting of 1,682 lines of code using only standard libraries at time of publish. The COIFv6 reference implementation may be found at https://github.com/puckowski/coif.

The best 25 matches (or fewer if the respective algorithm yielded less than 25 matches total) from the COIFv6 and Scale Invariant Feature Transform (SIFT) algorithms were evaluated for 73 image pairs. The results of the evaluation are detailed below.

Description	SIFT	COIF
Instances Equal	55	55
SIFT Better	11	-
COIF Better	-	8
Accuracy (%)	98.9589	98.5205
More Accurate (%)	+0.4384	-

Detailed Accuracy Distribution

COIFv6

Accuracy Range	Count
100%	60
99-95%	6
94-90%	4
89-85%	0
84-80%	3

SIFT

Accuracy Range	Count
100%	65
99-95%	4

94-90%	1
89-85%	2
84-80%	0
79-75%	1

In addition to feature matching accuracy, COIFv6 was evaluated for performance under varying light conditions, perspective transformations, and scale changes. The results of the second evaluation are detailed below.

Effect	**Accuracy Range**
Light Variation	+/- 10%
Perspective Transformation	25%
Scale Change	+/- 50%

COIFv6 is robust enough for typical variations in images of scenes taken with a camera, one image shortly after another. This makes COIFv6 suitable for image stitching applications. As COIFv6 is dependent on luminance histograms for feature matching, it is most sensitive to changes in light and performance degrades as the average luminance of an image decreases. One limitation to consider is the size of the COIFv6 feature in terms of pixels. A COIFv6 feature measures 68 pixels across. This means that no feature matches will be found within 68 pixels of the border around an image.

The COIFv6 algorithm is designed to be a fire and forget algorithm, meaning one set of parameter defaults and one algorithm should provide suitable feature matches for a broad variety of images. One deficiency of the COIFv6 algorithm is that it assumes the Moravec corner algorithm will find at least 2,500 corners with a threshold of 100. The threshold of 100 may be too high for some images and will result in less than the recommended 2,500 corners per image.

3. Parameter defaults

- Feature to feature maximum distinctiveness threshold is distinctiveness plus 10
- Feature to feature minimum distinctiveness threshold is distinctiveness minus 10
- Longest sequence count increment threshold is 25
- Concentric oval center offset is 4 pixels
- Distinctiveness threshold is 2 per bin
- Second innermost oval radius is outer radius squared divided by 3
- Innermost oval radius is outer radius divided by 7
- Moravec processor threshold 100
- Moravec processor local area corner maximum is 2 percent of the image area
- Moravec processor local area corner maximum count is 40
- Grayscale image scalar is 0.5
- Original bin threshold for match is 38 plus 2 per iteration
- Original bin merge count is 1 plus 1 per iteration
- Original bin distance increment negation threshold is 57 plus 3 per iteration
- Concentric oval feature outermost radius is 30 pixels
- Bin threshold floor scalar is 0.98
- Bin threshold ceiling scalar is 1.02
- Maximum bin difference threshold is 40

- Original feature distinctiveness scalar is 0.35 minus 0.05 per iteration
- Minimum feature count is 2,500 unless the original list's maximum is less than 2,500
- Feature longest sequence removal threshold is 70
- Original bin distance increment negation threshold scalar is 0.85
- Minimum feature match count is 5
- Feature match closeness threshold is 0.007
- Maximum bin threshold for match is 56

4. The algorithm

points1 = moravec corner detection on image1 with threshold 100, maximum 40 points for 2% of image area, and discarding corners in patches with less than 6.0 shannon entropy

points2 = moravec corner detection on image2 with threshold 100, maximum 40 points for 2% of image area, and discarding corners in patches with less than 6.0 shannon entropy

image1 = image1 * 0.5

image2 = image2 * 0.5

binThreshold = 38

binNegationThreshold = 57

featureMatchCount = 0

featureMatchCloseness = 0.00

LOOP WHILE (binThreshold < 56 AND (featureMatchCount < 5 OR featureMatchCloseness < 0.007)) OR first iteration

    binMergeCount = 1

    binThreshold = binThreshold + 2

    binNegationThreshold = binNegationThreshold + 3

```
reducedBinNegationThreshold = binNegationThreshold * 0.85

LOOP (featureMatchCount < 5 OR featureMatchCloseness < 0.007) OR first iteration
        featureList1 = empty list

        LOOP points1
                concentricOvalList = empty list

                IF point x,y - 4 in points1 +/- radius 30 fits within image1 bounds
                        outerHistogram = histogram of pixels within radius
                        surroundedHistogram = histogram of pixels within radius / 3
                        centralHistogram = histogram of pixels within radius / 7
                        concentricOvals = [ outerHistogram, surroundedHistogram,
                centralHistogram ]
                        concentricOvalList add concentricOvals

                IF point x - 4,y in points1 +/- radius 30 fits within image1 bounds
                        outerHistogram = histogram of pixels within radius
                        surroundedHistogram = histogram of pixels within radius / 3
                        centralHistogram = histogram of pixels within radius / 7
                        concentricOvals = [ outerHistogram, surroundedHistogram,
                centralHistogram ]
                        concentricOvalList add concentricOvals

                IF point x + 4,y in points1 +/- radius 30 fits within image1 bounds
                        outerHistogram = histogram of pixels within radius
                        surroundedHistogram = histogram of pixels within radius / 3
                        centralHistogram = histogram of pixels within radius / 7
                        concentricOvals = [ outerHistogram, surroundedHistogram,
                centralHistogram ]
                        concentricOvalList add concentricOvals

                IF point x,y + 4 in points1 +/- radius 30 fits within image1 bounds
                        outerHistogram = histogram of pixels within radius
                        surroundedHistogram = histogram of pixels within radius / 3
                        centralHistogram = histogram of pixels within radius / 7
                        concentricOvals = [ outerHistogram, surroundedHistogram,
                centralHistogram ]
                        concentricOvalList add concentricOvals

                featureList1 add concentricOvalList

        featureList2 = empty list
```

```
LOOP points2
        concentricOvalList = empty list

        IF point x,y - 4 in points1 +/- radius 30 fits within image2 bounds
                outerHistogram = histogram of pixels within radius
                surroundedHistogram = histogram of pixels within radius / 3
                centralHistogram = histogram of pixels within radius / 7
                concentricOvals = [ outerHistogram, surroundedHistogram,
        centralHistogram ]
                concentricOvalList add concentricOvals

        IF point x - 4,y in points1 +/- radius 30 fits within image2 bounds
                outerHistogram = histogram of pixels within radius
                surroundedHistogram = histogram of pixels within radius / 3
                centralHistogram = histogram of pixels within radius / 7
                concentricOvals = [ outerHistogram, surroundedHistogram,
        centralHistogram ]
                concentricOvalList add concentricOvals

        IF point x + 4,y in points1 +/- radius 30 fits within image2 bounds
                outerHistogram = histogram of pixels within radius
                surroundedHistogram = histogram of pixels within radius / 3
                centralHistogram = histogram of pixels within radius / 7
                concentricOvals = [ outerHistogram, surroundedHistogram,
        centralHistogram ]
                concentricOvalList add concentricOvals

        IF point x,y + 4 in points1 +/- radius 30 fits within image2 bounds
                outerHistogram = histogram of pixels within radius
                surroundedHistogram = histogram of pixels within radius / 3
                centralHistogram = histogram of pixels within radius / 7
                concentricOvals = [ outerHistogram, surroundedHistogram,
        centralHistogram ]
                concentricOvalList add concentricOvals

        featureList2 add concentricOvalList

        LOOP featureList1
                LOOP concentricOvalList concentricOvals
                        histogramLength = 256 / binMergeCount

                        binIndex = 0
```

```
angleIndex = 0

sum1 = 0

sum2 = 0

distances1 = histogram of histogramLength

LOOP histogramLength i
        sum1 += outerHistogram i

        sum2 += surroundedHistogram i

        binIndex = binIndex + 1

        IF binIndex = binMergeCount
                distances1 at angleIndex = sum1 - sum2

                angleIndex = angleIndex + 1

                binIndex = 0

distances2 = histogram of histogramLength

sum1 = 0

sum2 = 0

binIndex = 0

angleIndex = 0

LOOP histogramLength i
        sum1 += surroundedHistogram i

        sum2 += centralHistogram i

        binIndex = binIndex + 1

        IF binIndex = binMergeCount
                distances2 at angleIndex = sum1 - sum2

                angleIndex = angleIndex + 1
```

```
                    binIndex = 0

            concentricOvals distances1 = distances1

            concentricOvals distances2 = distances2

            score = 0

            LOOP outerHistogram i
                IF outerHistogram i < distinctiveness threshold 2
                    score = score + 1

            concentricOvals distinctiveness = 256 - score

            concentricOvals max compare distinctiveness =
concentricOvals distinctiveness + 10

            concentricOvals min compare distinctiveness =
concentricOvals distinctiveness - 10

            longestSequence = 0

            count = 0

            LOOP outerHistogram i
                IF outerHistogram i < 25
                    count = count + 1

                ELSE
                    IF longestSequence < count
                        longestSequence = count

                    count = 0

            concentricOvals longestSequence = longestSequence

LOOP featureList2
    LOOP concentricOvalList concentricOvals
            histogramLength = 256 / binMergeCount

            binIndex = 0

            angleIndex = 0
```

```
sum1 = 0

sum2 = 0

distances1 = histogram of histogramLength

LOOP histogramLength i
        sum1 += outerHistogram i

        sum2 += surroundedHistogram i

        binIndex = binIndex + 1

        IF binIndex = binMergeCount
                distances1 at angleIndex = sum1 - sum2

                angleIndex = angleIndex + 1

                binIndex = 0

distances2 = histogram of histogramLength

sum1 = 0

sum2 = 0

binIndex = 0

angleIndex = 0

LOOP histogramLength i
        sum1 += surroundedHistogram i

        sum2 += centralHistogram i

        binIndex = binIndex + 1

        IF binIndex = binMergeCount
                distances2 at angleIndex = sum1 - sum2

                angleIndex = angleIndex + 1

                binIndex = 0
```

```
                concentricOvals distances1 = distances1

                concentricOvals distances2 = distances2

                score = 0

                LOOP outerHistogram i
                        IF outerHistogram i < distinctiveness threshold 2
                                score = score + 1

                concentricOvals distinctiveness = 256 - score

                concentricOvals max compare distinctiveness =
concentricOvals distinctiveness + 10

                concentricOvals min compare distinctiveness =
concentricOvals distinctiveness - 10

                longestSequence = 0

                count = 0

                LOOP outerHistogram i
                        IF outerHistogram i < 25
                                count = count + 1

                        ELSE
                                IF longestSequence < count
                                        longestSequence = count

                                count = 0

                concentricOvals longestSequence = longestSequence

distinctivenessModifier = 0.35

LOOP (featureList1 - count < 2500 AND featureList1 > 2500) OR first
iteration
        sum = 0

        distinctivenessModifier = distinctivenessModifier - 0.05

        count = 0
```

```
        LOOP featureList1
                LOOP concentricOvalList concentricOvals
                        sum = sum + concentricOvals distinctiveness

                        count = count + 1

        sum = sum / count

        sumPiece = sum * distinctivenessModifier

        highSum = sum + sumPiece

        count = 0

        LOOP featureList1
                sum = 0

                LOOP concentricOvalList concentricOvals
                        sum = sum + concentricOvals distinctiveness

                sum = sum / 4

                IF sum < highSum
                        count = count + 1

LOOP featureList1
        sum = 0

        LOOP concentricOvalList concentricOvals
                sum = sum + concentricOvals distinctiveness

        sum = sum / 4

        IF sum < sumHigh
                featureList1 remove concentricOvalList

distinctivenessModifier = 0.35

LOOP (featureList2 - count < 2500 AND featureList2 > 2500) OR first
iteration
        sum = 0

        distinctivenessModifier = distinctivenessModifier - 0.05
```

```
        count = 0

        LOOP featureList2
                LOOP concentricOvalList concentricOvals
                        sum = sum + concentricOvals distinctiveness

                        count = count + 1

        sum = sum / count

        sumPiece = sum * distinctivenessModifier

        highSum = sum + sumPiece

        count = 0

        LOOP featureList2
                sum = 0

                LOOP concentricOvalList concentricOvals
                        sum = sum + concentricOvals distinctiveness

                sum = sum / 4

                IF sum < highSum
                        count = count + 1

LOOP featureList2
        sum = 0

        LOOP concentricOvalList concentricOvals
                sum = sum + concentricOvals distinctiveness

        sum = sum / 4

        IF sum < sumHigh
                featureList2 remove concentricOvalList

LOOP featureList1 > 20000
        Remove concentricOvalList at random from featureList1

LOOP featureList2 > 20000
        Remove concentricOvalList at random from featureList2
```

```
LOOP featureList1
	LOOP concentricOvalList concentricOvals
		IF concentricOvals longestSequence > 70
			featureList1 remove concentricOvals

LOOP featureList2
	LOOP concentricOvalList concentricOvals
		IF concentricOvals longestSequence > 70
			featureList2 remove concentricOvals

featureMatchList = empty array

LOOP featureList1
	LOOP featureList2
		lowestDistance = 99999

		compareIndex = 0

		lowestRoughBinDistance = 99999

		LOOP [ [ 0, 1, 2, 3 ], [ 1, 2, 3, 0 ], [ 2, 3, 0, 1 ], [ 3, 0, 1, 2 ], ]
			compareIndex = compareIndex + 1

			distanceFinal = 0

			roughBinDistance = 0

			IF feature1 distinctiveness < feature1 min
distinctiveness OR feature1 distinctiveness > feature1 max
distinctiveness
				distanceFinal = 99999

			secondDistances1 = featureList2
concentricOvalList2 distances1

			LOOP featureList1 concentricOvalList1 distances1 i
				val = distances1 i

				val2 = secondDistances1 i

				valLow = val * 0.98

				valThresholdCheck = | val - valLow |
```

```
                                    IF valThresholdCheck > 40
                                        valLow = val - 40

                                    valHigh = val * 1.02

                                    valThresholdCheckHigh = | val - valHigh |

                                    IF valThresholdCheckHigh > 40
                                        valHigh = val + 40

                                    IF val2 < valLow OR val2 > valHigh
                                        binDistance = binDistance + 1

                                        roughBinDistance =
roughBinDistance + 1

                                        IF | val2 - val | <
reducedBinNegationThreshold
                                            binDistance = binDistance - 1
                                        ELSE
                                            binDistance = binDistance +
1

                                        IF binDistance >= binThreshold
                                            BREAK LOOP

                                secondDistances2 = featureList2
concentricOvalList2 distances2

                                LOOP featureList2 concentricOvalList2 distances2 i
                                    val = distances2 i

                                    val2 = secondDistances2 i

                                    valLow = val * 0.98

                                    valThresholdCheck = | val - valLow |

                                    IF valThresholdCheck > 40
                                        valLow = val - 40

                                    valHigh = val * 1.02

                                    valThresholdCheckHigh = | val - valHigh |
```

```
                                        IF valThresholdCheckHigh > 40
                                            valHigh = val + 40

                                        IF val2 < valLow OR val2 > valHigh
                                            binDistance = binDistance + 1

                                            roughBinDistance =
                roughBinDistance + 1

                                            IF | val2 - val | <
                reducedBinNegationThreshold
                                                binDistance = binDistance - 1
                                            ELSE
                                                binDistance = binDistance +
                1

                                            IF binDistance >= binThreshold
                                                BREAK LOOP

                                    distanceFinal = distanceFinal + binDistance

                                IF lowestDistance > distanceFinal
                                    compareIndexMatch = compareIndex - 1

                                    lowestDistance = distanceFinal

                                    lowestRoughBinDistance = roughBinDistance

                            distanceFinal = lowestDistance

                            roughBinDistance = lowestRoughBinDistance

                            IF distanceFinal < binThreshold
                                featureMatchList add feature match with roughBinDistance
                        and compareIndexMatch

                featureMatchCount = featureMatchList size

                featureMatchCloseness = feature max x - min x * feature max y - min y / image1
        width * height

                binMergeCount = binMergeCount + 1
```

```
        featureMatchCloseness = feature max x - min x * feature max y - min y / image1 width *
        height

index0Sum = 0

index1Sum = 0

index2Sum = 0

index3Sum = 0

LOOP featureMatchList match
        IF match compareIndexMatch = 0
                index0Sum = index0Sum + 1

        IF match compareIndexMatch = 1
                index1Sum = index1Sum + 1

        IF match compareIndexMatch = 2
                index2Sum = index2Sum + 1

        IF match compareIndexMatch = 3
                index3Sum = index3Sum + 1

maxIndexSum = MAX index0Sum index1Sum index2Sum index3Sum

maxIndex = index of maxIndexSum

LOOP featureMatchList match
        IF match compareIndexMatch IS NOT maxIndex
                featureMatchList remove match
```

4.1 Selecting the best matches

An optional step is to select the best *N* matches. The pseudo-code for the algorithm to select the best N matches is detailed below.

```
featureMatches = featureMatches sorted by lowestRoughBinDistance ascending

featureMatchIndex featureMatches length - 1

LOOP featureMatches length > N AND featureMatchIndex >= 0

        featureMatch = featureMatches at featureMatchIndex
```

```
x = featureMatch x

y = featureMatch y

LOOP featureMatches i

        IF i = featureMatchIndex
                CONTINUE

        featureMatch2 = featureMatches at i

        IF featureMatch2 x >= x - 15 AND featureMatch2 x <= x + 15

                IF featureMatch2 y >= y - 15 && featureMatch2 <= y + 15

                        featureMatches remove i

                        IF featureMatchIndex > i

                                featureMatchIndex = featureMatchIndex - 1

                        i = i -  1

                        IF featureMatches size = N

                                featureMatchIndex = -1

                                BREAK

    featureMatchIndex = featureMatchIndex - 1

LOOP featureMatches size > N

    featureMatches remove featureMatches size - 1
```

4.2 Strategy for bounded homography

If enough feature matches are identified but some bounded homography method, such as RANSAC or some variation of RANSAC, does not yield a satisfactory result, then the original bin threshold for match should be reduced. If too few corners are detected, reduce the moravec processor threshold. Increasing the number of corners helps increase the number of matches found. A target of 2,500 corners per image balances time efficiency with, typically, a good number of feature matches.

5. Results

Matching COIF features yields enough reliable matches to be used for image stitching given a range of affine transformations.

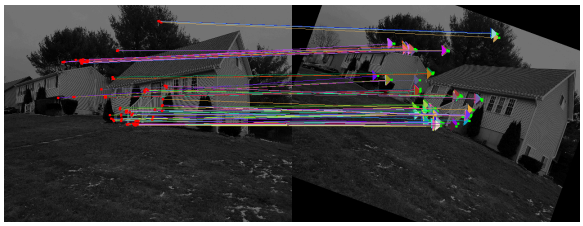

Figure 3

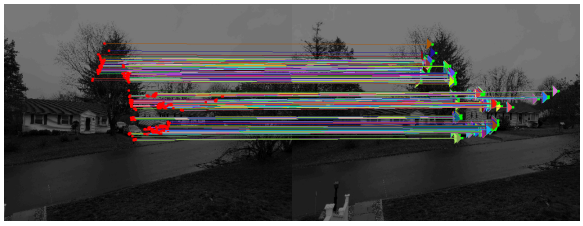

Figure 4

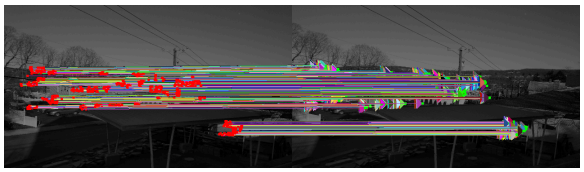

Figure 5

COIF is sensitive to blurs. Increasing the threshold t may yield more matches at the expense of introducing false positives. Further work, such as further computations on a scale pyramid of a given image to introduce scale invariance and refinements of the descriptor computation to be less sensitive to blurs may be researched.

References

[1] Hans P. Moravec. Techniques Towards Automatic Visual Obstacle Avoidance. https://frc.ri.cmu.edu/~hpm/project.archive/robot.papers/1977/aip.txt

[2] Constantin Vertan and Nozha Boujemaa. Using Fuzzy Histograms and Distances for Color Image Retrieval.

[3] Engin Tola, Vincent Lepetit, and Pascal Fua. Daisy: An Efficient Dense Descriptor Applied to Wide Baseline Stereo.

[4] Ethan Rublee, Vincent Rabaud, Kurt Konolige, and Gary Bradski. ORB: an efficient alternative to SIFT or SURF.

[5] Herbert Bay, Tinne Tuytelaars, and Luc Van Gool. SURF: Speeded Up Robust Features.

[6] David G. Lowe. Distinctive Image Features from Scale-Invariant Keypoints.

[7] Ye Luo, Ping Xue, and Qi Tian. Image Histogram Constrained SIFT Matching.

[8] Martin A. Fischler, and Robert C. Bolles. Random Sample Consensus: A Paradigm for Model Fitting with Applications to Image Analysis and Automated Cartography.