Machine Perception Assignment 1

Reece Jones   
Student ID: 19476111

# Task 1

## Harris Corner Detector

The Harris corner detector [1] is variant under scaling and invariant under rotation. Scale invariance is not achieved since the size or standard deviation of the window function is a fixed parameter of the algorithm; with images of different scale, the window will cover a different size image patch. I expect that the higher the ratio of window size to feature size, the lower the corner response, since the window will include a higher proportion of noncorner pixels. The experimental results are shown in Fig. 1. Interestingly, as the image size increases, the number of corners detected decreases, contrary to my hypothesis. I believe that in the process of resampling the images, the increase in the size of the edges has reduced the gradient, reducing the corner response. This effect is likely dependent on the resampling method used (preliminary testing with nearest-neighbour resampling indicates different results).

Rotational invariance is achieved because the corner response is calculated based on the eigenvalues of the autocorrelation matrix at a point [2]. These eigenvalues give the magnitudes of curvature of the image surface in the directions of most and least curvature [1]. That is, any rotation present has no effect on the eigenvalues, and therefore no effect on the corner response. The experimental results are shown in Fig. 2; as expected, the same corners are detected regardless of rotation.

The experimental results for the dugong image are similar and may be reproduced using the code in Appendix 1.

## Intensity Histograms

Image histograms are variant under scaling and may be variant under rotation (depending on the rotation). Scaling an image changes its resolution, and since histograms are based on pixel counts, the histogram will be affected. However, for small or moderate changes in scale, the relative ratios of intensities will be retained, i.e. the histogram will have approximately the same “shape” (if normalised histograms are used, then scale invariance is achieved). This is demonstrated in Fig. 3. Notice that the histogram peaks are vastly different, but the overall shape is almost identical.

Histograms may be invariant under rotation, depending on whether the rotation affects the image bounds. Typically, histograms are calculated across an image or bounding box, the content of which will change under most rotations (consider the 45-degree rotation of an entire image – some regions are cut off, and empty regions are introduced). If we apply a rotation but adjust the bounds within which we calculate the histogram accordingly, then the histogram will be approximately unchanged. The experimental results are shown in Fig. 4. A fixed bounding box was selected, with the image rotates with respect to it. The image content inside the box did change, but since the background (ocean) is mostly homogeneous, the histograms are similar.

|  |  |  |
| --- | --- | --- |
| **Scale** | **Keypoints** | **Keypoints common with 1x** |
| 1x | 16 | 16 |
| 1.25x | 25 | 13 |
| 1.5x | 32 | 16 |
| 2x | 45 | 16 |

|  |  |  |
| --- | --- | --- |
| **Rotation** | **Keypoints** | **Keypoints common with 0°** |
| 0° | 16 | 16 |
| 15° | 16 | 12 |
| 45° | 16 | 11 |
| 75° | 15 | 10 |

The experimental results for the playing card image are similar and may be reproduced using the code in Appendix 1.

## Scale-invariant Feature Transform (SIFT) Keypoints

SIFT [3, 4] keypoints are moderately invariant under small to moderate scaling and rotation. Scale invariance is achieved by examining the image at many scales (via Gaussian filtering and image downsampling), transforming the image into “scale space”. Keypoints are detected within this scale space, yielding not only the keypoint’s location in the image, but also the scale at which the keypoint exists. Further processing of the keypoint is adjusted for this scale, producing keypoints which are largely invariant to scale. The experimental results are summarised in Fig. 5. As the image size increases, the number of keypoints detected increases (presumably since there are simply more pixels in the image), but a significant proportion of them match the keypoints in the original image region. The match proportions upwards of 80% for small scaling factors coincides with the results presented in [3].

Rotational invariance is (theoretically) achieved since keypoint detection only considers the magnitude of the difference of Gaussians (which is isotropic) [3, 4]. The experimental results are summarised in Fig. 6. Surprisingly, only about 65-75% of keypoints from the original image region are present in the rotated versions. My only explanation as to why such a large proportion of keypoints did not match is that it is as a result of the image resampling during rotation, especially given that the image region is small, however, I am unsure. It seems plausible that a more complex explanation exists, due to the many steps in the SIFT algorithm.

The keypoint-annotated images are omitted for brevity but may be reproduced using the code in Appendix 1. The experimental results for the playing card image are similar and may be reproduced using the same code.

# Task 2

## Local Binary Patterns (LBP) Features

LBP [5] generates an 8-bit integer feature for every pixel in a greyscale image (or image region). The feature is determined from the 8 pixels neighbouring the target location. These neighbouring pixels are traversed in a fixed order and compared to the target pixel. If the neighbouring pixel is less than the target pixel, a 1 is recorded; if the neighbouring pixel is greater than the target pixel, a 0 is recorded. The 1s and 0s are concatenated to form an 8-bit value, which is the feature descriptor for the target location. Further processing may involve dividing the image into cells, within which histograms of the feature values are calculated.

The main advantage of LBP is that it is quite effective compared to other texture descriptors, as found by [5] and [6]. Contributing to its effectiveness is its invariance under global changes in illumination. The algorithm may be extended to consider neighbouring pixels at any radius, allowing adjustment of the locality of detail that is extracted [7]. Finally, the algorithm is simple and computationally efficient.

The disadvantages of LBP are that it is not (without extensions) scale nor rotationally invariant and does not consider colour information.

## Scale-invariant Feature Transform (SIFT) Features

SIFT [3, 4] generates feature descriptors from the local neighbourhoods around the detected keypoints. First, the 16x16 region of pixels centred at the keypoint is considered and divided into 16 4x4 pixel blocks. For each block, an 8-bin histogram of gradient orientations is created. The orientation of each pixel is calculated at the scale of the detected keypoint, achieving scale invariance, and is given relative to the orientation of the keypoint, achieving rotational invariance. Each pixel’s histogram vote is weighted by the magnitude of its gradient, and by a Gaussian window centred at the keypoint. The 16 histograms are concatenated into a single 128-element vector to form the feature descriptor. Finally, this vector is thresholded and normalised to achieve some invariance to illumination changes.

The main advantages of SIFT features is, of course, that they are invariant under scaling, rotation, and illumination changes. Another advantage, demonstrated in [4], is that the high distinctiveness of the descriptors enables reliable feature matching even within very large databases.

The disadvantages of SIFT features are that the algorithm is quite complex, and colour information is not considered. [4] also notes that the high dimensionality of the feature descriptors inhibited feature matching efficiency when used with nearest-neighbour classification.

## Histogram of Oriented Gradients (HOG) Features

HOG [8] generates a feature descriptor for an image (or image region), derived from 8x8 pixel cells. A gradient orientation histogram is created for each cell. The histogram uses 9 bins for 0° to 180°, with orientations taken modulo 180° (this was found to be most effective for human detection). Each pixel’s vote is linearly interpolated between the 2 nearest histogram bins and is additionally weighted by the magnitude of the gradient. The cells are then grouped into overlapping 2x2 cell (16x16 pixel) blocks and normalised, to account for local illumination changes. Finally, the blocks are flattened and concatenated into a single vector to form the feature descriptor.

The primary advantage of HOG features is their excellent performance in the detection of humans (significantly better than other techniques), in part due to their illumination invariance [8].

The disadvantages of HOG features are that they are not scale nor rotationally invariant. Another possible disadvantage is the high dimensionality of the descriptor, which may be a challenge for classification.

## Comparison of SIFT and HOG Descriptors

Fig. 7 and 8 compare the similarity of SIFT and HOG feature descriptors taken from a region of the dugong image, across different scales and rotations. The descriptors are taken at SIFT keypoints that match in location and orientation across all versions of the image. For each transformed version of the image, the Euclidean distance is calculated between its matrix of descriptors and the matrix of descriptors from the original, untransformed image.

As expected, the HOG descriptors exhibit less similarity as the degree of transformation increases, since HOG is variant under scaling and rotation. The SIFT descriptors also exhibit some dissimilarity; however, to a lesser amount. Critically, while the HOG descriptor similarity is strictly decreasing, the SIFT descriptor similarity is not particularly correlated to the amount of transformation. This result highlights the scale and rotational invariance of SIFT.

Interestingly, both SIFT and HOG descriptors exhibit a large jump in dissimilarity between the untransformed image versions and the first transformed version. From a theoretical standpoint, I would expect the smallest amount of transformation to produce the smallest descriptor difference. Additionally, the SIFT descriptor difference decreases from scales of 1.6x to 2x – I would expect that it stays the same or increases slightly. I believe these observations are effects of the image resampling during scaling/rotation. Preliminary testing with the application of a transformation followed by the inverse transformation – theoretically producing no change – shows similar patterns. Further investigation may be required to determine the effects of resampling on the results presented here.

The experimental results for the playing card image are similar and may be reproduced using the code in Appendix 2.

# Task 3

## Binarisation

Despite consisting of only 3 colours and simple geometry, it is not entirely trivial to binarise the playing card image. I assumed that both the black and white portions of the image are to be considered as background, meaning that both low and high intensity pixels are background. Therefore, neither a simple threshold nor Otsu’s binarisation of the greyscale image are applicable. However, between the white and black on the edge of the card, there is also a small region of medium-intensity grey pixels (presumably from interpolation or antialiasing), meaning that foreground pixels are not uniquely identified by medium intensity either. Conveniently, in HSV colour space, black, white, and grey all have low saturation values, while the red foreground does not. Therefore, the image is converted to HSV and a threshold applied on the saturation value. This scheme produces a very good binarisation, which may be seen in Fig. 9. The only imperfections present are aliasing artifacts on the edges of the diamonds, which I do not think are possible to avoid in a binary image.

As a real-world scene, the dugong image was somewhat more difficult to binarise. Simple thresholding or Otsu’s binarisation of the greyscale image were not applicable, as the ocean has a deceptively large intensity gradient between the outer edges and centre (this is not easily perceivable to the human eye). Such methods are likely to fail to classify the entire ocean as background. However, in HSV colour space, the ocean differs significantly in saturation from the foreground objects. A simple threshold on the saturation value was sufficient. There are some small speckles of lighter colour, present in the ocean that were similar enough to the dugongs’ colour to be frequently classified as foreground. To remove these outliers, the image is preprocessed with a small median filter. The median filter was chosen as it preserves edges better than a Gaussian or box filter. Its small size avoids the loss of fine detail such as the smaller dugong’s tail. The resulting binarisation is very good and is shown in Fig. 11.

## Extracted Objects

Since clean binarisations were achieved, the extraction of objects was straightforward using a connected components algorithm. In the playing card image, 6 objects were extracted. In the dugong image, 2 objects were extracted. The 2 dugongs could not be separated as distinct objects; their overlap and similar colour makes such a task significantly more difficult. The objects and their areas are shown in Fig. 10 and 12. Note that the areas include only the foreground pixels, not the entire bounding box.

|  |  |  |
| --- | --- | --- |
| **Feature Space** | **SSDC****[[1]](#footnote-1)** | **Rand Index[[2]](#footnote-2)** |
| Saturation | 11 | 0.9561 |
| Hue | 20 | 0.8177 |
| Red | 39 | 0.9521 |
| Green | 53 | 0.9501 |
| Blue | 54 | 0.9811 |
| Intensity | 61 | 0.9882 |
| L\*a\*b\* | 86 | 0.9893 |
| L\*u\*v\* | 89 | 0.9894 |
| YUV | 102 | 0.9894 |
| XYZ | 215 | 0.9885 |
| RGB | 290 | 0.9895 |
| HSV | 541 | 0.9886 |
| HLS | 786 | 0.9404 |

# Task 4

## Playing Card Image

|  |  |  |
| --- | --- | --- |
| **Feature Space** | **SSDC1** | **Rand Index2** |
| Hue | 20 | 0.5203 |
| Red | 114 | 0.9990 |
| Saturation | 190 | 0.9993 |
| Intensity | 190 | 0.4252 |
| Blue | 199 | 0.3486 |
| Green | 241 | 0.3692 |
| HLS | 319 | 0.5259 |
| YUV | 339 | 0.5040 |
| L\*u\*v\* | 360 | 0.5006 |
| HSV | 366 | 0.5054 |
| L\*a\*b\* | 368 | 0.5013 |
| XYZ | 432 | 0.4105 |
| RGB | 1534 | 0.9996 |

The playing card image was segmented with k-means into 3 clusters. The desired result was for the black, white, and red regions of the image to be classified into separate clusters. Since the segmentation is based on colour information, it is not possible to separate the individual red objects as done in section III-B. The performance of the feature spaces is summarised in Fig. 13. We see that the feature spaces that produce the most optimal segmentations as far as k-means is concerned (i.e. intracluster sum of squared distances (SSDC)) do not necessarily produce the best segmentations as evaluated by a human. The only feature spaces to produce a visibly completely acceptable result are L\*u\*v\*, YUV, RGB (Fig. 15a), and HSV, with the 6th, 5th, 3rd, and 2nd highest SSDCs, respectively. Interestingly, these are all 3D feature spaces. Red, intensity (greyscale) (Fig. 15b), and XYZ produce an almost correct result, mislabelling the grey edge around the playing card. Hue (Fig. 15c), which produces the 2nd lowest SSDC, discards most of the image information (black, white, and red all have hues close to 0) and thus achieves very low SSDC via the placement of most of the pixels into a single cluster. Saturation (Fig. 15d), which produces the lowest SSDC, yields a very clean segmentation similar to the binarisation achieved in section III-A, alas not the segmentation that is desired.

## Dugong Image

The dugong image was segmented with k-means into 3 clusters. The desired result was the for the dugongs (both as one region), seaweed, and ocean to be classified into separate clusters. The performance of the feature spaces is summarised in Fig. 14. A common result among the feature spaces was for the ocean to be classified into multiple regions rather than 1. Hue (Fig. 16a), L\*a\*b\*, L\*u\*v\*, YUV, HLS, and HSV split the ocean into 2 regions, while intensity (Fig. 16b), blue, green, and XYZ split the ocean into 3 regions. As mentioned in section III-A, there is a significant gradient in the ocean between the centre and edges of the image. I believe this gradient is large enough, and across a large enough number of pixels, that it is often optimal for k-means to divide the image on it and classify the dugong and seaweed as one region. The dugong and seaweed are simply too small and of too similar colour to pose a significant impact on the SSDC. I do not think this behaviour is an issue with convergence to a local minimum; I believe it is globally optimal for most colour spaces, as demonstrated by the hue segmentation (Fig. 16a), which has the lowest SSDC by far. However, evidently some of the feature spaces provided sufficient homogeneity in the ocean and difference between the ocean and dugong/seaweed that the ocean occupied only 1 cluster; these were RGB (Fig. 16c), saturation (Fig. 16d), and red. Unfortunately, none of these feature spaces also separated the dugongs and seaweed into entirely separate clusters – in hindsight, expecting such a segmentation was probably too ambitious considering the similarity in colour between them. I expect using 2 clusters instead of 3 may be more fruitful.

## Conclusions

It is evident from segmentation of the playing card image that selection of a feature space in which the desired clusters are well-separated is important for an acceptable result. Reducing the dimensionality of the data, for example remapping to intensity or saturation (1D), is likely to produce undesired results without careful consideration of the arrangement of the clusters in the new space. The segmentation of the dugong image highlights the significance of cluster sizes; data with clusters of highly variant sizes is prone to yielding poor results. Finally, both images demonstrate the importance of understanding the difference between k-means’ optimality and optimality as perceived by a human. The results consistently show little correlation between low SSDC and high Rand index. K-means simply minimises the SSDC, this may or may not produce a desirable clustering. An understanding of K-means and of one’s data is vital.

##### References

1. C. Harris and M. Stephens, “A combined corner and edge detector,” *Alvey Vision Conference*, vol. 15, no. 50, pp. 147-151, 1988.
2. R. Szeliski, *Computer Vision: Algorithms and Applications*, 2nd ed. Accessed Sep. 14, 2020. [Online]. Available: http://szeliski.org/Book.
3. D. G. Lowe, “Object recognition from local scale-invariant features,” in *Proceedings of the Seventh IEEE International Conference on Computer Vision*, 1999, pp. 1150-1157 vol.2, doi: 10.1109/ICCV.1999.790410.
4. D. G. Lowe, “Distinctive image features from scale-invariant keypoints,” *International Journal of Computer Vision*, vol. 60, no. 2, pp. 91-110, Nov. 2004, doi: 10.1023/B:VISI.0000029664.99615.94.
5. T. Ojala, M. Pietikäinen and D. Harwood, “A comparative study of texture measures with classification based on feature distributions,” *Pattern Recognition*, vol. 29, no. 1, pp 51-59, Jan. 1996, doi: 10.1016/0031-3203(95)00067-4.
6. T. Ahonen, A. Hadid and M. Pietikäinen, “Face description with local binary patterns: Application to face recognition,” in *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 28, no. 12, pp. 2037-2041, Dec. 2006, doi: 10.1109/TPAMI.2006.244.
7. T. Ojala, M. Pietikäinen and T. Mäenpää, “Multiresolution gray-scale and rotation invariant texture classification with local binary patterns,” in *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 24, no. 7, pp. 971-987, July 2002, doi: 10.1109/TPAMI.2002.1017623.
8. N. Dalal and B. Triggs, "Histograms of oriented gradients for human detection," 2005 IEEE Computer Society Conference on Computer Vision and Pattern Recognition, San Diego, CA, USA, 2005, pp. 886-893 vol. 1, doi: 10.1109/CVPR.2005.177.

##### Appendix 1

## Scaled Image Versions Generation

import cv2 as cv  
import os.path  
from pathlib import Path  
  
IMAGES = [  
 (**'data/card.png'**, ((0, 0), (-1, -1))),  
 (**'data/dugong.jpg'**, ((360, 195), (460, 295)))  
]  
SCALES = [1, 1.2, 1.25, 1.4, 1.5, 1.6, 1.75, 1.8, 2]  
OUTPUT\_DIR = **'generated/{}/scaled'**OUTPUT\_FILE = **'{}x.png'**for file, roi in IMAGES:  
 image = cv.imread(file, cv.IMREAD\_COLOR)  
 filename = os.path.splitext(os.path.basename(file))[0]  
 output\_dir = OUTPUT\_DIR.format(filename)  
 Path(output\_dir).mkdir(parents=True, exist\_ok=True)  
 x1, y1 = roi[0]  
 x2, y2 = roi[1]  
 image = image[y1:y2, x1:x2]  
 for scale in SCALES:  
 result = cv.resize(image, (0, 0), fx=scale, fy=scale, interpolation=cv.INTER\_CUBIC)  
 output\_file = os.path.join(output\_dir, OUTPUT\_FILE.format(scale))  
 cv.imwrite(output\_file, result)

## Rotated Image Versions Generation

import cv2 as cv  
import os.path  
from pathlib import Path  
  
INPUT = [  
 (**'data/card.png'**, ((51, 20), (131, 100))),  
 (**'data/dugong.jpg'**, ((360, 195), (460, 295)))  
]  
ROTATIONS = [0, 15, 30, 45, 60, 75, 90]  
OUTPUT\_DIR = **'generated/{}/rotated'**OUTPUT\_FILE = **'{}deg.png'**for file, roi in INPUT:  
 image = cv.imread(file, cv.IMREAD\_COLOR)  
 filename = os.path.splitext(os.path.basename(file))[0]  
 output\_dir = OUTPUT\_DIR.format(filename)  
 Path(output\_dir).mkdir(parents=True, exist\_ok=True)  
 x1, y1 = roi[0]  
 x2, y2 = roi[1]  
 centre = ((x1 + x2) / 2, (y1 + y2) / 2)  
 for angle in ROTATIONS:  
 height, width, \_ = image.shape  
 rotation = cv.getRotationMatrix2D(centre, angle, 1)  
 result = cv.warpAffine(image, rotation, (width, height), flags=cv.INTER\_CUBIC)  
 result = result[y1:y2, x1:x2]  
 output\_file = os.path.join(output\_dir, OUTPUT\_FILE.format(angle))  
 cv.imwrite(output\_file, result)

## Harris Corner Detection

The code in Appendices 1-A and 1-B should be run before running this script.

import cv2 as cv  
from glob import glob  
import numpy  
import os.path  
from pathlib import Path  
  
BASE\_IMAGES = [(**'card'**, 0.005), (**'dugong'**, 0.0003)]  
TRANSFORMS = [**'scaled'**, **'rotated'**]  
INPUT\_FILES = **'generated/{}/{}/\*.png'**OUTPUT\_DIR = **'results/{}/{}'**OUTPUT\_FILE = **'{}-harris.png'**HARRIS\_BLOCK\_SIZE = 3  
HARRIS\_SOBEL\_SIZE = 3  
HARRIS\_K = 0.05  
  
for base\_image, harris\_threshold in BASE\_IMAGES:  
 for transform in TRANSFORMS:  
 file\_pattern = INPUT\_FILES.format(base\_image, transform)  
 output\_dir = OUTPUT\_DIR.format(base\_image, transform)  
 Path(output\_dir).mkdir(parents=True, exist\_ok=True)  
 for file in glob(file\_pattern):  
 image = cv.imread(file, cv.IMREAD\_COLOR)  
 image\_grey = cv.cvtColor(image, cv.COLOR\_BGR2GRAY)  
 filename = os.path.splitext(os.path.basename(file))[0]  
  
 harris\_response = cv.cornerHarris(image\_grey, HARRIS\_BLOCK\_SIZE, HARRIS\_SOBEL\_SIZE, HARRIS\_K)  
 corners = harris\_response > harris\_threshold  
 result = image.copy()  
 for (i, j) in zip(\*numpy.where(corners)):  
 marker\_size = min(image\_grey.shape[0], image\_grey.shape[1]) // 10  
 result = cv.drawMarker(result, (j, i), (0, 0, 0), markerSize=marker\_size, thickness=1)  
 output\_file = os.path.join(output\_dir, OUTPUT\_FILE.format(filename))  
 cv.imwrite(output\_file, result)

## Histogram Calculation

The code in Appendices 1-A and 1-B should be run before running this script.

import cv2 as cv  
from glob import glob  
from matplotlib import pyplot  
import os.path  
from pathlib import Path  
  
BASE\_IMAGES = [**'card'**, **'dugong'**]  
TRANSFORMS = [**'scaled'**, **'rotated'**]  
INPUT\_FILES = **'generated/{}/{}/\*.png'**OUTPUT\_DIR = **'results/{}/{}'**OUTPUT\_FILE = **'{}-hist.png'**HISTOGRAM\_BINS = 16  
  
for base\_image in BASE\_IMAGES:  
 for transform in TRANSFORMS:  
 file\_pattern = INPUT\_FILES.format(base\_image, transform)  
 output\_dir = OUTPUT\_DIR.format(base\_image, transform)  
 Path(output\_dir).mkdir(parents=True, exist\_ok=True)  
 for file in glob(file\_pattern):  
 image = cv.imread(file, cv.IMREAD\_GRAYSCALE)  
 filename = os.path.splitext(os.path.basename(file))[0]  
  
 pyplot.clf()  
 pyplot.hist(image.flatten(), HISTOGRAM\_BINS, (0, 255))  
 pyplot.xlabel(**'Intensity'**)  
 pyplot.ylabel(**'Frequency'**)  
 pyplot.ylim((0, image.shape[0] \* image.shape[1]))  
 pyplot.tight\_layout()  
 output\_file = os.path.join(output\_dir, OUTPUT\_FILE.format(filename))  
 pyplot.savefig(output\_file)

## SIFT Keypoint Generation

The code in appendices 1-A and 1-B should be run before running this script.

import cv2 as cv  
from glob import glob  
import os.path  
from pathlib import Path  
  
BASE\_IMAGES = [**'card'**, **'dugong'**]  
TRANSFORMS = [**'scaled'**, **'rotated'**]  
INPUT\_FILES = **'generated/{}/{}/\*.png'**OUTPUT\_DIR = **'results/{}/{}'**OUTPUT\_FILE = **'{}-keypoints.png'**SCALE = 2  
  
for base\_image in BASE\_IMAGES:  
 for transform in TRANSFORMS:  
 file\_pattern = INPUT\_FILES.format(base\_image, transform)  
 output\_dir = OUTPUT\_DIR.format(base\_image, transform)  
 Path(output\_dir).mkdir(parents=True, exist\_ok=True)  
 for file in glob(file\_pattern):  
 image = cv.imread(file, cv.IMREAD\_COLOR)  
 filename = os.path.splitext(os.path.basename(file))[0]  
  
 sift = cv.xfeatures2d.SIFT\_create()  
 keypoints = sift.detect(image, None)  
 *# Scale everything up so we can see it better (doesn't affect keypoint location)* for kp in keypoints:  
 kp.pt = (kp.pt[0] \* SCALE, kp.pt[1] \* SCALE)  
 kp.size \*= SCALE  
 result = cv.resize(image, (0, 0), fx=SCALE, fy=SCALE)  
 result = cv.drawKeypoints(result, keypoints, None)  
 output\_file = os.path.join(output\_dir, OUTPUT\_FILE.format(filename))  
 cv.imwrite(output\_file, result)

##### Appendix 2

## Scaled Image Versions Generation

The same code as in Appendix 1-A was used.

## Rotated Image Versions Generation

The same code as in Appendix 1-B was used.

## SIFT and HOG Descriptor Comparison

The code in appendices 1-A and 1-B should be run before running this script.

import cv2 as cv  
from matplotlib import pyplot  
import numpy  
import os.path  
from pathlib import Path  
  
def inverse\_scale(image\_size, scale, keypoint):  
 T = [[1 / scale, 0],  
 [0, 1 / scale]]  
 return numpy.dot(keypoint.pt, T), keypoint.angle  
  
def inverse\_rotation(image\_size, angle, keypoint):  
 centre = (image\_size[0] / 2, image\_size[1] / 2)  
 T = cv.getRotationMatrix2D(centre, -angle, 1)  
 return numpy.dot(T, [keypoint.pt[0], keypoint.pt[1], 1]), keypoint.angle + angle  
  
BASE\_IMAGES = [**'card'**, **'dugong'**]  
TRANSFORMS = [  
 (**'scaled'**, **'{}x.png'**, [1, 1.2, 1.4, 1.6, 1.8, 2], inverse\_scale, **'Scale factor'**),  
 (**'rotated'**, **'{}deg.png'**, [0, 15, 30, 45, 60, 75], inverse\_rotation, **'Rotation angle (degrees)'**)  
]  
INPUT\_DIR = **'generated/{}/{}'**OUTPUT\_DIR = **'results/{}/{}'**KEYPOINT\_MATCH\_OUTPUT = **'keypoint\_matches.png'**DESCRIPTOR\_COMPARE\_OUTPUT = **'descriptor\_compare.png'**KEYPOINT\_MATCH\_DIST\_THRESHOLD = 1  
KEYPOINT\_MATCH\_ANGLE\_THRESHOLD = 15  
  
sift = cv.xfeatures2d.SIFT\_create()  
hog = cv.HOGDescriptor((16, 16), (16, 16), (8, 8), (8, 8), 9)  
for base\_image in BASE\_IMAGES:  
 for transform\_name, file\_template, transform\_vals, inverse\_transform, transform\_axis\_label in TRANSFORMS:  
 *# METHODOLOGY:  
 # To choose keypoints to examine, we first get all keypoints from all versions of the image.  
 # We then apply to the keypoints the inverse of the transformation that was applied to that  
 # image, to get the locations and orientations the keypoints "should have" if SIFT was  
 # perfectly scale/rotation invariant. Finally, we select keypoints that match in location  
 # and orientation across all versions of the image.* input\_dir = INPUT\_DIR.format(base\_image, transform\_name)  
 image\_versions = len(transform\_vals)  
 images = []  
 keypoints = []  
 for transform\_val in transform\_vals:  
 file = os.path.join(input\_dir, file\_template.format(transform\_val))  
 image = cv.imread(file, cv.IMREAD\_COLOR)  
 images.append(image)  
 actual\_keypoints = sift.detect(image, None)  
 adjusted\_keypoints = [inverse\_transform((image.shape[1], image.shape[0]), transform\_val, kp)  
 for kp in actual\_keypoints]  
 keypoints.append(list(zip(actual\_keypoints, adjusted\_keypoints)))  
  
 *# Find keypoints in the transformed images that match in location and orientation with the  
 # original image.* keypoint\_matches = [[kp] for kp, \_ in keypoints[0]]  
 for i in range(1, image\_versions):  
 for j, (orig\_keypoint, \_) in enumerate(keypoints[0]):  
 matches = [(kp, numpy.linalg.norm(orig\_keypoint.pt - adjusted\_kp[0]), abs(orig\_keypoint.angle - adjusted\_kp[1]))  
 for kp, adjusted\_kp in keypoints[i]]  
 matches = [(kp, dist, angle\_diff) for kp, dist, angle\_diff in matches  
 if dist < KEYPOINT\_MATCH\_DIST\_THRESHOLD and angle\_diff < KEYPOINT\_MATCH\_ANGLE\_THRESHOLD]  
 if matches:  
 *# Only need one match, so consider the distance, angle difference, and keypoint  
 # response to determine which is the "best" match.* def match\_cost(m):  
 return m[1] / KEYPOINT\_MATCH\_DIST\_THRESHOLD + m[2] / KEYPOINT\_MATCH\_ANGLE\_THRESHOLD - 10 \* m[0].response  
 match, \_, \_ = min(matches, key=match\_cost)  
 keypoint\_matches[j].append(match)  
  
 *# Select only keypoints that match in all the image versions.* keypoint\_matches = [matches for matches in keypoint\_matches if len(matches) == image\_versions]  
 keypoint\_matches = numpy.transpose(keypoint\_matches)  
  
 output\_dir = OUTPUT\_DIR.format(base\_image, transform\_name)  
 Path(output\_dir).mkdir(parents=True, exist\_ok=True)  
  
 result = cv.drawKeypoints(images[0].copy(), keypoint\_matches[0], None)  
 output\_file = os.path.join(output\_dir, KEYPOINT\_MATCH\_OUTPUT)  
 cv.imwrite(output\_file, result)  
  
 sift\_descriptors = numpy.array([sift.compute(image, keypoint\_matches[i])[1]  
 for i, image in enumerate(images)])  
 *# For some reason OpenCV makes the norm 512 instead of 1, so fix that so we can compare with  
 # HOG more easily.* sift\_descriptors /= 512  
  
 hog\_descriptors = numpy.empty((image\_versions, keypoint\_matches.shape[1], 36), dtype=numpy.float)  
 for i, image in enumerate(images):  
 for j, keypoint in enumerate(keypoint\_matches[i]):  
 *# To get the HOG descriptor around the keypoint, we extract the 16x16 region around  
 # the keypoint, and calculate HOG on that (easier than calculating HOG on the whole  
 # image then extracting the region). Need to clamp region to bounds of image  
 # (keypoint might be near edge of image).* region\_cx = numpy.clip(round(keypoint.pt[0]), 8, image.shape[1] - 8)  
 region\_cy = numpy.clip(round(keypoint.pt[1]), 8, image.shape[0] - 8)  
 region = image[region\_cy - 8:region\_cy + 8, region\_cx - 8:region\_cx + 8]  
 descriptor = hog.compute(region).ravel()  
 hog\_descriptors[i][j] = descriptor  
  
 pyplot.clf()  
 pyplot.xlabel(transform\_axis\_label)  
 pyplot.ylabel(**'Distance from original'**)  
 for descriptors, label in [(sift\_descriptors, **'SIFT'**), (hog\_descriptors, **'HOG'**)]:  
 distances = [numpy.linalg.norm(desc - descriptors[0]) for desc in descriptors]  
 *# Use plot() because of autoscale bug with scatter()* pyplot.plot(transform\_vals, distances, marker=**'o'**, ls=**''**, label=label)  
 pyplot.legend()  
 pyplot.tight\_layout()  
 output\_file = os.path.join(output\_dir, DESCRIPTOR\_COMPARE\_OUTPUT)  
 pyplot.savefig(output\_file)

##### Appendix 3

## Object Extraction

import cv2 as cv  
import numpy  
import os.path  
from pathlib import Path  
  
def threshold\_card(image):  
 *# Assume the background includes the black behind/around the card and the card's white bit.* image = cv.cvtColor(image, cv.COLOR\_BGR2HSV)  
 binary = image[:, :, 1] > 100  
 return binary.astype(numpy.uint8)  
  
def threshold\_dugong(image):  
 *# Median blur to get rid of small speckled bits in ocean.* image = cv.medianBlur(image, 3)  
 image = cv.cvtColor(image, cv.COLOR\_BGR2HSV)  
 binary = image[:, :, 1] < 150  
 return binary.astype(numpy.uint8)  
  
INPUTS = [  
 (**'data/card.png'**, threshold\_card),  
 (**'data/dugong.jpg'**, threshold\_dugong)  
]  
OUTPUT\_DIR = **'results/{}'**BINARY\_OUTPUT\_FILE = **'binary.png'**LABELS\_OUTPUT\_FILE = **'object\_labels.png'**OBJECTS\_OUTPUT\_FILE = **'objects.png'**AREAS\_OUTPUT\_FILE = **'object\_areas.txt'**LABEL\_COLOURS = numpy.array([  
 (255, 255, 255),  
 (255, 0, 0),  
 (0, 255, 0),  
 (0, 0, 255),  
 (255, 255, 0),  
 (0, 150, 255),  
 (255, 0, 255)  
])  
PADDING = 2  
  
for file, threshold\_func in INPUTS:  
 image = cv.imread(file, cv.IMREAD\_COLOR)  
 filename = os.path.splitext(os.path.basename(file))[0]  
 output\_dir = OUTPUT\_DIR.format(filename)  
 Path(output\_dir).mkdir(parents=True, exist\_ok=True)  
  
 binary = threshold\_func(image)  
  
 output\_file = os.path.join(output\_dir, BINARY\_OUTPUT\_FILE)  
 cv.imwrite(output\_file, binary \* 255)  
  
 label\_count, labels = cv.connectedComponents(binary, connectivity=8)  
  
 result = LABEL\_COLOURS[labels]  
 output\_file = os.path.join(output\_dir, LABELS\_OUTPUT\_FILE)  
 cv.imwrite(output\_file, result)  
  
 *# Group pixels by label.* points\_by\_label = [numpy.array(list(zip(\*numpy.where(labels == l)))) for l in range(1, label\_count)]  
 *# Sort objects by area.* points\_by\_label = sorted(points\_by\_label, key=lambda ps: ps.shape[0], reverse=True)  
  
 *# Extract the objects and collate them.* bounding\_boxes = [cv.boundingRect(ps[:, [1, 0]]) for ps in points\_by\_label]  
 largest\_width = max(width for x, y, width, height in bounding\_boxes)  
 total\_height = sum(height for x, y, width, height in bounding\_boxes)  
 result = numpy.full((total\_height + PADDING \* label\_count - 1, largest\_width, 3), 255, numpy.uint8)  
 y = 0  
 for (box\_x, box\_y, box\_width, box\_height) in bounding\_boxes:  
 src\_index = (slice(box\_y, box\_y + box\_height), slice(box\_x, box\_x + box\_width))  
 obj = image[src\_index] \* numpy.repeat(binary[src\_index][:, :, numpy.newaxis], 3, axis=2)  
 margin = (largest\_width - box\_width) // 2  
 result[y:y + box\_height, margin:box\_width + margin] = obj  
 y += box\_height + PADDING  
 output\_file = os.path.join(output\_dir, OBJECTS\_OUTPUT\_FILE)  
 cv.imwrite(output\_file, result)  
  
 with open(os.path.join(output\_dir, AREAS\_OUTPUT\_FILE), **'w'**) as areas\_file:  
 areas\_file.writelines(**f'**{ps.shape[0]}\n**'** for ps in points\_by\_label)

##### Appendix 4

## Image Segmentation

import cv2 as cv  
import numpy  
import os.path  
from pathlib import Path  
  
INPUTS = [  
 (**'data/card.png'**, **'data/card-region\_labels.png'**),  
 (**'data/dugong.jpg'**, **'data/dugong-region\_labels.png'**)  
]  
OUTPUT\_DIR = **'results/{}/segmentation'**OUTPUT\_FILE = **'{}.png'**PERFORMANCE\_FILE = **'performance.txt'**CLUSTERS = 3  
ITERATIONS = 200  
EPSILON = 0.01  
ATTEMPTS = 5  
CLUSTER\_COLOURS = numpy.array([  
 (255, 0, 0),  
 (0, 255, 0),  
 (0, 0, 255)  
])  
  
for image\_file, labels\_file in INPUTS:  
 image = cv.imread(image\_file, cv.IMREAD\_COLOR)  
 filename = os.path.splitext(os.path.basename(image\_file))[0]  
 output\_dir = OUTPUT\_DIR.format(filename)  
 Path(output\_dir).mkdir(parents=True, exist\_ok=True)  
  
 correct\_labels = cv.imread(labels\_file, cv.IMREAD\_GRAYSCALE).ravel()  
  
 features = [  
 (**'intensity'**, cv.cvtColor(image, cv.COLOR\_BGR2GRAY)[:, :, numpy.newaxis]),  
 (**'rgb'**, image),  
 (**'hsv'**, cv.cvtColor(image, cv.COLOR\_BGR2HSV)),  
 (**'hls'**, cv.cvtColor(image, cv.COLOR\_BGR2HLS)),  
 (**'xyz'**, cv.cvtColor(image, cv.COLOR\_BGR2XYZ)),  
 (**'lab'**, cv.cvtColor(image, cv.COLOR\_BGR2LAB)),  
 (**'luv'**, cv.cvtColor(image, cv.COLOR\_BGR2LUV)),  
 (**'yuv'**, cv.cvtColor(image, cv.COLOR\_BGR2YUV)),  
 (**'red'**, image[:, :, 2][:, :, numpy.newaxis]),  
 (**'green'**, image[:, :, 1][:, :, numpy.newaxis]),  
 (**'blue'**, image[:, :, 0][:, :, numpy.newaxis]),  
 (**'hue'**, cv.cvtColor(image, cv.COLOR\_BGR2HSV)[:, :, 0][:, :, numpy.newaxis]),  
 (**'saturation'**, cv.cvtColor(image, cv.COLOR\_BGR2HSV)[:, :, 1][:, :, numpy.newaxis])  
 ]  
  
 feature\_performances = []  
 for feature\_name, data in features:  
 height, width, feature\_size = data.shape  
 data = data.reshape((width \* height, feature\_size)).astype(numpy.float32)  
 data = (data - numpy.min(data)) / (numpy.max(data) - numpy.min(data))  
 kmeans\_criteria = (cv.TERM\_CRITERIA\_MAX\_ITER + cv.TERM\_CRITERIA\_EPS, ITERATIONS, EPSILON)  
 kmeans\_flags = cv.KMEANS\_RANDOM\_CENTERS  
 ssdc, labels, centres = cv.kmeans(data, CLUSTERS, None, kmeans\_criteria, ATTEMPTS, kmeans\_flags)  
 labels = labels.reshape((height, width))  
 result = CLUSTER\_COLOURS[labels]  
 output\_file = os.path.join(output\_dir, OUTPUT\_FILE.format(feature\_name))  
 cv.imwrite(output\_file, result)  
  
 labels = labels.ravel()  
 confusion\_matrix = numpy.array([numpy.bincount(labels[correct\_labels == l], minlength=CLUSTERS)  
 for l in range(CLUSTERS)])  
 total = labels.shape[0] / 2 \* (labels.shape[0] - 1)  
 points\_per\_class = numpy.sum(confusion\_matrix, axis=1)  
 points\_per\_cluster = numpy.sum(confusion\_matrix, axis=0)  
 tp\_fp = numpy.sum(points\_per\_cluster / 2 \* (points\_per\_cluster - 1))  
 tp\_fn = numpy.sum(points\_per\_class / 2 \* (points\_per\_class - 1))  
 tp = numpy.sum(confusion\_matrix / 2 \* (confusion\_matrix - 1))  
 tn = total - tp\_fp - tp\_fn + tp  
 rand\_index = (tp + tn) / total  
 feature\_performances.append((feature\_name, ssdc, rand\_index))  
  
 feature\_performances = sorted(feature\_performances, key=lambda t: (t[1], t[2]))  
 with open(os.path.join(output\_dir, PERFORMANCE\_FILE), **'w'**) as performance\_file:  
 for method, ssdc, rand\_index in feature\_performances:  
 performance\_file.write(**f'**{method}**:** {round(ssdc)}**,** {rand\_index:**.4f**}\n**'**)

## Playing Card Image Region Labels

This is the human-labelled image required for the Rand index calculation in the code in Appendix 4-A. The image contains a single 8-bit channel with the pixel labels. (The image is intentionally dark.)

## Dugong Image Region Labels

This is the human-labelled image required for the Rand index calculation in the code in Appendix 4-A. The image contains a single 8-bit channel with the pixel labels. (The image is intentionally dark.)

##### Appendix Notes

The code in these appendices assumes the playing card and dugong images are located within a directory “data” in the current working directory, with the following files:

* “card.png” – the playing card image
* “dugong.jpg” – the dugong image
* “card-region\_labels.png” – the image provided in Appendix 4-B
* “dugong-region\_labels.png” – the image provided in Appendix 4-C.

The scaled and rotated versions of the images are produced to a directory “generated” in the current working directory.

Results are produced to a directory “results” in the current working directory.

1. The feature spaces were normalised to the range of [0, 1] for ease of comparison of the SSDCs. [↑](#footnote-ref-1)
2. The Rand index was derived from a human-labelled image, see Appendices 4-B and 4-C. [↑](#footnote-ref-2)