Curtin University – Department of Computing

Assignment Cover Sheet / Declaration of Originality

Complete this form if/as directed by your unit coordinator, lecturer or the assignment specification.

Last name:	Patel	Student ID:	19460606
Other name(s):	Harshil		
Unit name:	Machine Perception	Unit ID:	Comp 3007
Lecturer / unit coordinator:	Senjian	Tutor:	
Date of submission:	05/10/2020	Which assignment?	1 (Leave blank if the unit has only one assignment.)

I declare that:

- The above information is complete and accurate.
- The work I am submitting is *entirely my own*, except where clearly indicated otherwise and correctly referenced.
- I have taken (and will continue to take) all reasonable steps to ensure my work is *not accessible* to any other students who may gain unfair advantage from it.
- I have *not previously submitted* this work for any other unit, whether at Curtin University or elsewhere, or for prior attempts at this unit, except where clearly indicated otherwise.

I understand that:

- Plagiarism and collusion are dishonest, and unfair to all other students.
- Detection of plagiarism and collusion may be done manually or by using tools (such as Turnitin).
- If I plagiarise or collude, I risk failing the unit with a grade of ANN ("Result Annulled due to Academic Misconduct"), which will remain permanently on my academic record. I also risk termination from my course and other penalties.
- Even with correct referencing, my submission will only be marked according to what I have done myself, specifically for this assessment. I cannot re-use the work of others, or my own previously submitted work, in order to fulfil the assessment requirements.
- It is my responsibility to ensure that my submission is complete, correct and not corrupted.

Signature:  Date of signature: 05/10/2020

(By submitting this form, you indicate that you agree with all the above text.)

Machine Perception Assignment 1

Harshil Patel
19460606

In this report, I have created one section for each of the tasks required of us in this assignment. Following each task are the images I have produced for these tasks, and I have labeled and referenced them in my descriptions as (see Figure *n*). After the Tasks, I have the sections for Appendices, here I have included all of my code. Appendix *n* refers to task *n*. I also have an Appendix 0, in which I have included the contents of a file called tools.py, which I created for code reuse, as there are methods that are repeated many times throughout this assignment. I have also created a google drive with my results images so you can open and look at them closer if converting to PDF has affected the quality of them (https://drive.google.com/drive/folders/1-iIM-xr02I6eRHbfgLbuJksGutJYoKcd?usp=sharing). I have referenced external code throughout the assignment, and those references are above the actual code, ie. it will read - Appendix *n* - then references - then code.

I. Task 1 - Image histogram, Harris corners and SIFT key points (code is in Appendix 1)

i Variance / Invariance against scaling and rotations:

a) Histograms for the scaled images will be the same shape as the non scaled images, this is because the image will still consist of the same colours and/or intensities in the same proportions. This does mean that the scale of the histogram will change too, ie. the values at the x and y axis will change. The rotated images, however, will show variance, as this will either cut corners, or introduce black corners, as is done in my example.

b) Corners detected using Harris Corner Detection should remain the same regardless of orientation, as this is rotation invariant. Scale may effect the corners detected depending on the magnitude of the change. The larger, or more up-scaled the image is, ie. the closer you zoom in, the corners will start too look more straight, and this will mean that they wont be detected as easily. Same goes for shrinking an image, a curved line that wasn't originally a corner may look like a corner when zoomed out far enough, and thus will be detected.

c) SIFT is rotation and scale invariant, so should not matter if there are any rotations, or change in scale, however, this is only true up to around $2 - 2.5x$ scaling, and $40°$ rotations.

ii Comparison of scaled and/or rotated images and their key points:

a) In the generation of scaled and rotated images, I realised that rotations would pose a challenge in a couple of ways. Firstly, when using a "non square" rotation, ie. non $90°$ $180°$ or $270°$, I would face the problem of truncating the corners of the image, and/or introducing unnecessary black corners. Secondly, if I was to only use a $90°$ rotation, my testing of SIFT features would not be accurate, as they are only accurate up to $40°$. To counter this, I created two rotated images for each of the original images, one which was a $90°$ rotation, and one which was a $30°$ rotation. With the second rotation, I used code from an external source (which I have referenced in my appendix) to crop the corners down to fit the image, this would minimize the excess black corners.

iii Findings:

a) My findings through experimentation were mostly accurate to my original expectations. The histograms for the scaled and original images looked proportionately identical, however their x and y values had scaled up. With the rotated histograms, the $30°$ histogram was different as it included black corners, and so there was a huge spike near the "low intensity" side, just as I had suspected. The $90°$ histogram however was the exact same, as the same colors and intensities, in the same proportions were detected. These results were all very consistent regardless of image, ie. the diamond card image and the dugong image both yielded the same outcomes. In the below image, the top row is for the playing card image, and the bottom row for the dugongs. From left to right, each figure represents - Original, scaled, rotated $30°$, rotated $90°$, rotated $90°$ and then scaled (see Figure 1).

Harris corner detection was varient to scaling, as described above, the more zoomed in an image is, the less the curves look like corners, this was clearly indicated in my diamond card image, when comparing the corner detection in the original and scaled versions. The main points are still the same, however, on the bottom left corner of the card, there are detections in the original, which are not present in the resized image (see Figure 5). There are also less detections on the dugong in the scaled up version, most notably, on the nose of the mother (see Figure 10). I am sure that a more aggressive rescaling would result in a more spectacular difference in the number of detections. There was also an obvious variance when the image was rotated. In the $30°$ rotation, none of the corners of the card were detected in the diamond image (see

Figure 3, and in the dugong image, there were a few detections in the water missing that were in the original, and also the nose of the mother dugong again (see Figure 8. In the 90° rotation, a few corners went undetected in the diamond card image (see Figure 4), however fared well in the dugong image, and I could not see any missed detections here (see Figure 9).

My biggest issue was with the SIFT key-points I was detecting. Regardless of the fact that they should be invariant to scaling and rotation, my results seemed to suggest otherwise. I only used a scaling of a $1.5x$ increase (see Figures 15 and 20), and two rotations, of each, 30° (see Figures 13 and 18) and 90° (see Figures 14 and 19). Despite SIFT being invariant up to $2 - 2.5x$, and 40° rotation, I detected different key-points for my scalings and rotations the to original image. I originally assumed that the SIFT detector might only be returning the strongest n key-points, however, for the number of key-points detected differed from image to image, and so I'm not entirely sure why it detects different key-points with such little scaling and rotations. Another theory was that maybe it was detecting them, but not displaying them on the image, however, once again, I am not sure.

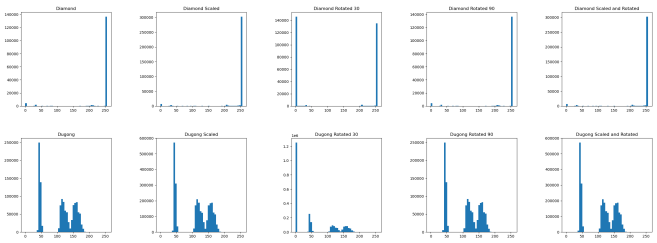

Figure 1. Histogram Comparison

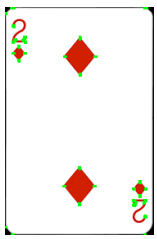

Figure 2. Original Diamond Card

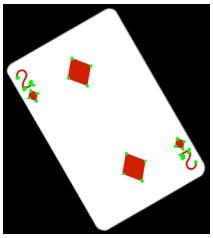

Figure 3. Diamond Rotated 30°

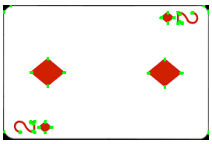

Figure 4. Diamond Rotated 90°

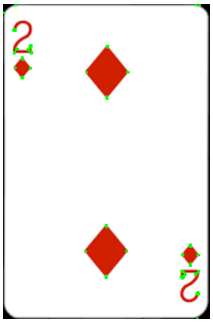

Figure 5. Diamond Scaled

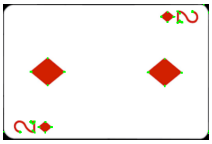

Figure 6. Diamond Rotated 90° then Scaled

II. Task 2 - Image Features (code is in Appendix 2)

i LBP, HOG and SIFT

a) Local Binary Patterns - LBP's are texture descriptors. To perform an LBP extraction on an image, first we must convert the image to grayscale, as this is the only colour space it will work on, we will be comparing intensities of particular pixels. Then we divide the image into neighborhoods, each with a set number of pixels surrounding the center pixel, usually the 8 pixels surrounding the center. We then take the center pixel and use it as a threshold value, and convert all pixels in its neighborhood to 1 if it is a higher value than the center pixel, or 0 if it is a lower value. We then start at one of the surrounding pixels (but be consistant with the pixel chosen!) in the neighborhood, and work our way clockwise building a binary string, representing a number in binary, this is then converted into a decimal, and is stored in an output array the same size as the original image [1].

b) Histogram of Gradients - HOG's are detected by first cropping/scaling the image to a desired size, then calculate the histogram of gradients, which is done by filtering the image with these kernals - $[-1, 0, 1]$ for x axis and $[-1, 0, 1]^T$ for y axis. Then find the magnitude and direction of the gradients. We then want to calculate the histogram of gradients in 8x8 cells. This can vary. Each gradient value is placed into 1 of 9 bins in an array, representing the degrees. These bins often look like this $[0, 20, 40, 60, 80, 100, 120, 140, 160]$ and each gradient magnitude is entered into these bins depending on the gradients direction. ie. if a pixel has a gradient angle of 80, and a magnitude of 4, 4 will be placed inside the

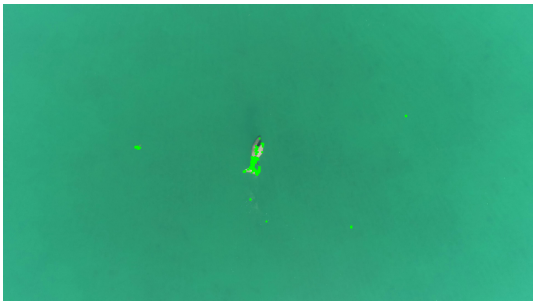

Figure 7. Original Dugong

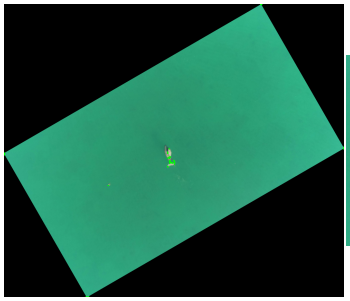

Figure 8. Dugong Rotated 30°

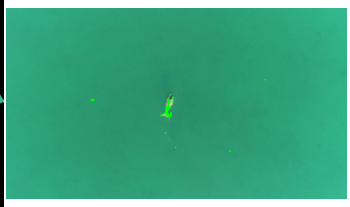

Figure 9. Dugong Rotated 90°

Figure 10. Dugong Scaled

Figure 11. Dugong Rotated 90° then Scaled

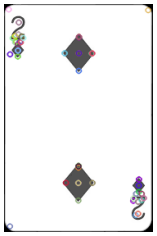

Figure 12. Diamond SIFT

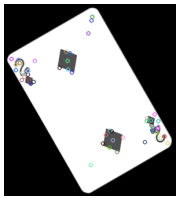

Figure 13. Diamond Rotated 30° SIFT

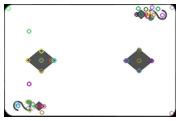

Figure 14. Diamond Rotated 90° SIFT

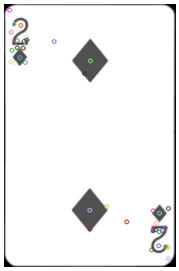

Figure 15. Diamond Scaled SIFT

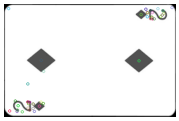

Figure 16. Diamond Rotated 90° then Scaled SIFT

80 bin. If you had a direction of 10 and a magnitude of 2, then 1 will be placed in the 20, and one in the 0 bin. You will then have a 9 bin histogram for each 8x8 cell. Then we normalize the gradient strengths in each block, (eg. in our case we might use 16x16 blocks). We then have our final HOG descriptor [2].

c) Scale Invariant Feature Transform - SIFT is a feature descriptor and detector. The steps required for SIFT are as follows:

i) **Scale-space extrema detection -** This step creates multiple copies of the image at varying scales to help make the detection scale invariant. This is done to avoid variance when an image is a different size, like in Harris Corner detection. For example when an image is zoomed in, or scaled up, a curve the once registered as a corner, may not anymore as it would look less ”sharp”. During this step, multiple levels of Gaussian Blurs are applied on these images, to find the Difference of Gaussian, ie. DoG analysis. Then the DoG is used to calculate the LoG, ie. the Laplacian of Gaussian, which is just applying a Laplacian (second order differential) to these. Each pixel is checked against its 8 neighbouring pixels, and also the 9 pixels directly above in the next scale, and the 9 below it. Therefore there are 26 checks, and this will help detect edges and corners that are scale invariant.

ii) **Keypoint localization -** In this step we eliminate ant low contrast keypoints, which is done by comparing intensities, eg. DoG has a high response for edges, so some of these need to be removed from the detection.

iii) **Orientation assignment -** Next step, we check rotation invariance, this is done by once again taking the neighbourhood around a keypoint, and an orientation histogram is created with 36 bins, each at 10° increments, thus covering a full 360°. Then we add an amount proportional to the gradient at that point, to the corresponding bin, this is a similar process

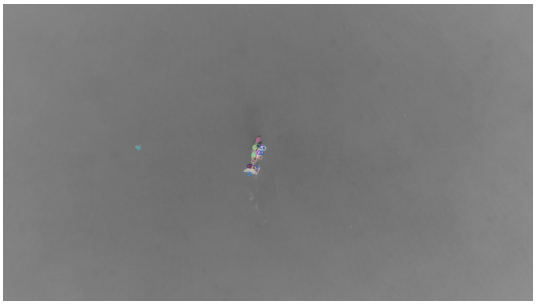

Figure 17. Dugong SIFT

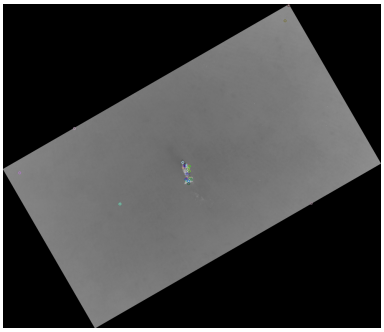

Figure 18. Dugong Rotated 30° SIFT

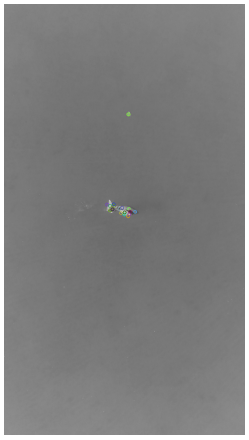

Figure 19. Dugong Rotated 90° SIFT

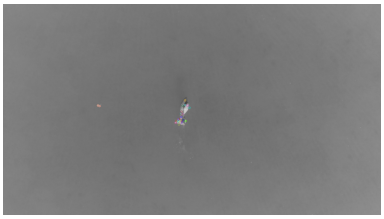

Figure 20. Dugong Scaled SIFT

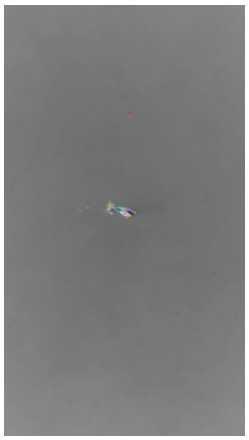

Figure 21. Dugong Rotated 90° then Scaled SIFT

to HOG's binning system. Out of these points, the highest peak, and any peak above 80% is considered to calculate the orientation.

iv) **Keypoint descriptors -** Now each keypoint has a location, scale and orientation, now we need to compute descriptors for the local image region around each keypoint that is as distinctive and invariant as possible. This uses a $16x16$ block around the keypoint, which is further divided into 16 $4x4$ blocks. Then we create an 8 bin orientation histogram for each of these blocks, this will give us 128 bin values for the $16x16$ sample size around the keypoint. This vector is rotation dependant, so the keypoints rotation is subtracted from each orientation, so that the gradient orientation is relative to the keypoint. The vector is also illumination dependent, and therefore applying a threshold will make it independent.

v) **Keypoint matching -** Keypoints are matched by identifying their nearest neighbors.

[3]

d) These feature detectors have their advantages and disadvantages and also some similarities. All three of these techniques use a local neighborhood to calculate keypoints. LBP is a very simple, and thus trivial feature descriptor. It is easy to implement, and quick to compute. It is quite good at capturing fine grained details in images. These can be used for segmentation and classification. Unfortuantely the original algorithm is unable to compute details at scales other than its $3x3$ scale. This is solved with an extension to the algorithm however. HOG is another popular technique, often used with object detection, and detection of humans etc. in footage. This is useful in human detection as people vary in size, colour and clothing, and HOG will look at a more general gradient oriented approach, and so these differences don't impact the detector much. The downside to HOG is that as we work in set image sizes, to look at a larger image, we need a larger region size, or more iterations, and this can be more computationally taxing. Finally, HOG is rotation variant, as this is all about orientations and magnitudes of local gradients. SIFT's strongest advantage is its robustness to scale and rotation, this is extremely useful as it is very accurate, and should be prioritized when serious computational efficiency is not required, as there is a lot of processing to be done to detect these keypoints.

ii For each of the images, I chose a feature, cropped the image to get a good representation of it, then ran the detections on both the cropped image, and the original, and then matched the keypoints to the original image. For the diamond card image, I chose the top left diamond, and cropped it out of the original card, and also cropped it out of all of the variations. I did that same thing to the calf and mother on the dugong image. Then I ran SIFT feature detection, on all cropped images, and the original non scaled/rotated images, and matched the keypoints using a brute force matcher, to perform a knn keypoint match. The results I got for the diamond card were very poor, And it failed to match even one keypoint to the original (see Figures 22 to 25). I assumed this is due to a couple of things, first of all, I was matching keypoints from a cropped portion of a rotated or scaled image to the original, so maybe if I matched the cropped scaled image keypoints to the scaled image keypoints, I'd have better luck. Secondly, I was using a brute force

matching algorithm using knn keypoint matching. Maybe if I had tried to use FLANN matching I might have had better success. When I ignored the keypoint matcher, and simply looked at the sift results, there were still some differences in the detected images. Some of the results were promising, eg. the cropped diamond that was rotated 30° matched similar corners to the 90° rotation (see Figures 32 and 33), however the scaled image was not detecting the same thing (see Figure 31.

When I tried the same technique, using the keypoint matcher, on the dugong images, the number of correct matches was much much higher, and SIFT proved to be rather robust against these scalings and rotations (see Figures 26 to 29. And again, by visually inspecting these, we find that the keypoints detected are similar, and many of the same ones are detected, however, there are some discrepancies in detections still (see Figures 34 to 37).

Unfortunately I could not get HOG working in time to submit to this assignment, and so I cannot support my arguments, that SIFT features have more advantages than HOG features, but as mentioned above, HOG features lack the robustness and invariance that SIFT features have.

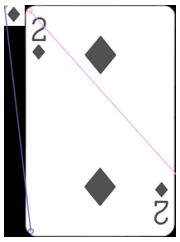

Figure 22. Cropped Diamond Card SIFT Matched Keypoints

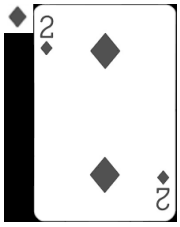

Figure 23. Cropped Scaled Diamond Card SIFT

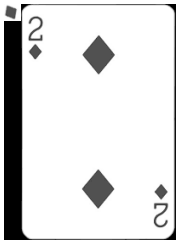

Figure 24. Cropped Rotated 30° Diamond Card SIFT Matched Keypoints

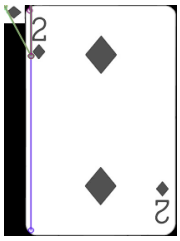

Figure 25. Cropped Rotated 90° Diamond Card SIFT Matched Keypoints

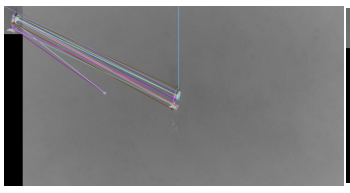

Figure 26. Cropped Dugong SIFT Matched Keypoints

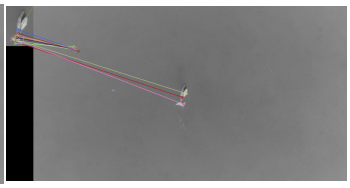

Figure 27. Cropped Scaled Dugong SIFT Matched Keypoints

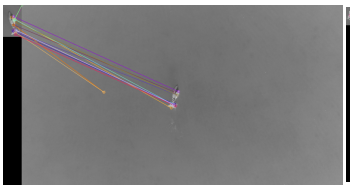

Figure 28. Cropped Rotated 30° Dugong SIFT Matched Keypoints

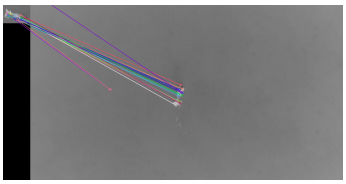

Figure 29. Cropped Rotated 90° Dugong SIFT Matched Keypoints

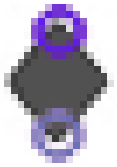

Figure 30. Cropped Diamond Card SIFT

Figure 31. Cropped Scaled Diamond Card SIFT

Figure 32. Cropped Diamond Rotated 30° SIFT

Figure 33. Cropped Diamond Rotated 90° SIFT

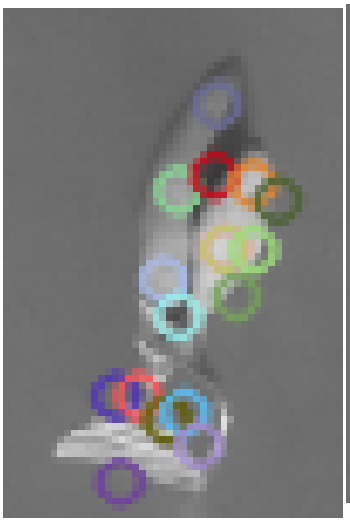

Figure 34. Cropped Dugong SIFT

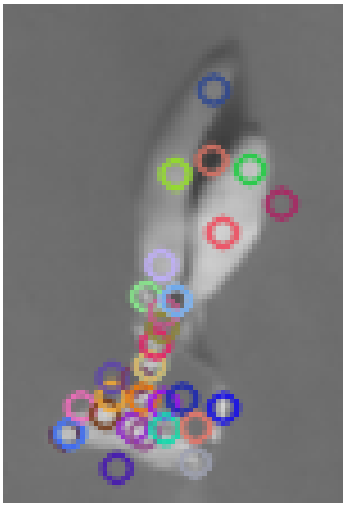

Figure 35. Cropped Scaled Dugong SIFT

Figure 36. Cropped Dugong Rotated 30° SIFT

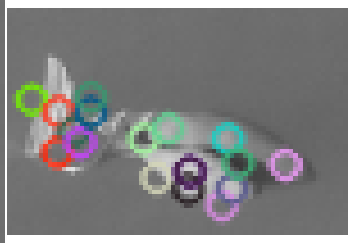

Figure 37. Cropped Dugong Rotated 90° SIFT

III. Task 3 - Object Extraction (code is in Appendix 3)

i For the binary transform of the images, I decided to take two separate approaches for each image. The diamond card image was straight forward, I used OTSU's binarization to separate foreground from background, making sure to get the inverse of this thresholding, as we want the diamonds and the numbers in the foreground (see Figure 38).

ii The dugong image proved rather challenging to get right. I tried a plethora of techniques, including OTSU's binarization (see Figure 42), adaptive thresholding, and even removing colour channels from the image to try to manipulate the image to get a more definite foreground and background. Using this technique, I noticed that removing the green colour channel provides a much better separation from the water, and the dugong/seaweed. I wasn't satisfied with this result however (see Figure 55), and so decided to try something different. I tried working in a completely different colour-space, and the results were astonishing. When I changed the colour-space to HSV, and displayed the image with no further manipulation, I found that the background was a yellowish colour, and the seaweed and dugong were a bright red (see Figure 44). This prompted me to try the thresholding in this colour-space. This attempt was far more successful than the previous ones, however, there were still some tiny holes in the detection, and in the background, and another slight problem was that the thresholding wasn't in black and white, it was yellow and red (see Figure 40).

iii For CCL and blob detection, I used openCV's connectedComponentsWithStats() method, with a connectivity of 8. I tried both 8 and 4 way pixel connectivity, and found no difference in result. This is most likely as each image was rather large, and so were the blobs (see Figures 39 and 41). After detecting the blobs, I calculate the area, width and height of each blob, and print them to the console. I also display the number of blobs detected for each image, and display that to console (see Figures 46 and 47). For the diamond card image, I found a total of 8 labels, ie. 7 blobs plus the background. For the dugong image I found 3 labels, ie. 2 blobs (one for the dugong and calf, and the other for seaweed), plus the background.

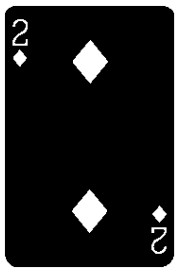

Figure 38. Diamond Binarized

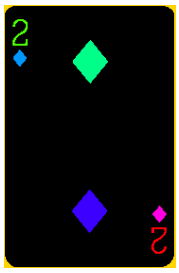

Figure 39. Diamond With Labeled Blobs

Figure 40. Dugong Binarized

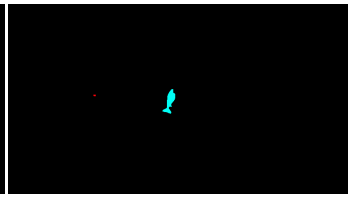

Figure 41. Dugong With Labeled Blobs

Figure 42. Dugong BGR Binarized

Figure 43. Dugong Binarized Without Red Channel

Figure 44. Dugong HSV

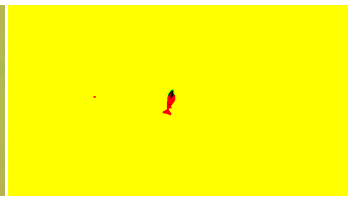

Figure 45. Dugong HSV Binarized

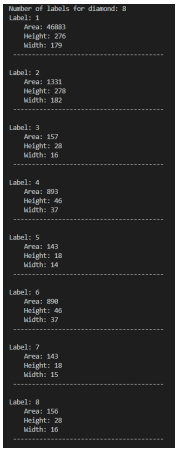

Figure 46. Diamond Blob Details

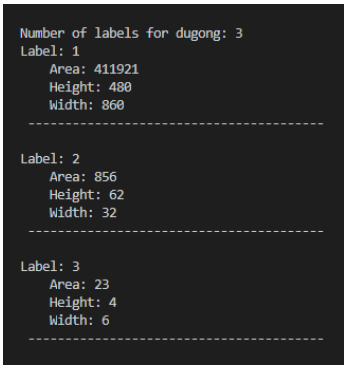

Figure 47. Dugong Blob Details

IV. Task 4 - Image segmentation with K-means (code is in Appendix 4)

1) K-means segmentation relies greatly on colour, and so the diamond card image was very easy to segment using k-means clustering. I knew that the dugong image would prove to be much more challenging. To make this process easier, I decided to try many different combinations of colour channels and spaces, to see which ones were the most effective. I also tried a combination of 3 clusters and 4 clusters,all no. clusters (k values) are 3 unless otherwise specified. After the clustering, I disable the non-important clusters, so that our area of interest is all that remains, just to prove that this segmentation method was successful.
With no manipulation of the colour space or channels, and using 3 clusters, the background was not entirely separated, and parts of the dugong were clustered in the same groups as some parts of the background, as shown in (see Figure 50), here I have disabled all other clusters, thus we can see the ”holes” in the dugong where the clustering grouped parts of the dugong in the same segments as one of the background clusters. I decided try increasing the number of clusters, as this may provide one extra cluster that the background could fit into, and so isolate the dugong and background into different clusters. Trying this, I found that 4 clusters successfully separated the entire background from the dugong. As we can see in (see Figure 52), this creates 4 distinct colour clusters, where the dugong and seaweed, our areas of interest, are a different colour to all of the background. Looking at this, I disabled the other 3 clusters, ie. turned them black, to isolate our interest as shown in (see Figure 53), and the results were quite satisfactory. However, I didn't want to stop there, there were a few gaps in the dugong. I could solve this by applying a morphological opening, however, this felt like cheating, and I was sure I could do it with pure segmentation on the raw images.
This time I tried disabling different colour channels. Disabling the red colour channel did not work so well, the colour separation when the red channel is removed is poor, and so the clustering does not work properly, as the colours are similar to each other. This resulted in a dugong with holes, and a section of background in the same cluster (see Figure 57). Keeping red and removing the other two channels proved to be more useful, removing either green or blue seemed to be equally effective, and resulted in a nicely segmented image, which, when I disable all background clusters, presents a good looking dugong, and seaweed. The results were better than the simple k=4 with no colour channel manipulation, however there was still black holes/lines in the dugong (though less significant than the previous tries) (see Figures 55 and 54). I also tried to remove both green and blue channels, leaving only red, but this didn't work well (see Figure 56).
Finally it was time to try changing colour space, which meant I could try what seems to have worked well for me in the previous task. I converted the image to HSV, and ran the k-means segmentation, and the results were already significantly better than manipulating colour channels, the dugong and seaweed were without holes, the background was fully separated, with only 3 clusters (ie. k = 3). The only quarrel I had with this was that near the tails of the dugongs, there is a little bit of white water from splashing, and it was picking this up (see Figure 51). To solve this I threw in a morphological opening, to get rid of it as it was very thin (see Figure 58).

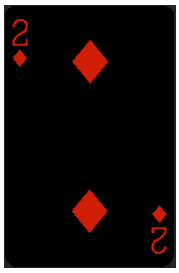

Figure 48. Segmented Diamond

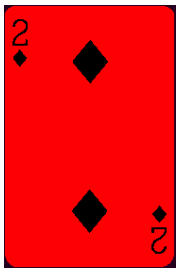

Figure 49. Segmented Diamond HSV

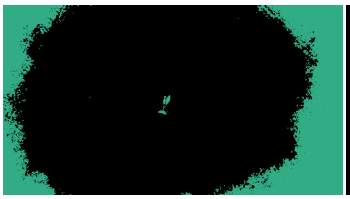

Figure 50. Segmented Dugong

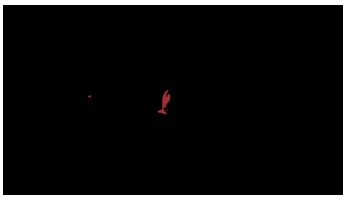

Figure 51. Segmented Dugong HSV

Figure 52. Segmented Dugong k=4 With Background

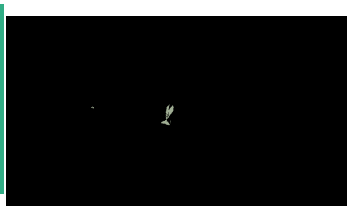

Figure 53. Segmented Dugong k=4

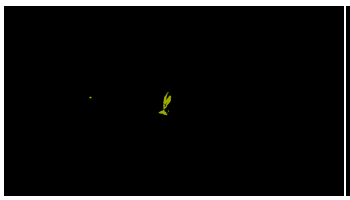

Figure 54. Segmented Dugong Without Blue Channel

Figure 55. Segmented Dugong Without Green Channel

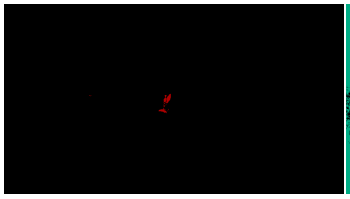

Figure 56. Segmented Dugong Without Blue Or Green Channel

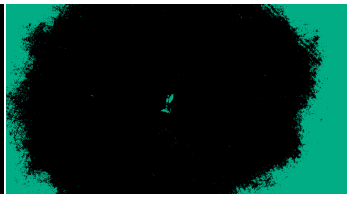

Figure 57. Segmented Dugong Without Red Channel

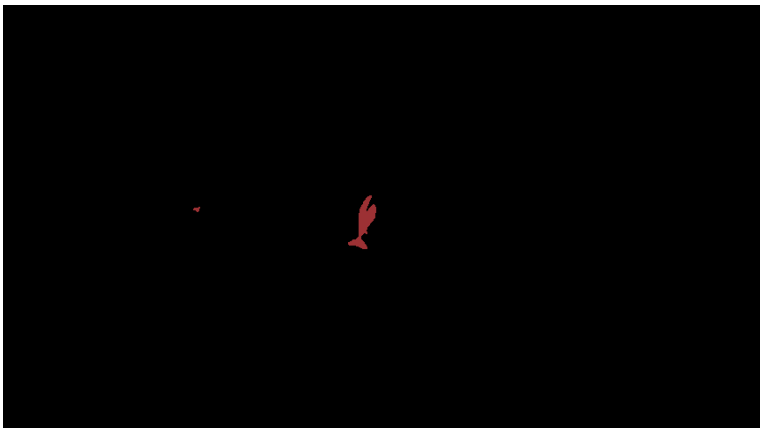

Figure 58. Segmented Dugong HSV With Opening

V. Appendix 0 - This file is **tools.py** used for code reuse

[4] [5] [6] [7] [8] [9]

```python
import cv2 as cv
import numpy as np
import math
from matplotlib import pyplot as plt

# Referenced from https://docs.opencv.org/3.4/d8/dbc/tutorial_histogram_calculation.html

# Here we will make the histograms

def histogramCalc(image):
    bgr_planes = cv.split(image)

    # no. bins
    hist_size = 256

    hist_range = (0, 256)

    accumulate = False

    b_hist = cv.calcHist(bgr_planes, [0], None, [hist_size], hist_range,
    ↪  accumulate=accumulate)
    g_hist = cv.calcHist(bgr_planes, [1], None, [hist_size], hist_range,
    ↪  accumulate=accumulate)
    r_hist = cv.calcHist(bgr_planes, [2], None, [hist_size], hist_range,
    ↪  accumulate=accumulate)

    hist_w = 512
    hist_h = 400
    bin_w = int(round(hist_w / hist_size))

    hist_image = np.zeros((hist_h, hist_w, 3), dtype=np.uint8)

    cv.normalize(b_hist, b_hist, alpha=0, beta=hist_h, norm_type=cv.NORM_MINMAX)
    cv.normalize(g_hist, g_hist, alpha=0, beta=hist_h, norm_type=cv.NORM_MINMAX)
    cv.normalize(r_hist, r_hist, alpha=0, beta=hist_h, norm_type=cv.NORM_MINMAX)

    for ii in range(1, hist_size):
        cv.line(hist_image, (bin_w * (ii - 1), hist_h - int(round(b_hist[ii - 1][0]))),
                (bin_w * (ii), hist_h - int(round(b_hist[ii][0]))), (255, 0, 0),
                ↪  thickness=2)
        cv.line(hist_image, (bin_w * (ii - 1), hist_h - int(round(g_hist[ii - 1][0]))),
                (bin_w * (ii), hist_h - int(round(g_hist[ii][0]))), (0, 255, 0),
                ↪  thickness=2)
        cv.line(hist_image, (bin_w * (ii - 1), hist_h - int(round(r_hist[ii - 1][0]))),
                (bin_w * (ii), hist_h - int(round(r_hist[ii][0]))), (0, 0, 255),
                ↪  thickness=2)

    return hist_image

# Referenced from https://docs.opencv.org/3.4/dc/d0d/tutorial_py_features_harris.html

def cornerHarris(image):
    # gray = cv.cvtColor(img, cv.COLOR_BGR2GRAY)
    img = image.copy()
    gray = cv.cvtColor(img, cv.COLOR_BGR2GRAY)
    gray = np.float32(gray)

    dst = cv.cornerHarris(gray, 2, 3, 0.04)
    dst = cv.dilate(dst, None)

    img[dst > 0.01 * dst.max()] = [0, 255, 0]

    return img
```

```
58
59  # Referenced from https://docs.opencv.org/3.4/da/df5/tutorial_py_sift_intro.html
60
61  def siftKeyPoints(image):
62      img = image.copy()
63      gray = cv.cvtColor(img, cv.COLOR_BGR2GRAY)
64
65      sift = cv.SIFT_create()
66      kp = sift.detect(gray, None)
67
68      img = cv.drawKeypoints(gray, kp, img)
69
70      return img
71
72  # https://docs.opencv.org/3.4/dc/dc3/tutorial_py_matcher.html
73
74  def siftKeyPointMatcher(original, image, filename):
75      og = original.copy()
76      img = image.copy()
77
78      og_gray = cv.cvtColor(og, cv.COLOR_BGR2GRAY)
79      gray = cv.cvtColor(img, cv.COLOR_BGR2GRAY)
80
81      sift = cv.SIFT_create()
82
83      og_kp, og_desc = sift.detectAndCompute(og_gray, None)
84      kp, desc = sift.detectAndCompute(gray, None)
85
86      bf = cv.BFMatcher()
87      matches = bf.knnMatch(desc, og_desc, k=2)
88
89      good = []
90      for m,n in matches:
91          if m.distance < 0.45*n.distance:
92              good.append([m])
93
94      output = cv.drawMatchesKnn(og_gray, og_kp, gray, kp, good, None,
        ↪  flags=cv.DrawMatchesFlags_NOT_DRAW_SINGLE_POINTS)
95
96      cv.imwrite(filename, output)
97      plt.imshow(output),plt.show()
98
99      return output
100
101
102 # Used from
    ↪  https://stackoverflow.com/questions/22041699/rotate-an-image-without-cropping-in-opencv-in-c/335
103
104 def rotate(image, angle):
105     diagonal = int(math.sqrt(pow(image.shape[0], 2) + pow(image.shape[1], 2)))
106     offset_x = (diagonal - image.shape[0])//2
107     offset_y = (diagonal - image.shape[1])//2
108     dst_image = np.zeros((diagonal, diagonal, 3), dtype='uint8')
109     image_center = (diagonal/2, diagonal/2)
110
111     R = cv.getRotationMatrix2D(image_center, angle, 1.0)
112     dst_image[offset_x:(offset_x + image.shape[0]), \
113     offset_y:(offset_y + image.shape[1]), \
114     :] = image
115     dst_image = cv.warpAffine(dst_image, R, (diagonal, diagonal), flags=cv.INTER_LINEAR)
116
117     # Calculate the rotated bounding rect
118     x0 = offset_x
119     x1 = offset_x + image.shape[0]
120     x2 = offset_x
121     x3 = offset_x + image.shape[0]
```

```
122
123     y0 = offset_y
124     y1 = offset_y
125     y2 = offset_y + image.shape[1]
126     y3 = offset_y + image.shape[1]
127
128     corners = np.zeros((3,4))
129     corners[0,0] = x0
130     corners[0,1] = x1
131     corners[0,2] = x2
132     corners[0,3] = x3
133     corners[1,0] = y0
134     corners[1,1] = y1
135     corners[1,2] = y2
136     corners[1,3] = y3
137     corners[2:] = 1
138
139     c = np.dot(R, corners)
140
141     x = int(c[0,0])
142     y = int(c[1,0])
143     left = x
144     right = x
145     up = y
146     down = y
147
148     for i in range(4):
149         x = int(c[0,i])
150         y = int(c[1,i])
151         if (x < left): left = x
152         if (x > right): right = x
153         if (y < up): up = y
154         if (y > down): down = y
155     h = down - up
156     w = right - left
157
158     cropped = np.zeros((w, h, 3), dtype='uint8')
159     cropped[:, :, :] = dst_image[left:(left+w), up:(up+h), :]
160     return cropped
161
162 #
    ↪ https://www.delftstack.com/howto/matplotlib/how-to-display-multiple-images-in-one-figure-correct
163
164 def displayImages(images):
165     rows = 2
166     cols = 5
167     axes = []
168     figure = plt.figure()
169
170     for ii in range(rows * cols):
171         axes.append(figure.add_subplot(rows, cols, ii + 1))
172         plt.imshow(images[ii])
173
174     axes[0].set_title("Diamond")
175     axes[1].set_title("Diamond Scaled")
176     axes[2].set_title("Diamond Rotated 30")
177     axes[3].set_title("Diamond Rotated 90")
178     axes[4].set_title("Diamond Scaled and Rotated")
179     axes[5].set_title("Dugong")
180     axes[6].set_title("Dugong Scaled")
181     axes[7].set_title("Dugong Rotated 30")
182     axes[8].set_title("Dugong Rotated 90")
183     axes[9].set_title("Dugong Scaled and Rotated")
184
185     figure.tight_layout()
186     plt.show()
```

```
187
188 def saveImages(images, filenames):
189     if len(images) != len(filenames):
190         print("Filenames and images not compatible")
191         return 0
192
193     for ii, image in enumerate(images):
194         cv.imwrite(filenames[ii], image)
```

VI. Appendix 1 - variance_invariance.py

[10] [9]

```python
import cv2 as cv
import numpy as np
from matplotlib import pyplot as plt
from tools import *

diamond = cv.imread('Images/diamond.png')
dugong = cv.imread('Images/dugong.jpg')

imgs = [diamond, dugong]
scaled = rows = cols = translations = img_translations = M = rotated = [None] * 2

# ---------------- SCALE AND ROTATE ----------------

# Lets create a scaled version of the diamonds and dugongs
scaled_diamond = cv.resize(diamond, None, fx=1.5, fy=1.5, interpolation=cv.INTER_LINEAR)
rows_diamond, cols_diamond = scaled_diamond.shape[:2]

scaled_dugong = cv.resize(dugong, None, fx=1.5, fy=1.5, interpolation=cv.INTER_LINEAR)
rows_dugong, cols_dugong = scaled_dugong.shape[:2]

# Create 2 rotations, as SIFT is scale invarient up to 2.5x and 40 degree rotation invarient
rotated_90_diamond = cv.rotate(diamond, cv.ROTATE_90_CLOCKWISE)
rotated_90_dugong = cv.rotate(dugong, cv.ROTATE_90_CLOCKWISE)

rotated_30_diamond = rotate(diamond, 30)
rotated_30_dugong = rotate(dugong, 30)

# Hit 'em with both
rotated_scaled_diamond = cv.rotate(scaled_diamond, cv.ROTATE_90_CLOCKWISE)
rotated_scaled_dugong = cv.rotate(scaled_dugong, cv.ROTATE_90_CLOCKWISE)

# ---------------- Histograms ----------------

# Now we use these to check the variance/invariance of certain key-point detections
# We now have 10 images:
# diamond, dugong
# scaled_diamond, scaled_dugong
# rotated_30_diamond, rotated_30_dugong
# rotated_90_diamond, rotated_90_dugong
# rotated_scaled_diamond, rotated_scaled_dugong

images = [diamond, scaled_diamond, rotated_30_diamond, rotated_90_diamond, rotated_scaled_diamond, \
    dugong, scaled_dugong,  rotated_30_dugong,  rotated_90_dugong, rotated_scaled_dugong]

# https://matplotlib.org/3.1.1/gallery/statistics/histogram_multihist.html

bins = 50

figure, axes = plt.subplots(nrows=2, ncols=5)
axes = axes.flatten()

for ii in range(len(images)):
    axes[ii].hist(images[ii].ravel(), bins, [0, 256])

axes[0].set_title("Diamond")
axes[1].set_title("Diamond Scaled")
axes[2].set_title("Diamond Rotated 30")
axes[3].set_title("Diamond Rotated 90")
axes[4].set_title("Diamond Scaled and Rotated")
axes[5].set_title("Dugong")
axes[6].set_title("Dugong Scaled")
```

```
axes[7].set_title("Dugong Rotated 30")
axes[8].set_title("Dugong Rotated 90")
axes[9].set_title("Dugong Scaled and Rotated")


figure.tight_layout()
figure.subplots_adjust(left=0.05, wspace=0.5)
plt.show()

# ---------------- Harris Corner Detection ----------------

corners = []

for ii, img in enumerate(images):
    corners.append(cornerHarris(images[ii]))

#
↪  https://www.delftstack.com/howto/matplotlib/how-to-display-multiple-images-in-one-figure-correct

# width = 5
# height = 5
rows = 2
cols = 5
axes = []
figure = plt.figure()

for ii in range(rows * cols):
    axes.append(figure.add_subplot(rows, cols, ii + 1))
    plt.imshow(corners[ii])

axes[0].set_title("Diamond")
axes[1].set_title("Diamond Scaled")
axes[2].set_title("Diamond Rotated 30")
axes[3].set_title("Diamond Rotated 90")
axes[4].set_title("Diamond Scaled and Rotated")
axes[5].set_title("Dugong")
axes[6].set_title("Dugong Scaled")
axes[7].set_title("Dugong Rotated 30")
axes[8].set_title("Dugong Rotated 90")
axes[9].set_title("Dugong Scaled and Rotated")

figure.tight_layout()
plt.show()

filenames_harris = []

filenames_harris.append("Results\Harris\diamond_harris.png")
filenames_harris.append("Results\Harris\diamond_scaled_harris.png")
filenames_harris.append("Results\Harris\diamond_rotated_30_harris.png")
filenames_harris.append("Results\Harris\diamond_rotated_90_harris.png")
filenames_harris.append("Results\Harris\diamond_rotated_scaled_harris.png")
filenames_harris.append("Results\Harris\dugong_harris.png")
filenames_harris.append("Results\Harris\dugong_scaled_harris.png")
filenames_harris.append("Results\Harris\dugong_rotated_30_harris.png")
filenames_harris.append("Results\Harris\dugong_rotated_90_harris.png")
filenames_harris.append("Results\Harris\dugong_rotated_scaled_harris.png")

saveImages(corners, filenames_harris)
# ---------------- SIFT ----------------

sifts = []

for ii, img in enumerate(images):
    sifts.append(siftKeyPoints(images[ii]))

displayImages(sifts)
```

```
127
128 filenames_sift = []
129
130 filenames_sift.append("Results\SIFT\diamond_sift.png")
131 filenames_sift.append("Results\SIFT\diamond_scaled_sift.png")
132 filenames_sift.append("Results\SIFT\diamond_rotated_30_sift.png")
133 filenames_sift.append("Results\SIFT\diamond_rotated_90_sift.png")
134 filenames_sift.append("Results\SIFT\diamond_rotated_scaled_sift.png")
135 filenames_sift.append("Results\SIFT\dugong_sift.png")
136 filenames_sift.append("Results\SIFT\dugong_scaled_sift.png")
137 filenames_sift.append("Results\SIFT\dugong_rotated_30_sift.png")
138 filenames_sift.append("Results\SIFT\dugong_rotated_90_sift.png")
139 filenames_sift.append("Results\SIFT\dugong_rotated_scaled_sift.png")
140
141 saveImages(sifts, filenames_sift)
```

VII. Appendix 2 - Image_Features.py

```
import cv2 as cv
import numpy as np
from tools import *

diamond = cv.imread('Images/diamond.png')
dugong = cv.imread('Images/dugong.jpg')

imgs = [diamond, dugong]
scaled = rows = cols = translations = img_translations = M = rotated = [None] * 2

# This just extracts the top right diamond
cropped_diamond = diamond[45:70, 5:30]

# This extracts the mother and her calf
cropped_dugong = dugong[205:280, 383:433]

# ---------------- SCALE AND ROTATE ----------------

# Lets create a scaled version of the diamonds and dugongs
scaled_diamond = cv.resize(diamond, None, fx=1.5, fy=1.5, interpolation=cv.INTER_LINEAR)
rows_diamond, cols_diamond = scaled_diamond.shape[:2]

cropped_scaled_diamond = scaled_diamond[68:105, 8:45]

scaled_dugong = cv.resize(dugong, None, fx=1.5, fy=1.5, interpolation=cv.INTER_LINEAR)
rows_dugong, cols_dugong = scaled_dugong.shape[:2]

cropped_scaled_dugong = scaled_dugong[310:420, 575:650]

# Create 2 rotations, as SIFT is scale invarient up to 2.5x and 40 degree rotation
↪  invarient
rotated_90_diamond = cv.rotate(diamond, cv.ROTATE_90_CLOCKWISE)
rotated_90_dugong = cv.rotate(dugong, cv.ROTATE_90_CLOCKWISE)

cropped_90_diamond = rotated_90_diamond[8:30, 210:235]
cropped_90_dugong = rotated_90_dugong[380:430, 200:275]

rotated_30_diamond = rotate(diamond, 30)
rotated_30_dugong = rotate(dugong, 30)

print(rotated_30_dugong.shape)

cropped_30_diamond = rotated_30_diamond[122:142, 35:55]
cropped_30_dugong = rotated_30_dugong[390:475, 450:500]

# Hit 'em with both

images = [diamond, scaled_diamond, rotated_30_diamond, rotated_90_diamond, \
    dugong, scaled_dugong,  rotated_30_dugong,  rotated_90_dugong]

cropped = [cropped_diamond, cropped_scaled_diamond, cropped_30_diamond,
↪  cropped_90_diamond,\
        cropped_dugong, cropped_scaled_dugong, cropped_30_dugong, cropped_90_dugong]

# ---------------- SIFT ----------------

sifts_cropped = []

filenames_diamond = ["Results\Features\SIFT\cropped_diamond.png",
↪  "Results\Features\SIFT\cropped_scaled_diamond.png", \
    "Results\Features\SIFT\cropped_30_diamond.png",
    ↪  "Results\Features\SIFT\cropped_90_diamond.png"]
filenames_dugong = ["Results\Features\SIFT\cropped_dugong.png",
↪  "Results\Features\SIFT\cropped_scaled_dugong.png", \
```

```
    "Results\Features\SIFT\cropped_30_dugong.png",
    ↪  "Results\Features\SIFT\cropped_90_dugong.png"]

filenames_single = ["Results\Features\SIFT\cropped_single_diamond.png",
↪  "Results\Features\SIFT\cropped_scaled_single_diamond.png", \
    "Results\Features\SIFT\cropped_30_single_diamond.png",
    ↪  "Results\Features\SIFT\cropped_90_single_diamond.png", \
    "Results\Features\SIFT\cropped_single_dugong.png",
    ↪  "Results\Features\SIFT\cropped_scaled_single_dugong.png", \
    "Results\Features\SIFT\cropped_30_single_dugong.png",
    ↪  "Results\Features\SIFT\cropped_90_single_dugong.png"]

for ii, img in enumerate(images):
    sifts_cropped.append(siftKeyPoints(cropped[ii]))


for ii in range(len(cropped) - 4):
    siftKeyPointMatcher(cropped[ii], images[0], filenames_diamond[ii])

for ii in range(4, len(cropped)):
    siftKeyPointMatcher(cropped[ii], images[4], filenames_dugong[ii - 4])

saveImages(sifts_cropped, filenames_single)

# ---------------- HOG ---------------
```

VIII. Appendix 3 - **Object_Extraction.py**

[11] [12]

```python
import cv2 as cv
import numpy as np

# https://stackoverflow.com/questions/46441893/connected-component-labeling-in-python

diamond = cv.imread('Images/diamond.png', cv.IMREAD_GRAYSCALE)
dugong = cv.imread('Images/dugong.jpg')

# This was used to remove the green channel as an attempt to separate background and foreground further
# dugong[:,:,1] = np.zeros([dugong.shape[0], dugong.shape[1]])

# We wont gaussian blur the diamond as this is not noisy
diamond_thresh, diamond_binary = cv.threshold(diamond, 0, 255, cv.THRESH_BINARY_INV+cv.THRESH_OTSU)

# We'll threshold the dugong twice, once as a HSV image, as this seems to isolate the
# dugong from the background really well, and then once to convert it to a gray binary image
dugong = cv.cvtColor(dugong, cv.COLOR_BGR2HSV)
dugong_blur = cv.GaussianBlur(dugong, (5, 5), 0)

# A threshold of 130 works nicely with this image
dugong_thresh, dugong_binary = cv.threshold(dugong_blur, 130, 255, cv.THRESH_BINARY)

# Now we convert back to gray
# dugong_binary = cv.cvtColor(dugong_binary, cv.COLOR_HSV2BGR)
dugong_binary = cv.cvtColor(dugong_binary, cv.COLOR_BGR2GRAY)

# Apply a closing to ensure objects that are very close to eachother form one
kernel = np.ones((3, 3), np.uint8)
dugong_binary = cv.morphologyEx(dugong_binary, cv.MORPH_CLOSE, kernel)

# Final threshold to make the image a black and white binary image
dugong_thresh, dugong_binary = cv.threshold(dugong_binary, 0, 255, cv.THRESH_BINARY_INV+cv.THRESH_OTSU)

# We will use the built in CCL in opencv with 8 way connectivity
connectivity = 8
num_labels_diamond, labels_diamond, stats_diamond, centroids_diamond = cv.connectedComponentsWithStats(diamond_binary, connectivity, cv.CV_32S)
num_labels_dugong, labels_dugong, stats_dugong, centroids_dugong = cv.connectedComponentsWithStats(dugong_binary, connectivity, cv.CV_32S)

# https://www.programcreek.com/python/example/89340/cv2.connectedComponentsWithStats

# Map component labels to hue val, 0-179 is the hue range in OpenCV
label_hue_diamond = np.uint8(179 * labels_diamond/np.max(labels_diamond))
label_hue_dugong = np.uint8(179 * labels_dugong/np.max(labels_dugong))

blank_ch_diamond = 255 * np.ones_like(label_hue_diamond)
blank_ch_dugong = 255 * np.ones_like(label_hue_dugong)

# Merges the "colored components" to the blank shape to create the image
labeled_img_diamond = cv.merge([label_hue_diamond, blank_ch_diamond, blank_ch_diamond])
labeled_img_dugong = cv.merge([label_hue_dugong, blank_ch_dugong, blank_ch_dugong])

# Converting to BGR
labeled_img_diamond = cv.cvtColor(labeled_img_diamond, cv.COLOR_HSV2BGR)
labeled_img_dugong = cv.cvtColor(labeled_img_dugong, cv.COLOR_HSV2BGR)

# Set the background to be black
labeled_img_diamond[label_hue_diamond==0] = 0
```

```
labeled_img_dugong[label_hue_dugong==0] = 0

print("Number of labels for diamond: " + str(num_labels_diamond))
for ii in range(num_labels_diamond):
    area = stats_diamond[ii, cv.CC_STAT_AREA]
    width = stats_diamond[ii, cv.CC_STAT_WIDTH]
    height = stats_diamond[ii, cv.CC_STAT_HEIGHT]

    string = "Label: " + str(ii + 1) + \
             "\n    Area: " + str(area) + \
             "\n    Height: " + str(height) + \
             "\n    Width: " + str(width) + \
             "\n ---------------------------------------- \n"

    print(string)

print("Number of labels for dugong: " + str(num_labels_dugong))
for ii in range(num_labels_dugong):
    area = stats_dugong[ii, cv.CC_STAT_AREA]
    width = stats_dugong[ii, cv.CC_STAT_WIDTH]
    height = stats_dugong[ii, cv.CC_STAT_HEIGHT]

    string = "Label: " + str(ii + 1) + \
             "\n    Area: " + str(area) + \
             "\n    Height: " + str(height) + \
             "\n    Width: " + str(width) + \
             "\n ---------------------------------------- \n"

    print(string)

cv.imshow("Labeled diamond", labeled_img_diamond)
cv.imshow("Labeled dugong", labeled_img_dugong)
cv.waitKey()
```

IX. Appendix 4 - Image_Segmentation.py

[13] [14]

```python
import cv2 as cv
import numpy as np

# https://docs.opencv.org/master/d1/d5c/tutorial_py_kmeans_opencv.html
# https://www.thepythoncode.com/article/kmeans-for-image-segmentation-opencv-python

diamond = cv.imread('Images/diamond.png')
dugong = cv.imread('Images/dugong.jpg')

# Removing colour channels to make the dugong stand out more from the background
# This is to try different combiniation of colour channels to see what works best
# [0 1 2] is [B G R]

# dugong[:,:,0] = np.zeros([dugong.shape[0], dugong.shape[1]])
# dugong[:,:,1] = np.zeros([dugong.shape[0], dugong.shape[1]])
# dugong[:,:,2] = np.zeros([dugong.shape[0], dugong.shape[1]])

kernel = np.ones((2,2))

dugong = cv.morphologyEx(dugong, cv.MORPH_OPEN, kernel)

diamond = cv.cvtColor(diamond, cv.COLOR_BGR2HSV)
dugong = cv.cvtColor(dugong, cv.COLOR_BGR2HSV)

Z_diamond = diamond.reshape((-1, 3))
Z_diamond = np.float32(Z_diamond)
Z_dugong = dugong.reshape((-1, 3))
Z_dugong = np.float32(Z_dugong)

criteria = (cv.TERM_CRITERIA_EPS + cv.TERM_CRITERIA_MAX_ITER, 10, 0.2)
K = 3
ret_diamond, labels_diamond, center_diamond = cv.kmeans(Z_diamond, K, None, criteria, 10,
↪  cv.KMEANS_RANDOM_CENTERS)
ret_dugong, labels_dugong, center_dugong = cv.kmeans(Z_dugong, K, None, criteria, 10,
↪  cv.KMEANS_RANDOM_CENTERS)

center_diamond = np.uint8(center_diamond)
center_dugong = np.uint8(center_dugong)

labels_diamond = labels_diamond.flatten()
labels_dugong = labels_dugong.flatten()

segmented_diamond = center_diamond[labels_diamond.flatten()]
segmented_dugong = center_dugong[labels_dugong.flatten()]

color_diamond = 1
segmented_diamond[labels_diamond == color_diamond] = [0, 0, 0]

# Make the water around the dugong segment black
color_dugong_1 = 1
segmented_dugong[labels_dugong == color_dugong_1] = [0, 0, 0]

color_dugong_2 = 0
segmented_dugong[labels_dugong == color_dugong_2] = [0, 0, 0]

segmented_diamond = segmented_diamond.reshape(diamond.shape)
segmented_dugong = segmented_dugong.reshape(dugong.shape)

cv.imshow("Diamond: ", segmented_diamond)
cv.imshow("Dugong: ", segmented_dugong)
cv.waitKey()
```

REFERENCES

REFERENCES

[1] A. Rosebrock, "Local binary patterns with python opencv," *Pyimagesearch*. [Online]. Available: https://www.pyimagesearch.com/2015/12/07/local-binary-patterns-with-python-opencv/

[2] S. Mallick, "Histogram of oriented gradients," *Learn OpenCV*. [Online]. Available: https://www.learnopencv.com/histogram-of-oriented-gradients/

[3] "Introduction to sift (scale invariant feature transform)," *Medium, url =*.

[4] "Histogram calculation," *Open Source Computer Vision*. [Online]. Available: https://docs.opencv.org/3.4/d8/dbc/tutorial$_histogram_calculation.html$

[5] "Harris corner detection," *Open Source Computer Vision*. [Online]. Available: https://docs.opencv.org/3.4/dc/d0d/tutorial$_py_features_harris.html$

[6] "Introduction to sift (scale-invariant feature transform)," *Open Source Computer Vision*. [Online]. Available: https://docs.opencv.org/3.4/da/df5/tutorial$_py_sift_intro.html$

[7] "Feature matching," *Open Source Computer Vision*. [Online]. Available: https://docs.opencv.org/3.4/dc/dc3/tutorial$_py_matcher.html$

[8] R. Perrone. Rotate an image without cropping in opencv in c++. [Online]. Available: https://stackoverflow.com/questions/22041699/rotate-an-image-without-cropping-in-opencv-in-c

[9] "How to display multiple images in one figure correctly in matplotlib," *DelftStack*. [Online]. Available: https://www.delftstack.com/howto/matplotlib/how-to-display-multiple-images-in-one-figure-correctly-in-matplotlib/

[10] J. H. et. al, "The histogram (hist) function with multiple data sets," *Matplotlib*. [Online]. Available: https://matplotlib.org/3.1.1/gallery/statistics/histogram$_multihist.html$

[11] Alkasm and YoniChechik. connected component labeling in python. [Online]. Available: https://stackoverflow.com/questions/46441893/connected-component-labeling-in-python

[12] YoongiKim, "Python cv2.connectedcomponentswithstats() examples," *ProgramCreek*. [Online]. Available: https://www.programcreek.com/python/example/89340/cv2.connectedComponentsWithStats

[13] "K-means clustering in opencv," *Open Source Computer Vision*. [Online]. Available: https://docs.opencv.org/master/d1/d5c/tutorial$_py_kmeans_opencv.html$

[14] A. Rockikz, "How to use k-means clustering for image segmentation using opencv in python," *The Python Code*. [Online]. Available: https://www.thepythoncode.com/article/kmeans-for-image-segmentation-opencv-python