Preamble:

Module Code: CS3VI18
Assignment Report Title: Kinect-based Object Recognition
Student Number: ######
Date Completed: 24/03/21
Time Spent: ~25 Hours
Assignment Evaluation:

- Difficult and limited direction from either the lectures or specification. Feels a bit too difficult for an assignment only weighing 15% of the module.
- Interesting concept for the coursework. If we were given more direction, or longer, the coursework would be quite fun.
- The Microsoft Teams team was beneficial in asking questions.

Name	Contribution in %	Note	Signature
William Parker	100%	Code, Introduction, Methodology, Results / Discussion	WP
#####	100%	Document Review	#####
#####	100%	Abstract, Results/ Discussion, Conclusion, Document Review	#####

Abstract:

This report focuses on implementing a classification algorithm using depth and colour information gathered with an Xbox Kinect for object recognition. Training sets pertaining to 14 distinct items were provided to train the classification model. Hu moments and Zernike moments are calculated along with a principal component analysis of RGB data and a colour histogram. A random forest classifier containing 100 trees was provided with all of these components and the resulting classifications were calculated. The final classifier was found to have an accuracy of ~33%. Future improvements to this classifier are also discussed.

Introduction:

Visual intelligence is a long-standing field of computer science. Concerned with all aspects of computer vision from image enhancement to facial recognition.

This report explores the use of classifiers and features used to build them to accurately classify objects that are being presented to a Kinect Sensor camera. The assignment can be broken down into three major sections; gathering useful training data from the provided source, selecting the features from the training set and building the classifier, and applying the classifier to a second source and evaluating its accuracy.

Methodology:

Building the training set of Images:
Building the training set is an important part of this task as it's what teaches the classifier, so poor quality training data can cause the best tuned classifier to still perform badly. The process taken to build this training data relies on the depth data of the video stream from the Kinect to identify where the item is, if one appears.

The first step to do this, is to threshold the data, in such a way that only the closest parts of the depth image are shown, as demonstrated in **Fig.1.** Additional image processing such as erosion is used to close gaps and produce fewer, but larger, blobs [15].

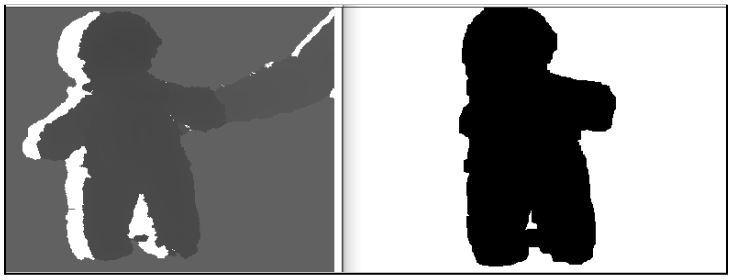

Figure. 1: *Depth Image before and after thresholding*

OpenCV's contours can then be used to identify the area in which the item is. These contours are filtered, to ensure only ones within a certain size range are recognised. This stops any large contours being found around the edge of the image, or small contours that we don't want as part of the training set.

Once accurate contours of the image has been found, a bounding rectangle can be produced that will outline where the image is. This bounding rectangle can then be applied to the RGB image, with an offset, to show the image area, as shown in **Fig.2.** This bounding rectangle is a representation of the area of both the depth image and the rgb image, that will be cropped to add to the training set.

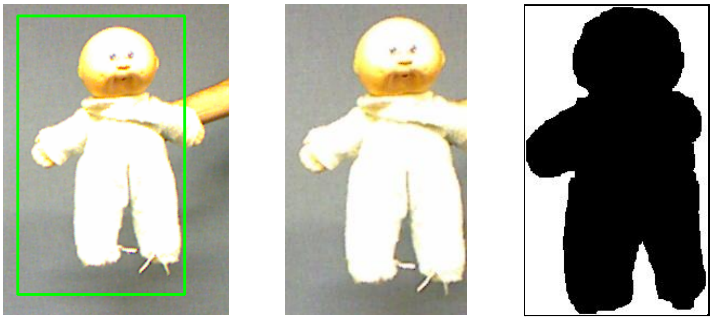

Figure 2: *The bounded item (left), and the corresponding rgb (middle) and depth (right) that are added to the training set.*

Using the provided text file that gives the order of the objects shown in the Kinect Video, the images are written to the folder corresponding to that object. (i.e. for **Fig.2**; *TrainingSet/Baby/depth/71.jpg* and *TrainingSet/Baby/rgb/71.jpg*). Due to the area of the bounding rectangle changing size, depending on the contours found, all images within the training set are resized to remain consistent.

Build Classifier:

To first start building a classifier, features of the training data need to be decided. For the best classifier, the features collected need to accurately reflect what distinguishes one object from the other, and this was taken into consideration when deciding them. In this project, once all features are found, they are reshaped to a vector and placed inside a numpy array called 'image-features'. The image-features array of all images is added to a global-features array, which is what the classifier trains from.

Hu Moments:

Image moments, in their simplest form, are a method of capturing information about the shape of a blob in a binary image [1]. Therefore, we can use moments to find information about the shape of the blob given in the depth image, post thresholding.

Hu Moments in particular are movements that are invariant to translation, rotation or scaling. This broadly means that only the shape of the object is taken into account, for the classifier; it doesn't matter if the shape is moved, scaled or rotated, as demonstrated in **Table 1.**. In this context, this is beneficial, as the object within the bounding rectangle can often translate elsewhere, or be of different scales and rotations, due to human error in holding the object up to a camera.

Hu Moments are a set of seven numbers that are calculated using normalised central moments. Moments in an image constitute a weighted average of the image pixel intensities, while the central moments is a function of a moment at a particular point in the image - therefore these moments depend on the intensity of pixels and their placement within the image.

id	Image	H[0]	H[1]	H[2]	H[3]	H[4]	H[5]	H[6]
K0		2.78871	6.50638	9.44249	9.84018	-19.593	-13.1205	19.6797
S0		2.67431	5.77446	9.90311	11.0016	-21.4722	-14.1102	22.0012
S1		2.67431	5.77446	9.90311	11.0016	-21.4722	-14.1102	22.0012
S2		2.65884	5.7358	9.66822	10.7427	-20.9914	-13.8694	21.3202
S3		2.66083	5.745	9.80616	10.8859	-21.2468	-13.9653	21.8214
S4		2.66083	5.745	9.80616	10.8859	-21.2468	-13.9653	-21.8214

Table 1: *The Hu-moments of various images [1]*

In this project, OpenCV was used to calculate the Hu-Moments of the image [2]. These were then added to an image-features array in the form of a 7-dimension vector.

Zernike Moments:
Zernike Moments are similar to Hu-Moments, in the sense that it is concerned with the shape detection like other image moments. In addition, Zernike moments are also invariant to translation, rotation, and scaling [16]. However, Zernike moments have additional desirable properties including a robustness to noise and fast computation [3]. This is desirable in this context as the thresholded depth data can often lead to shapes with noise.

Zernike moments are generally found to outperform Hu Moments [4]. Zernike moments are an orthogonal set of moments that are created from Zernike polynomials. These are polynomials that are continuous and orthogonal over a unit circle [5], and therefore have the advantage of needing a lower precision to represent differences [6].

While Zernike moments are often used for reconstruction of an input image [7], they can be used to identify images based on their shape, like Hu-Moments [8]. In this implementation, the Zernike moments of the depth images are found using the Mahotas library [17], which produces a 25-dimensional vector that can be added to an image's feature-array.

Principal Component Analysis:
Principal Component Analysis (PCA) is a process of transforming an appearance space into a new coordinate system. The major use of PCA is to reduce the dimensionality of data, such that the bulk of information about the data can be represented in a smaller space. For example, when applied to images, the first *x* number of Principal Components can taken and stored as a form of lossy compression, as demonstrated in **Fig.3**

Figure 3: *Compressed images with (left ot right) 20 ,50,100,200 principle components [18]*

As demonstrated by **Fig.3,** the essential parts of the image are stored in the first principle components, as the improvement between 20 and 50 components is much more significant than between 100 and 200 components. The more essential components are decided based on the ones that have the most variation in data.

Within this project context, the RGB images of the training data are converted to grayscale and reduced via PCA. This reduced image is then reshaped to 1D vector, that is added to the feature-array, as shown in **Fig.4**.

```
# Standardisation
data = image - np.mean(image, axis=0)
scatter_matrix = np.dot(data, data.T)
# Retrieve Eigenvectors and corresponding Eigenvalues
eig_val, eig_vec = np.linalg.eig(scatter_matrix)
# Get and store the 1st component
new_reduced_data = np.real(np.sqrt(eig_val[0]) * eig_vec.T[0].reshape(-1,1))
return np.squeeze(new_reduced_data)
```

Figure 4: *Code to convert an Image to it's 1st Principle Component (1-D Vector) [20] [19]*

Colour histogram:

A colour histogram is a representation of the distribution of colours in an image. This implementation utilises the RGB images in the data set, to add to the classifier as a way of identifying an item through colour, as colour is often an important aspect in identifying items.

Colour histograms are often used as descriptors in classifiers [9], either in the RGB or HSV colour space. **Fig 5.** demonstrates the code used to produce the colour histogram and put it in a format for the image's feature-array.

```
# Colour Hist of input Image, all channels, bins = 8 for all, and range of 0-256 for all
hist  = cv2.calcHist([image],[0,1,2],None,[8,8,8], [0, 256, 0, 256, 0, 256])
cv2.normalize(hist,hist)
return hist.flatten()
```

Figure 5: *Code to get an images Colour Histogram, as a 1-D Vector..*

Random Forest Classifier:

Random forests are a supervised learning algorithm, used both for classification and regression [10]. Random Forests will create decision trees on randomly selected data samples and get a prediction result from each decision tree. A vote then takes place for each prediction result. The result with the most votes is the final prediction. This process is visualised in **Fig 6**.

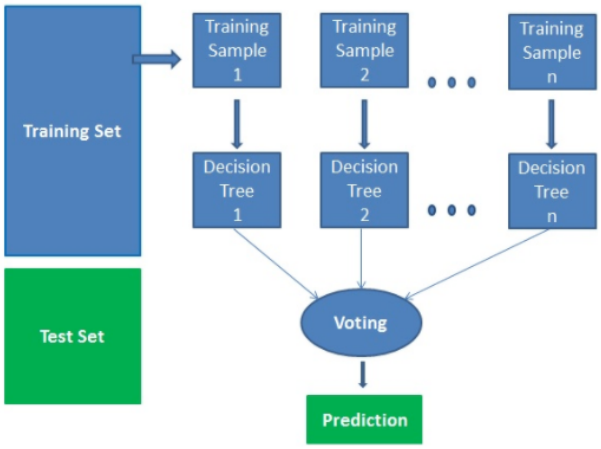

Figure 6: *Visualisation of a Random Forest Classifier. [10]*

The number of decision trees in the forest often have an affect on the prediction outcomes. Generally, the more trees, the more accurate the prediction. This comes with a trade off for longer times to build the random forest classifier, and longer time for a prediction, which can cause an issue in this project, due to the input data being frames of a continuous video. 100 trees were ultimately used to build the random forest classifier.

For this project, the random forest classifier has been implemented using the sklearn library [11]. The data is fed into the classifier via 2 *h5py* files [12]. Then, for each frame in the playing video, the features described above are found and passed to the classifier. The prediction of the classifier is then shown above the object, as demonstrated in **Fig.7**.

Significant help for the random forest classifier implementation was gained from [13].

Results Evaluation and Discussion:

The results were gained by setting the '`VALIDATION`' variable at the top of *cv-demo.py* to `True` [20]. When run on the dataset *Set2/*, the object is identified by the label that is written above the bounding rectangle the most, as demonstrated in **Fig.7**.

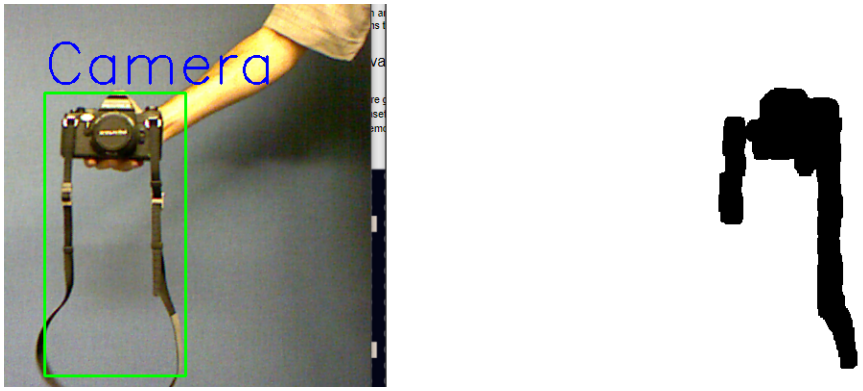

Figure 7: *Demonstration of the Camera being correctly identified by the Classifier.*

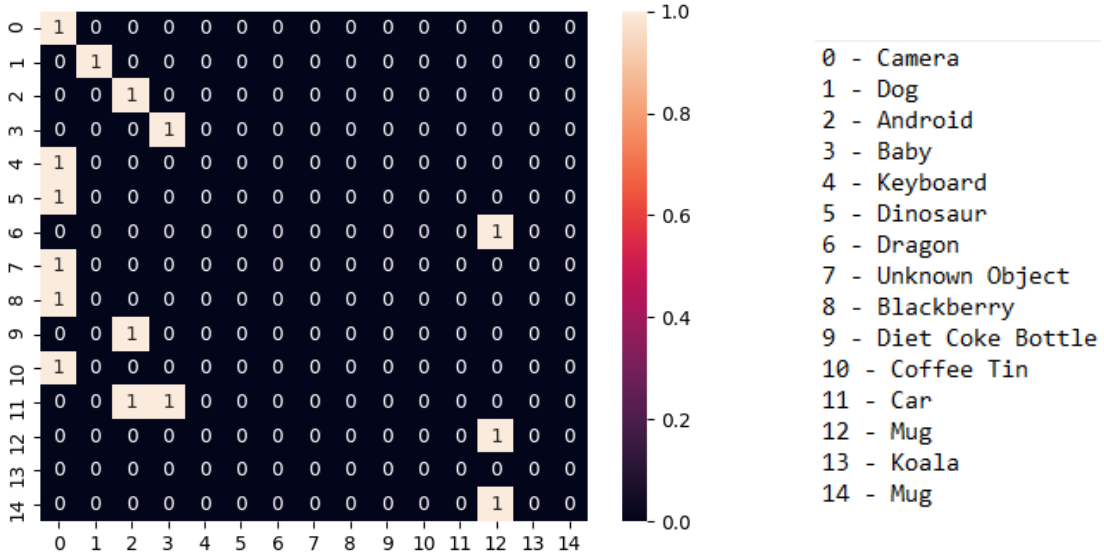

Figure 8: *Confusion Matrix on the second data set*

Fig.8 shows a confusion matrix of the outcome, when running the classifier against the dataset *Set2*. An object's overall prediction was manually determined by taking the most common prediction given by the classifier, as the object was onscreen. In addition, due to the nature of random forest classifiers, the outcome was often fairly erratic.
The classifier was able to accurately predict 5 out of 15 images, for an accuracy of ~33%. Consistently, objects were incorrectly identified as either the camera or the android phone.

The implementation stands to be improved significantly. Firstly, putting the principal components of the RGB into the 'image-features' array, will dwarf the other descriptors as it produces a vector up to 100x bigger than the others. Instead, PCA should be used to reduce images to be put through other descriptors.
The number of intervals (bins) in the colour histogram could also be increased. It can be seen between objects that the difference between the colours are very subtle, with most objects falling into one of two categories - being mostly black (such as the 'Android' and 'Camera' objects), or mostly beige (such as the 'Koala', and 'Baby' object). Increasing the number of intervals would allow the classifier to differentiate the colours to a better degree.

For the selection of which descriptors to give to the classifier, a standard such as MPEG-7 could have been followed [14]. The standard defines descriptors categories such as Colour, Texture, Shape and Motion. Had 1 descriptor for each of these categories be implemented, the objects may have been more easily discernible for the classifier.
Finally, an improvement could be made to the training set. The images produced from the training data often contained instances where the object was not fully in frame, or the bounding rectangle didn't capture the whole object. Removing these images could further make the difference between the shape of objects easier for the classifier to discern.

Conclusion:

To conclude, it was found that using Hu moments, Zernike Moments, principal component analysis and an RGB histogram as input attributes, a random forest classifier with 100 decision trees was able to correctly classify a set of 14 objects in a video ~33% of the time. While the results are far from optimal, the classification still manages to perform at a level far above a random classification (~7%). Improving on the model could come through many different methods, for example, changing the input attributes, rescaling the random forest to contain more trees, or even using a different classification model altogether. Overall, the implementation and results provide excellent groundwork for further development of an object classifier.

References:

1. Mallick, S., 2018. *Shape Matching using Hu Moments (C++ / Python) | Learn OpenCV*. [online] Learn OpenCV | OpenCV, PyTorch, Keras, Tensorflow examples and tutorials. Available at: <https://learnopencv.com/shape-matching-using-hu-moments-c-python/> [Accessed 23 March 2021].

2. Docs.opencv.org. 2019. *Structural Analysis and Shape Descriptors — OpenCV 2.4.13.7 documentation*. [online] Available at: <https://docs.opencv.org/2.4/modules/imgproc/doc/structural_analysis_and_shape_descriptors.html?highlight=cvmatchshapes#humoments> [Accessed 23 March 2021].

3. k.Whoi-Yul,K.Yong-Sung. 2000. *A region-based shape descriptor using Zernike moments*. [publication] Available at: <https://www.sciencedirect.com/science/article/pii/S0923596500000199> [Accessed 23 March 2021].

4. Sabhara, Reza. (2013). Comparative Study of Hu Moments and Zernike Moments in Object Recognition. The Smart Computing Review. 3. 10.6029/smartcr.2013.03.003. Available at: <https://www.researchgate.net/publication/272672013_Comparative_Study_of_Hu_Moments_and_Zernike_Moments_in_Object_Recognition> [Accessed 23 March 2021].

5. Lakshminarayanan, Vasudevan & Fleck, Andre. (2011). Zernike polynomials: A guide. Journal of Modern Optics - J MOD OPTIC. 58. 1678-1678. 10.1080/09500340.2011.633763. Available at: <https://www.researchgate.net/publication/241585467_Zernike_polynomials_A_guide> [Accessed 23 March 2021].

6. Shutler, J., 2002. *Orthogonal moments*. [online] Homepages.inf.ed.ac.uk. Available at: <https://homepages.inf.ed.ac.uk/rbf/CVonline/LOCAL_COPIES/SHUTLER3/node9.html> [Accessed 23 March 2021].

7. Amayeh, Gholamreza & Erol, Ali & Bebis, George & Nicolescu, Mircea. (2005). Accurate and Efficient Computation of High Order Zernike Moments. 3804. 462-469. 10.1007/11595755_56. Available at: <https://www.researchgate.net/publication/220845163_Accurate_and_Efficient_Computation_of_High_Order_Zernike_Moments> [Accessed 23 March 2021].

8. Rosebrock, A., 2014. *HOW-TO: Indexing an image dataset using Zernike moments and shape descriptors.*. [online] PyImageSearch. Available at: <https://www.pyimagesearch.com/2014/04/07/building-pokedex-python-indexing-sprites-using-shape-descriptors-step-3-6/> [Accessed 23 March 2021].

9. J.P.Figueroa, V.R.Bykbaev,. 2012. *Image Retrieval Based on the Combination of RGB and HSV's Histograms and Colour Layout Descriptor* [publication]. Available at: <https://core.ac.uk/download/pdf/333533097.pdf> [Accessed 23 March 2021].

10. A.Navlani. 2018. *Understanding Random Forests Classifiers in Python.* (website). Available at: <https://www.datacamp.com/community/tutorials/random-forests-classifier-python> [Accessed 23 March 2021]

11. Scikit-learn.org. 2020. *sklearn.ensemble.RandomForestClassifier — scikit-learn 0.24.1 documentation*. [online] Available at: <https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.RandomForestClassifier.html> [Accessed 23 March 2021].

12. H5py.org. 2021. *HDF5 for Python*. [online] Available at: <https://www.h5py.org/> [Accessed 23 March 2021].

13. S.Singh. 2018. *Random-Forest-Image-Classification-using-Python.py*. [online] Available at <https://github.com/87surendra/Random-Forest-Image-Classification-using-Python/blob/master/Random-Forest-Image-Classification-using-Python.py/> [Accessed 23 March 2021].

14. Mpeg.chiariglione.org. 2008. *Visual | MPEG*. [online] Available at: <https://mpeg.chiariglione.org/standards/mpeg-7/visual> [Accessed 23 March 2021].

15. Docs.opencv.org. 2021. *OpenCV: Eroding and Dilating*. [online] Available at: <https://docs.opencv.org/3.4/db/df6/tutorial_erosion_dilatation.html> [Accessed 23 March 2021].

16. A. Khotanzad and Y. H. Hong, "Invariant image recognition by Zernike moments," in *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 12, no. 5, pp. 489-497, May 1990, doi: 10.1109/34.55109. Available at: https://ieeexplore.ieee.org/document/55109 [Accessed 23 March 2021]

17. GeeksforGeeks. 2021. *Mahotas - Zernike Moments - GeeksforGeeks*. [online] Available at: <https://www.geeksforgeeks.org/mahotas-zernike-moments/> [Accessed 23 March 2021].

18. Wasnik, A., 2021. *Principal Component Analysis For Image Data in Python - AskPython*. [online] AskPython. Available at: <https://www.askpython.com/python/examples/principal-component-analysis-for-image-data> [Accessed 23 March 2021].

19. 'Pgrenholm'. 2017. *Principal Component Analysis converts 3d array to 1d array using Python* . [online]. StackOverflow. Available at:<https://stackoverflow.com/a/43446362> [Accessed 23 March 2021]

20. https://csgitlab.reading.ac.uk/CS3VI18-2020-2021/group_13 [Code associated with Project]