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:	Yap	Student ID:	19257442	
Other name(s):	Kay Men Yap			
Unit name:	Machine Perception	Unit ID:	COMP3007	
Lecturer / unit coordinator:	Dr. Senjian An	Tutor:	Bradley Ezard	
Date of submission:	04/10/2020	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: Kay Men Yap Date of signature: 03/10/2020

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

MP Assignment 1

Kay Men Yap
Student No: 19257442
Curtin University
Perth, Australia
19257442@student.curtin.edu.au

I. Task 1

A. Part i

Image histogram is scale invariant and rotation invariant as scaling wouldn't change the proportion of pixels of various intensity and rotating the image will not change the proportion of pixels of various intensity so the shape of the histogram will visibly look very similar.

Harris corner is scale variant as changing the size of the image will change the eigenvalues detected by the sliding window when the sliding window goes through the same image patch when compared to the original image. Harris corner will detect some corners as edges for enlarged images and some edges as corners for scaled down images as the Harris corner sliding window size doesn't change but the image size has changed so the sliding window of Harris corner might not detect correctly where an edge or corner appear. Harris corner is rotation invariant as rotating the image doesn't change the eigenvalues detected by the sliding window so it can still detect the same edges and corners as the original image regardless of orientation.

SIFT is scale invariant up to 2-2.5 scale as per lecture 4 of MP as it uses Scale-Space Extrema Detection to account of different scales of the image and is rotation invariant up to 40 degrees as stated in Lecture 4 slides of MP but past 40 degrees it is still somewhat rotation invariant as it will still pick up all of the important keypoint features but also some additional keypoints because it uses Orientation assignment to account for different orientations of the image.

B. Part ii

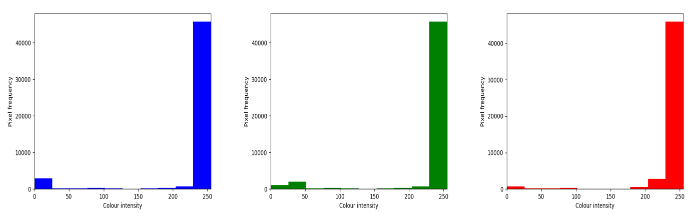

Fig. 1. Histogram of original diamond image

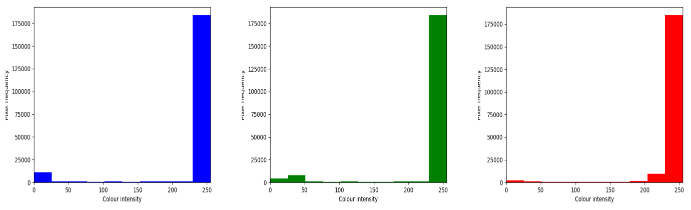

Fig. 2. Histogram of 200% scaled diamond image

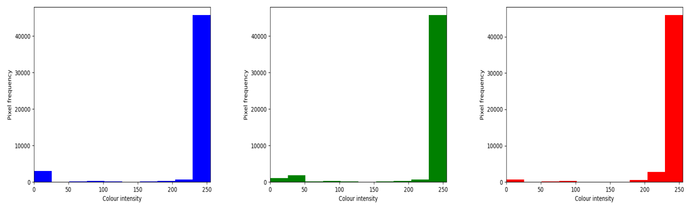

Fig. 3. Histogram of rotated 90 degrees diamond image

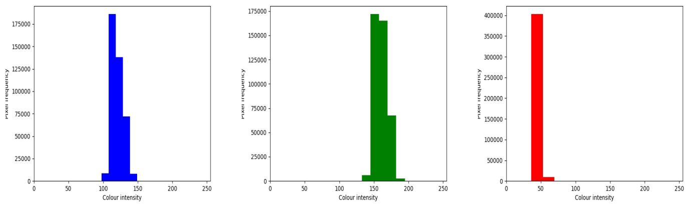

Fig. 4. Histogram of original Dugong image

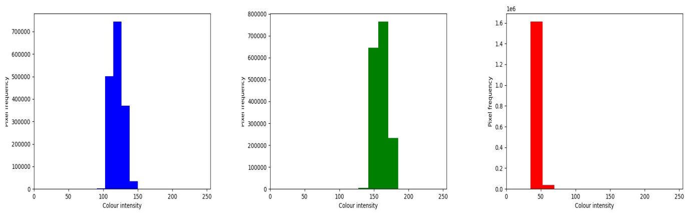

Fig. 5. Histogram of 200% scaled dugong image

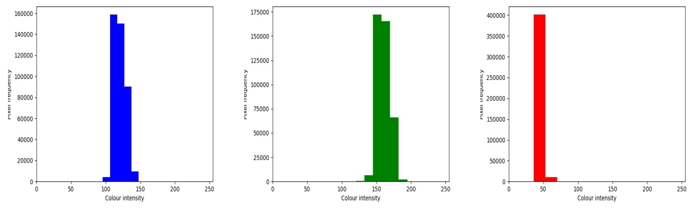

Fig. 6. Histogram of rotated 90 degrees dugong image

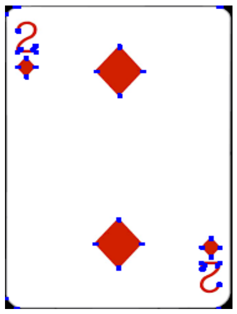

Fig. 7. Harris corner of original diamond image

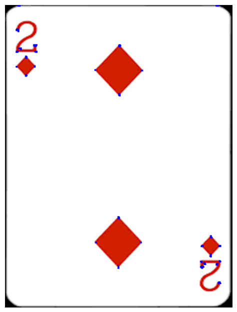

Fig. 8. Harris corner of scaled 200% diamond image

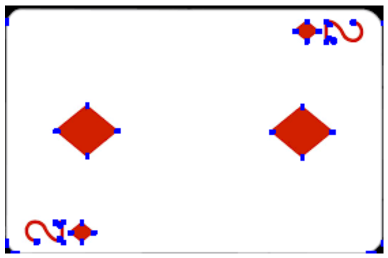

Fig. 9. Harris corner of rotated diamond image

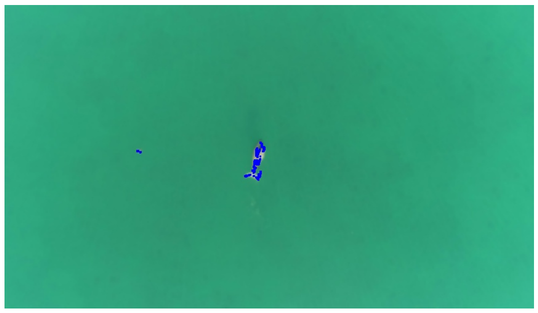

Fig. 10. Harris corner of original Dugong image

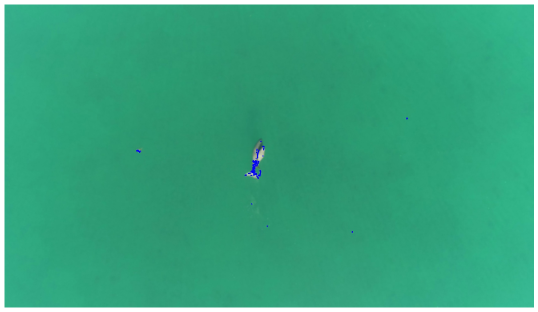

Fig. 11. Harris corner of scaled 200% Dugong image

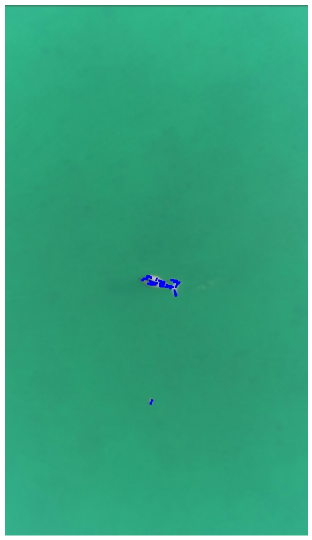

Fig. 12. Harris corner of rotated Dugong image

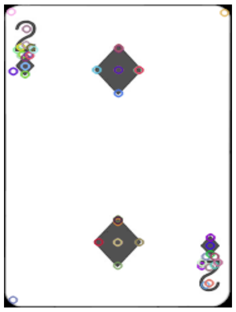
Fig. 13. SIFT of original diamond image

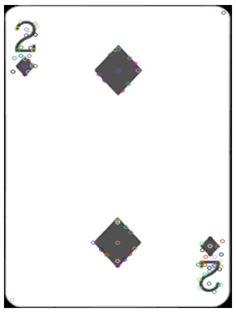
Fig. 14. SIFT of scaled 200% diamond image

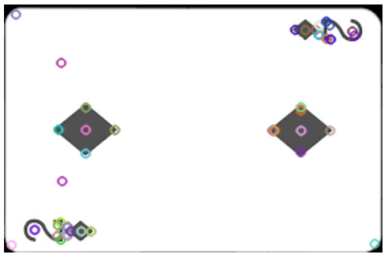
Fig. 15. SIFT of rotated diamond image

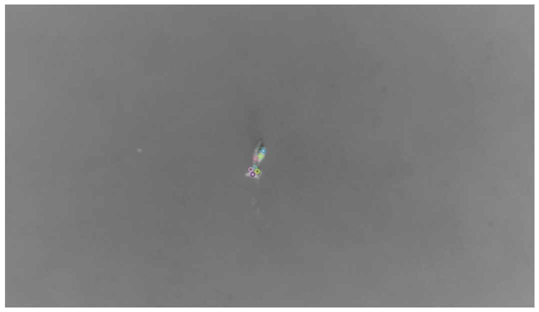
Fig. 16. SIFT of original Dugong image

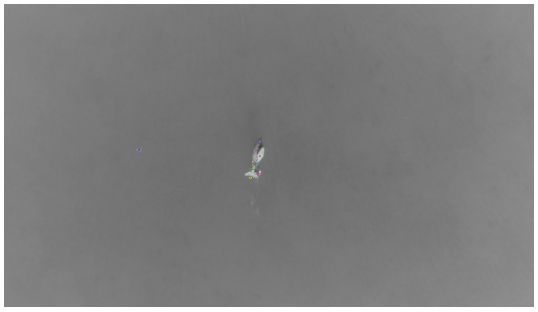
Fig. 17. SIFT corner of scaled 200% Dugong image

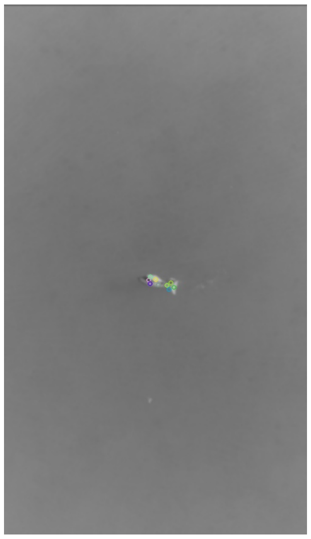
Fig. 18. SIFT corner of rotated Dugong image

C. Part iii

Histogram is evidently scale invariant and rotation invariant as the proportions of the histograms in the histogram figures in Part ii of Task 1 do not change significantly when you scale or rotate the image except for the first bin of the histogram for the images rotated at an angle that is not 90 degrees as opencv's implementation of rotation will pad black pixels to the rotated image to have a square image which is accounted for when determining if the results align with what is mentioned in Part i of Task 1. Also, the Dugong image's histograms are not 100% exactly the same proportion due to noise and I have already applied GaussianBlur to Dugong image to reduce the noisiness and ensure Dugong image and its scaled and rotated counterparts yielded similar shaped histograms.

Harris corner is evidently scale variant as the corner keypoint in the top right and bottom left of the 200& scaled diamond image is not detected as a keypoint but it is detected as a keypoint in the original diamond image. Also, in Figure 11, there is additional corners detected by Harris Corner when compared to the original Harris Corner image.

Harris corner is evidently rotation invariant as the same number of keypoints are detected and located at the same keypoints of the original diamond image and rotated 90 diamond image. The rotated 45 diamond image doesn't have the keypoints detected at the corners of the diamond that do not have the "2" or diamond shape which might be because of the black pixel background around the card due to how rotation without cropping works in opencv. Similar results are present in the Figure 10 and 12

SIFT is evidently scale and rotation invariant to a certain degree as for the diamond image, all of the important features are picked up in the original, rotated and scaled diamond image on top of the extra keypoints detected at the 2 center diamonds in the Figure 14 and additional keypoints at the left side of Figure 15 between the left big diamond and the top and bottom of the figure. In Figure 16 and 18, the same amount of keypoints at nearly the same locations are detected at the Dugong part of the image whereas Figure 17 has the same keypoints as the 2 previous Figures mentioned and also a keypoint detected at the seaweed so SIFT is scaled invariant to a certain degree.

II. Task 2

A. *Part i*

Main steps of LBP are to create an blank image of the same size as the original image then in a 3x3 sliding window, the center pixel of the window is used to threshold the neighouring pixels where if the pixel value of the center pixel is greater than or equal to the pixel of neighbour pixel, the neighbour pixel will have a value of 1 or else get the value of 0. Then, the thresholded value of the neighbour pixels are used to construct an 8-bit binary number and that will be the value stored at the coordinate of the center pixel in the blank image. The order of thresholding the neighbour pixels and constructing the 8-bit binary number doesn't matter as long as the ordering is consistently used for every pixel [2]. Then continue until LBP is performed on every pixel in the image. The once blank image is the output of LBP once LBP is finish performing its task.

Main steps of HOG are to preprocess the image if needed, then calculate the horizontal and vertical gradients of the image using a kernel such as the Sobel kernel size of 1, and then calculate the histogram of gradients by first using the horizontal and vertical gradients in 8x8 cells of the image to determine the gradient magnitudes and gradient directions in the cell and afterwards using those gradient magnitudes and directions to construct the histogram of gradients of 8 bins that contribute to 180 degrees of direction by voting to the respective bins using the gradient direction and contributing the gradient magnitude to the bins voted. The histogram of gradients calculation are repeated for the rest of the images. Then, block normalization in 16x16 is performed on the histograms of gradients. Finally, all of the normalized histograms of gradients are concatenated into a single vector and outputted as the Histogram of Oriented Gradients result. [3]

Main steps of SIFT are to perform the Scale-space Extrema Detection that involves Difference of Gaussian to account for different scales of the image as to make SIFT scale invariant, then keypoint localization is performed on the image to remove any low-contrast keypoints and edge keypoints to only keep the strong keypoints. Afterwards, orientation assignment is performed on the image to account for different orientations of the image thus making SIFT rotation invariant to a certain degree. Keypoint descriptors are then generated on those detected keypoints by examining in a 16x16 window centered on the keypoint and these descriptors will have robustness to illumination changes, rotation and scale due to the previous steps taken. Finally, the concatened vector of all keypoint descriptors will be the output of SIFT that can be used to match keypoints in other images. [1]

Common components of LBP, HOG and SIFT is that they all utilise sliding windows to calculate values in a smaller window of the image and they utilise the pixel values of the neighbouring pixels when considering a pixel in the sliding window in the calculation.

LBP is different to HOG and SIFT as that LBP doesn't take into account of the gradient of the pixels and only the pixel values when performing its calculation whereas HOG and SIFT take into account of gradient to show orientation. HOG is differnt to LBP and SIFT as HOG is a regional descriptor [4] since it takes into account of the entire image when generating descriptors whereas LBP and SIFT are local descriptors as LBP generate local patterns.

Advantage of LBP is that it is computationally the fastest amongst LBP, HOG and SIFT, and can capture patterns at a very small scale of 3x3 window size**??**. Disadvantages of LBP is that it is scale and rotation variant because the output of LBP for a rotated or scaled image will be different when compared to the original image as the algorithm of LBP doesn't account for scale or orientation of image, and the scale used for finding patterns cannt be changed for LBP as it is fixed to 3x3 **??** so it is not possible to find patterns of larger scales with LBP.

Advatanges of HOG are that it is geometric and photometric invariant so it is widely used in detection of humans, animals and vehicles as it is not affected by clothing, illumination, variations in pose, appearance and partial occlusions as mentioned in Lecture 4 of MP. The disadvantage of HOG is that it is not rotation or scale invariant since it will generate different HOG descriptor if a scaled and/or rotated image was provided.

Advantage of SIFT is that it is scale and rotation invariant because it uses Scale-Space Extrema Detection to account of different scales of the image and Orientation assignment to account for the any rotation done to the image [1]. The disadvantages of SIFT are that you require a license to use it in commercial use due to SIFT being licensed, it is the most computationally intensive algorithm amongst LBP, HOG and SIFT as SIFT involves Scale-Space Extrema Detection and orientation assignment, and it is only scale and rotation invariant to a certain degree and unable to be invariant to extreme scales or rotations as mentioned in lecture 4 of MP.

B. *Part ii*

	Scaled 2x	Rotated 90
Keypoint	0.0%	0.0%
Full Image	212.71%	157.14%

Table 1: HOG variations of diamond image when compared to original image

	Scaled 2x	Rotated 90
Keypoint	0.0%	0.0%
Full Image	41.34%	38.74%

Table 2: HOG variations of Dugong image when compared to original image

	Scaled 2x	Rotated 90
Keypoint	1.37%	0.0%

Table 3: SIFT variations of diamond image when compared to original image

	Scaled 2x	Rotated 90
Keypoint	3.01%	1.30%

Table 4: SIFT variations of Dugong image when compared to original image

C. Part iii

The keypoints extracted are using SIFT and are sorted by ascending distance to ensure the match is the same keypoint in both images. As per the 0.0% figures in the HOG variation tables, I wasn't able to get the absolute difference in variation between original and scaled or rotated image at a particular keypoint and I wasn't able to figure why it was always output a result of 0.0 even when cv2.norm(H0). Hence, I also included the absolute differences of HOG results using the entire image between original and scaled or rotated image in the HOG variations table just to have some figures available for comparison between HOG and SIFT. SIFT variations in both images are all below 5% whereas HOG variations in both images are all above 30% which evidently shows that SIFT's advantage over HOG is that SIFT is scale and rotation invariant when generating feature descriptors whereas HOG isn't.

III. Task 3

A. Implementation reasoning

I only used the red colour channel of BGR colour space and Gaussian blur with kernel size of 5,5 to get a somewhat accurate binary image that evidently shows the dugong and the seaweed as without these methods, some parts of the ocean are included in the binary image. The reason I only used the red colour channel in Task 3 for Dugong image is that in Task 4, only using the red colour channel of BGR yielded valid image segmentation and the way I found out about that in Task 4 was through brute forcing through different combinations of colour channels until I got valid image segmentation. I used cv.THRESH_BINARY_INV for diamond as the card background is white and cv.THRESH_INV for dugong as dugong doesn't have a white background.

B. Diamond

Statistics: Number of components: 7
The respective area of each components are as follows: 1331, 157 893 143, 890, 143, 156

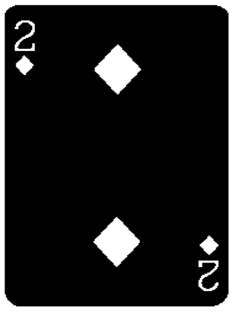
Fig. 19. Binary image of diamond image

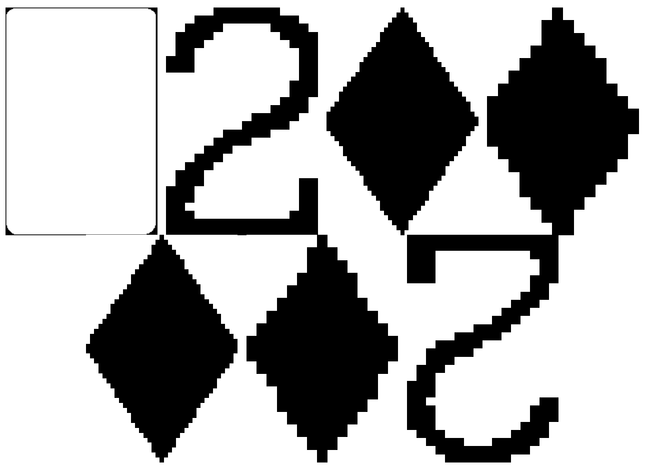
Fig. 20. Seven components extracted from binary image

C. Dugong

Statistics: Number of components: 2
The respective area of each components are as follows: 677, 16
First component being the dugong and second component being the seaweed

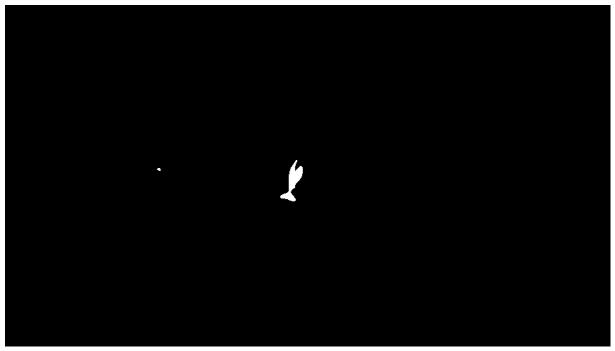
Fig. 21. Binary image of diamond image

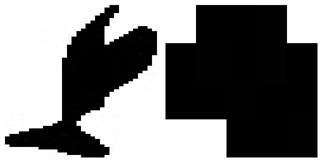

Fig. 22. Two components extracted from binary image

IV. Task 4

A. Diamond

Below is the result of performing K-Means Image Segmentation on the diamond2.png image with cluster size of 2 using 3 different methods.

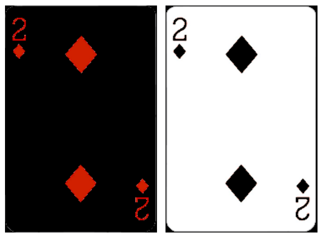

Fig. 23. Diamond image with BGR

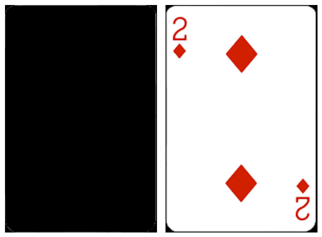

Fig. 24. Diamond image with only red channel of BGR

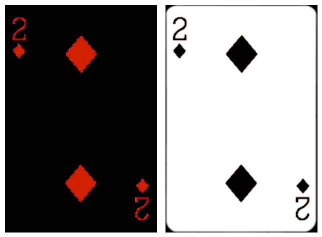

Fig. 25. Diamond image with HS channels of HSV colour space

B. Dugong

Below is the result of performing K-Means Image Segmentation on the Dugong.jpg image with cluster size of 2 using 3 different methods.

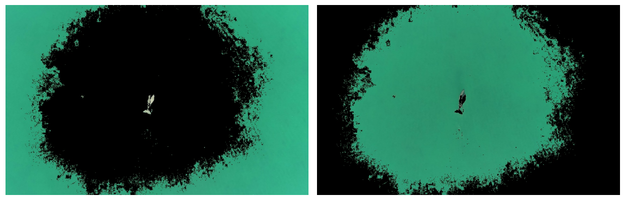

Fig. 26. Dugong image with BGR

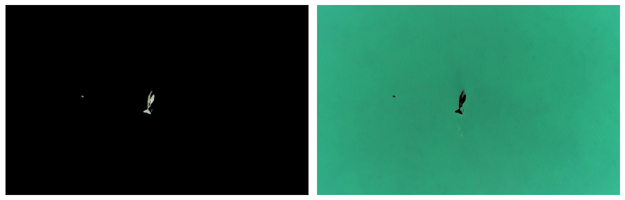

Fig. 27. Dugong image with only red channel of BGR

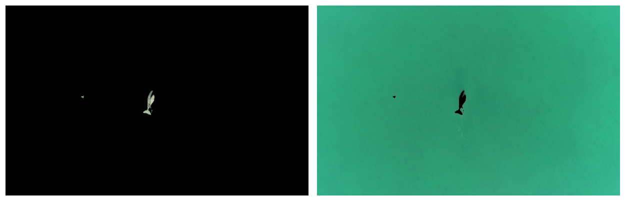

Fig. 28. Dugong image with HS channels of HSV colour space

C. Different methods used

The reasoning behind a cluster size of 2 is because in both images, there is 2 distinct colours to segment by which are white and red for diamond image and white and green for dugong image. The seaweed and dugong will be in the same cluster since they are both similar in color.

I have tried using the full BGR colour space, only the red channel of BGR colour space and the hue and saturation channel of the HSV colour space. The way I found out about these 3 methods is through brute forcing through different combinations of colour channels of BGR colour space then doing the same with the HSV colour space until i got 3 different methods that yielded 3 different results overall. Full BGR colour space doesnt work with Dugong as the parts of the ocean is in the same cluster as the Dugong but it works with diamond image since it has very distinct colours so this method will work on very different colours of objects. Only red channel works well with Dugong image but doesnt work with the diamond image as both the background, diamond and number 2 all have a high red value because it is hard to segment the white card background with the image, so this method won't work with any image that has red only objects with other objects that have high red values as well. Hue and saturation channel of HSV colour space works on both images as both image' objects have a different hue and saturation value when compared to the their respective background, and this method will work well with objects and background that have a distinctively different hue and saturation value.

References

References

[1] OpenCV. OpenCV: Introduction to SIFT (Scale-Invariant Feature Transform). https://docs.opencv.org/3.4.2/da/df5/tutorial_py_sift_intro.html (accessed 30/9/2020).
[2] Adrian Rosebrock. Local Binary Patterns with Python & OpenCV - PyImageSearch. https://www.pyimagesearch.com/2015/12/07/local-binary-patterns-with-python-opencv/ (accessed 30/9/2020).
[3] Satya Mallick. Histogram of Oriented Gradients — Learn OpenCV. https://www.learnopencv.com/histogram-of-oriented-gradients/ (accessed 03/10/2020).
[4] Tony Lindeberg. Scale Invariant Feature Transform -Scholarpedia. http://www.scholarpedia.org/article/Scale_Invariant_Feature_Transform#Histograms_of_oriented_gradients_.28HOG.29 (accessed 30/9/2020).

Appendix

A. *Appendix 1*

Resizing image python script

```python
import cv2 as cv
import numpy as np
import sys
from pathlib import Path

'''
Conains code copied and modified from https://docs.opencv.org/3.4.2/da/d6e/
    tutorial_py_geometric_transformations.html
Last accessed 30/9/2020
'''
def resize(image, fileName, fileExt, scale):
        resized = cv.resize(image, None, fx=scale/100, fy=scale/100, interpolation=cv.INTER_CUBIC)
        cv.imwrite(fileName + "-scale-" + str(scale) + fileExt,resized)


def main():
    if(len(sys.argv) == 3):
        path = Path(sys.argv[1])
        file = sys.argv[1]
        image = cv.imread(file)
        resize(image, path.stem, path.suffix, int(sys.argv[2]))
    else:
        print("Please run the program with 'python resize.py file scale'")

if(__name__ == '__main__'):
    main()
```

Rotate image python script

```python
import cv2 as cv
import numpy as np
import sys
from pathlib import Path

'''
Contains code copied from https://stackoverflow.com/questions/22041699/rotate-an-image-without-cropping-in-
    opencv-in-c/37347070#37347070
Last accessed on 30/09/2020
'''
def rotate(image, fileName, fileExt, angle):
    height,width = image.shape[:2]
    center = (width/2, height/2)
    rot = cv.getRotationMatrix2D(center, angle, 1.0)

    abs_cos = abs(rot[0,0])
    abs_sin = abs(rot[0,1])

    bound_w = int(height * abs_sin + width * abs_cos)
    bound_h = int(height * abs_cos + width * abs_sin)

    rot[0, 2] += (bound_w/2) - center[0]
    rot[1, 2] += (bound_h/2) - center[1]

    rotated = cv.warpAffine(image, rot, (bound_w, bound_h))

    cv.imwrite(fileName + "-rotated-by-" + str(angle) + fileExt, rotated)
```

```python

def main():
    if(len(sys.argv) == 3):
        path = Path(sys.argv[1])
        file = sys.argv[1]
        image = cv.imread(file)
        angle = int(sys.argv[2])
        rotate(image, path.stem, path.suffix, angle)
    else:
        print("Please run the program with 'python rotate.py file scale'")

if(__name__ == '__main__'):
    main()
```

Task 1 script

```python
import cv2 as cv
import numpy as np
import sys
from pathlib import Path
from matplotlib import pyplot as plt
import os

'''
Contains code copied and adapted from https://docs.opencv.org/3.4.2/d1/db7/tutorial_py_histogram_begins.
    html
Last accessed on 03/10/2020
'''
def histogram(image, fileName, fileExt, folder):
    color = ('b','g','r')
    for i,j in enumerate(color):
        plt.hist(image[:,:,i].reshape(-1), 10, color=j)
        #plt.hist(image[:,:,i].ravel(), 10, color=j)
        plt.xlabel('Colour intensity')
        plt.ylabel('Pixel frequency')
        plt.xlim(0,255)
        plt.savefig(folder + fileName + "-histogram-" + j + fileExt)
        plt.clf()

'''
Contains code copied and adapted from https://docs.opencv.org/3.4.2/dc/d0d/tutorial_py_features_harris.html
Last accessed on 03/10/2020
'''
def harrisDetection(image, fileName, fileExt, folder):
    gray = cv.cvtColor(image, cv.COLOR_BGR2GRAY)
    gray = np.float32(gray)
    dst = cv.cornerHarris(gray, 2, 3, 0.04)
    dst = cv.dilate(dst, None)
    image[dst>0.01*dst.max()] = [255,0,0]
    cv.imwrite(folder + fileName + "-harris" + fileExt, image)

'''
Contains code copied and adapted from https://docs.opencv.org/3.4.2/da/df5/tutorial_py_sift_intro.html
Last accessed on 03/10/2020
'''
def sift(image, fileName, fileExt, folder):
    gray = cv.cvtColor(image, cv.COLOR_BGR2GRAY)

    siftObj = cv.xfeatures2d.SIFT_create()
    kp, des = siftObj.detectAndCompute(gray,None)
    textFile = open(folder + fileName + "-sift-descriptors.txt","w")
    np.set_printoptions(threshold=np.inf)
    textFile.write(np.array_str(des))
    image = cv.drawKeypoints(gray, kp, image)
    cv.imwrite(folder + fileName + "-sift-keypoints" + fileExt, image)
    cv.imwrite(folder + fileName + "-sift-descriptors" + fileExt, des)

def main():
    if(len(sys.argv) == 2):
        path = Path(sys.argv[1])
        folder = "task1/"
        if not os.path.exists(folder):
            os.mkdir(folder)
        file = sys.argv[1]
        image = cv.imread(file)
        if(path.stem.startswith("Dugong")):
            image = cv.GaussianBlur(image, (7,7),0)
            #image = cv.medianBlur(image, 5)
        histogram(image.copy(), path.stem, path.suffix, folder)
        harrisDetection(image.copy(), path.stem, path.suffix, folder)
        sift(image.copy(), path.stem, path.suffix, folder)


    else:
        print("Please run the program with 'python task1.py file'")

if(__name__ == '__main__'):
    main()
```

B. *Appendix 2*

Resizing image python script

```
import cv2 as cv
import numpy as np
import sys
from pathlib import Path

'''
Conains code copied and modified from https://docs.opencv.org/3.4.2/da/d6e/
    tutorial_py_geometric_transformations.html
Last accessed 30/9/2020
'''
def resize(image, fileName, fileExt, scale):
        resized = cv.resize(image, None, fx=scale/100, fy=scale/100, interpolation=cv.INTER_CUBIC)
        cv.imwrite(fileName + "-scale-" + str(scale) + fileExt,resized)


def main():
    if(len(sys.argv) == 3):
        path = Path(sys.argv[1])
        file = sys.argv[1]
        image = cv.imread(file)
        resize(image, path.stem, path.suffix, int(sys.argv[2]))
    else:
        print("Please run the program with 'python resize.py file scale'")

if(__name__ == '__main__'):
    main()
```

Rotate image python script

```
import cv2 as cv
import numpy as np
import sys
from pathlib import Path

'''
Contains code copied from https://stackoverflow.com/questions/22041699/rotate-an-image-without-cropping-in-
    opencv-in-c/37347070#37347070
Last accessed on 30/09/2020
'''
def rotate(image, fileName, fileExt, angle):
    height,width = image.shape[:2]
    center = (width/2, height/2)
    rot = cv.getRotationMatrix2D(center, angle, 1.0)

    abs_cos = abs(rot[0,0])
    abs_sin = abs(rot[0,1])

    bound_w = int(height * abs_sin + width * abs_cos)
    bound_h = int(height * abs_cos + width * abs_sin)

    rot[0, 2] += (bound_w/2) - center[0]
    rot[1, 2] += (bound_h/2) - center[1]

    rotated = cv.warpAffine(image, rot, (bound_w, bound_h))

    cv.imwrite(fileName + "-rotated-by-" + str(angle) + fileExt, rotated)
```

```python

def main():
    if(len(sys.argv) == 3):
        path = Path(sys.argv[1])
        file = sys.argv[1]
        image = cv.imread(file)
        angle = int(sys.argv[2])
        rotate(image, path.stem, path.suffix, angle)
    else:
        print("Please run the program with 'python rotate.py file scale'")

if(__name__ == '__main__'):
    main()
```

Task 2 script

```python
import cv2 as cv
import numpy as np
import sys
from pathlib import Path
from matplotlib import pyplot as plt
import os


'''
Contains code copied and adapted from
https://stackoverflow.com/questions/6090399/get-hog-image-features-from-opencv-python/31042366#31042366
Last accessed on 03/10/2020
'''
def hog(image1, image2, fileName, fileExt, folder):
    gray1 = cv.cvtColor(image1, cv.COLOR_BGR2GRAY)
    gray2 = cv.cvtColor(image2, cv.COLOR_BGR2GRAY)
    winSize = (64,128)
    blockSize = (16,16)
    blockStride = (8,8)
    cellSize = (8,8)
    nbins = 9
    hog = cv.HOGDescriptor(winSize,blockSize,blockStride,cellSize,nbins)

    winStride = (8,8)
    padding = (8,8)
    locations = ((10,20),)
    h1 = hog.compute(gray1,winStride,padding,locations)
    h2 = hog.compute(gray2,winStride,padding,locations)
    diff = cv.norm(h1-h2)
    original = cv.norm(h1)
    if(original > 0):
        ratio = diff/original*100
    else:
        ratio = diff
    print("HOG: " + str(ratio) + "%")

'''
Contains code copied and adapted from https://docs.opencv.org/master/dc/dc3/tutorial_py_matcher.html
and https://docs.opencv.org/3.4.2/da/df5/tutorial_py_sift_intro.html
Last accessed on 03/10/2020
'''
def siftCompare(image1, image2, fileName, fileExt, folder):
    gray1 = cv.cvtColor(image1, cv.COLOR_BGR2GRAY)
    gray2 = cv.cvtColor(image2, cv.COLOR_BGR2GRAY)

    siftObj = cv.xfeatures2d.SIFT_create()
    kp1, des1 = siftObj.detectAndCompute(gray1,None)
    kp2, des2 = siftObj.detectAndCompute(gray2,None)

    bf = cv.BFMatcher()
    matches = bf.match(des1,des2)
    # Sort the matches to get the best match to ensure that we get the same keypoint from the 2 images
    matches = sorted(matches, key = lambda x:x.distance)
    result = cv.drawMatches(image1,kp1,image2,kp2,matches[:1], None, flags=2)
    cv.imwrite(folder + fileName + "-sift-matches" + fileExt, result)

    # Adjust ratio for cropping scaled images
    if("scale" in fileName):
        ratio = int(fileName.split("-")[2])
        offset = int(25 * (ratio / 100))
    else:
        offset = 25

    # Pick the best match that you can crop a square from the center of the keypoint
    index = 0
    for i in range (0,len(matches)):
        x1,y1 = [int(j) for j in kp1[matches[i].queryIdx].pt]
        x2,y2 = [int(j) for j in kp2[matches[i].trainIdx].pt]
        if(x1 > offset and y1 > offset):
            index = i
            break

    cropped1 = image1[y1-25:y1+25, x1-25:x1+25]
    cropped2 = image2[y2-offset:y2+offset, x2-offset:x2+offset]
    var1 = des1[matches[index].queryIdx]
    var2 = des2[matches[index].trainIdx]
    diff = cv.norm(var1 - var2)
    original = cv.norm(var1)
    print("SIFT variation: " + str(diff/original*100) + "%")
    return cropped1.copy(), cropped2.copy()

def main():
    if(len(sys.argv) == 3):
        path = Path(sys.argv[2])
        folder = "task2/"
        if not os.path.exists(folder):
            os.mkdir(folder)
        file = sys.argv[1]
        image1 = cv.imread(file)
        file2 = sys.argv[2]
        image2 = cv.imread(file2)
        cropped1, cropped2 = siftCompare(image1.copy(), image2.copy(), path.stem, path.suffix, folder)
        cv.imwrite(folder + path.stem + "-cropped1" + path.suffix, cropped1)
        cv.imwrite(folder + path.stem + "-cropped2" + path.suffix, cropped2)
        print("Keypoint cropped image ", end='')
        hog(cropped1.copy(), cropped2.copy(), path.stem, path.suffix, folder)
        print("Full image ", end='')
        hog(image1.copy(), image2.copy(), path.stem, path.suffix, folder)
    else:
        print("Please run the program with 'python task2.py file file2'")

if(__name__ == '__main__'):
    main()
```

C. Appendix 3

```
import cv2 as cv
import numpy as np
import sys
from pathlib import Path
import os

'''
Contains code copied and adapted from https://docs.opencv.org/master/d7/d4d/tutorial_py_thresholding.html
Last accessed on 30/09/2020
'''
def connectedComponentLabelling(image, fileName, fileExt, folder):
    # Best result when only using the red channel colours for Dugong image
    if(fileName.startswith("Dugong")):
        # Sets the blue colour channel to zero
        image[:,:,0] = 0
        # Sets the green colour channel to zero
        image[:,:,1] = 0
        # Median blur gets a 'thinner' binary image
        #image = cv.medianBlur(image,5)
        image = cv.GaussianBlur(image,(7,7),0)

    gray = cv.cvtColor(image, cv.COLOR_BGR2GRAY)
    cv.imwrite(fileName + '-gray' + fileExt, gray)
    if(fileName.startswith("Dugong")):
        (thresh, binary) = cv.threshold(gray, 128, 255, cv.THRESH_BINARY + cv.THRESH_OTSU)
    else:
        # Use binary inv because the card's background is white
        (thresh, binary) = cv.threshold(gray, 128, 255, cv.THRESH_BINARY_INV + cv.THRESH_OTSU)
    numLabels, labels, stats, centroids = cv.connectedComponentsWithStats(binary, 8, cv.CV_32S)
    textFile = open(folder  +fileName+"-ccl-stats.txt","w")
    # numLabels - 1 because excluding the background
    textFile.write("Number of components: " + str(numLabels-1) + "\n")
    for i in range(1, numLabels):
        mask =  image.copy()
        mask[labels==i] = 0
        mask[labels!=i] = 255
        x, y, xOffset, yOffset, area= stats[i]
        crop = mask[y:y+yOffset, x:x+xOffset]
        cv.imwrite(folder + fileName + '-ccl-' + str(i) + fileExt, crop)
        textFile.write("Area of component " + str(i) + ": " +  str(area) + "\n")

    cv.imwrite(folder + fileName + '-binary' + fileExt, binary)

def main():
    if(len(sys.argv) == 2):
        path = Path(sys.argv[1])
        folder = "task3/"
        if not os.path.exists(folder):
            os.mkdir(folder)
        file = sys.argv[1]
        image = cv.imread(file)
        connectedComponentLabelling(image.copy(), path.stem, path.suffix, folder)
    else:
        print("Please run the program with 'python task3.py file'")

if(__name__ == '__main__'):
    main()
```

D. Appendix 4

```
import cv2 as cv
import numpy as np
import sys
from pathlib import Path
from matplotlib import pyplot as plt
import os

def method_red_only(image):
    # Good image segmentation result if only using the red channel
    # colours for Dugong image but doesnt work with diamond image
    # Sets the blue colour channel to zero
    image[:,:,0] = 0
    # Sets the green colour channel to zero
    image[:,:,1] = 0
    return image

def method_hsv_no_v(image):
    image = cv.cvtColor(image, cv.COLOR_BGR2HSV)
    # Sets the Value colour channel to zero
    image[:,:,2] = 0
    return image

'''
Contains code copied and adapted from https://docs.opencv.org/3.4.2/d1/d5c/tutorial_py_kmeans_opencv.html
The code I've written below is similar to https://www.pyimagesearch.com/2015/12/07/local-binary-patterns-
    with-python-opencv/
but I didn't copy the code from the website except adapted code on how to disable clusters to display a
    certain cltuer
and i did read the comments in the code in that website for further understanding of how kmeans image
    segmentation work in OpenCV
Last accessed on 30/09/2020
'''
def kMeans(original, image, _type, fileName, fileExt, folder):

    # reshape the image to a 2D array of dimension Mx3 where M is the number of pixels and 3 is number of
    colour channels in BGR
    pixel_values = image.reshape((-1, 3))
    # convert to float
    pixel_values = np.float32(pixel_values)

```

```python
    # define stopping criteria
    criteria = (cv.TERM_CRITERIA_EPS + cv.TERM_CRITERIA_MAX_ITER, 10, 1.0)
    # number of clusters (K)
    k = 2
    # apply kmeans
    _, labels, (centers) = cv.kmeans(pixel_values, k, None, criteria, 10, cv.KMEANS_RANDOM_CENTERS)

    # convert back to 8 bit values, only needed if want to show full segmented image
    centers = np.uint8(centers)

    # flatten the labels array
    labels = labels.flatten()
    textFile = open(folder + fileName+"-labels.txt","w")
    np.set_printoptions(threshold=np.inf)
    textFile.write(np.array_str(labels))

    # loop through all cluster and only output the image of that cluster
    for i in range(0,k):
        segmented_image = np.copy(original)
        # convert to 2d array similiar to what is used for k-means so that we can set clusters of pixels to
         have the pixel value of black colour
        segmented_image = segmented_image.reshape((-1, 3))
        # set every other cluster that isn't i to be black colour
        segmented_image[labels != i] = [0, 0, 0]
        # convert back to original shape
        segmented_image = segmented_image.reshape(image.shape)
        # save the image
        cv.imwrite(folder + fileName + "-" + _type + '-kmeans-' + str(i) + fileExt, segmented_image)

def main():
    if(len(sys.argv) == 2):
        path = Path(sys.argv[1])
        folder = "task4/"
        if not os.path.exists(folder):
            os.mkdir(folder)
        file = sys.argv[1]
        original = cv.imread(file)
        image = original.copy()
        kMeans(original, image, "BGR", path.stem, path.suffix, folder)
        image = method_red_only(original.copy())
        kMeans(original, image, "R", path.stem, path.suffix, folder)
        image = method_hsv_no_v(original.copy())
        kMeans(original, image, "HS", path.stem, path.suffix, folder)
    else:
        print("Please run the program with 'python task4.py file'")

if(__name__ == '__main__'):
    main()
```