L-Università ta' Malta
Faculty of Information & Communication Technology

Assignment – Principles of Computer Vision

Daniel Calafato 31002L
Liam Bugeja Douglas 17002L

Lecturer: Dr. Dylan Seychell
Course Code: ARI 2129
Date: 19/06/2022

FACULTY OF INFORMATION AND COMMUNICATION TECHNOLOGY

Declaration

Plagiarism is defined as “the unacknowledged use, as one’s own work, of work of another person, whether or not such work has been published” (Regulations Governing Conduct at Examinations, 1997, Regulation 1 (viii), University of Malta).

I / We*, the undersigned, declare that the [assignment / Assigned Practical Task report / Final Year Project report] submitted is my / our* work, except where acknowledged and referenced.

I / We* understand that the penalties for making a false declaration may include, but are not limited to, loss of marks; cancellation of examination results; enforced suspension of studies; or expulsion from the degree programme.

Work submitted without this signed declaration will not be corrected, and will be given zero marks.

(N.B. If the assignment is meant to be submitted anonymously, please sign this form and submit it to the Departmental Officer separately from the assignment).

Daniel Calafato	[illegible]
Student Name	Signature
Liam Bugeja Douglas	[illegible]
Student Name	Signature

ARI2129	Computer Vision Assignment
Course Code	Title of work submitted

18/06/2022

Date

Contents

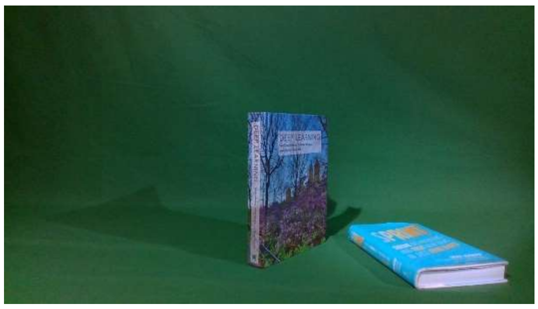
Part 1 – Object Blending

This part of the assignment consisted of manipulating images by blending objects into images using predefined OpenCV[1] functions. This was done by first extracting the object from an image using it mask, then a filter is applied and blended into the new image. The images used in this assignment can be found in the COTS dataset [2].

Stage 1

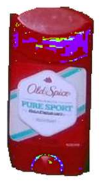
As previously discussed, this part of the assignment consisted of extracting objects from images and blending them into new ones. This was done using the following functions: ExtractObject(), ApplyFilter(), ObjectBlender(), ComapreResults().

The goal of this part was to achieve a similar result below:



Where the standing book is perfectly blended into the second image.

The first function ExtractedObject() takes an image and the mask of an object as a parameter, and extracts the object. This is done by multiplying the image colours and object colours channel, this will create a new image with the target object and a black background. Finally, the function bitwise_not is used to invert the colours of the new image.



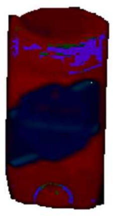
Next the function ApplyFilter() is used, which takes the extracted object and an index as parameters. The index can give a range between 0 and 3 and specifies the filter to be used, in this assignment the following were used: No filter, Opening filter, Gaussian blur and Histogram Equalisation filter. No filter returns the extracted object as is and is the first choice (0). Opening is a morphology operation which first dilates and erodes an image, by doing this it removes small objects from the background, it is the second choice (1). Next choice (2) is Gaussian blur, this filter blurs the image and reduces the background noise whilst also reduces the detail of the image. Finally, the last choice (3) is the Histogram Equalisation filter, this filter improves the contrast in the image by spreading the most frequent intensity values in the image.

No Filter:

Opening Filter:

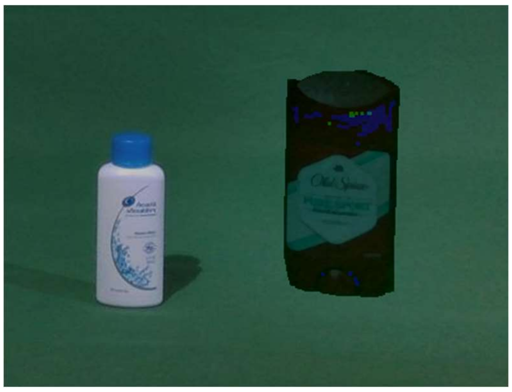
Gaussian Blur:

Histogram Equalisation filter:



The next function takes care of blending the extracted object with the filter in the new image, the function is called ObjectBlender() and takes an image and the filtered extracted object as parameters. This was done by using the addWeighted() function by adding the two images together and pass the alpha, beta and gamma values.

Extracted object using opening filter blended into new image:



The final function takes the blended image, the original image and a metric choice as parameters, it is called CompareResults(). The original image and the blended ones are compared using the Mean Squared Error (MSE) and the Sum of Squared Distance Error (SSD).

The following results were obtained:

Filter	MSE	SSD
No Filter	938.03	144692003
Opening Filter	946.14	144695891
Gaussian Blur	939.99	144819905
Histogram Equalisation Filter	1295.06	144928653

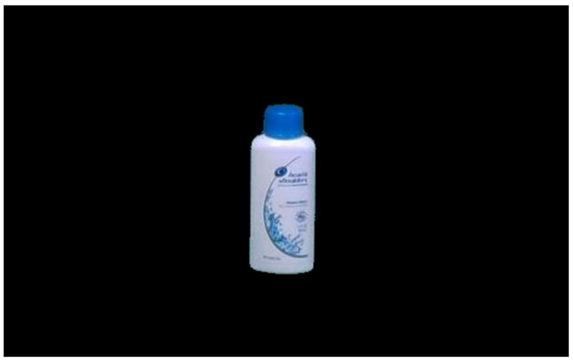
The best results were given by applying no filter on the extracted object, this is due to the fact that both images were taken in the same environment and lighting.

Stage 2

The following stage removes the green background of the images and place the objects in a new background. This was done by first removing the green background from the images and then merge the objects with a new background.

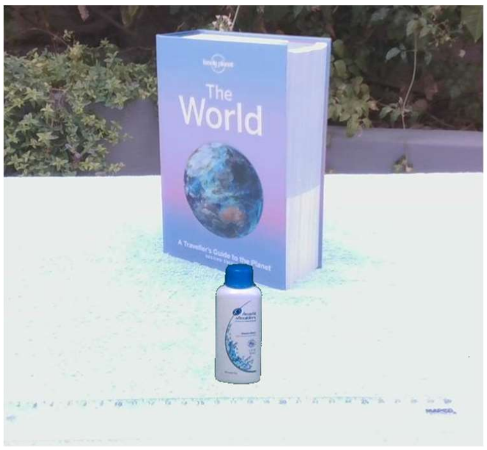
The first function, RemoveGreen() takes only an image as a parameter. First, the upper and lower bound of the green background is defined, after some testing the best ranges where the following: Lower Bound = [0,25,0] , Upper Bound = [100,120,100]. Next the mask of the background is created by using the predefined function of OpenCV called inRange() which takes an image and the desired upper and lower bound. Finally, the image is compared with the mask and the background is removed.

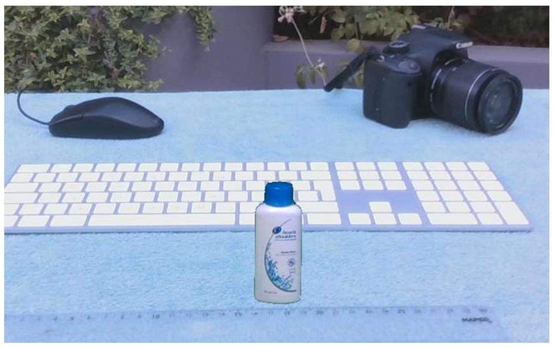
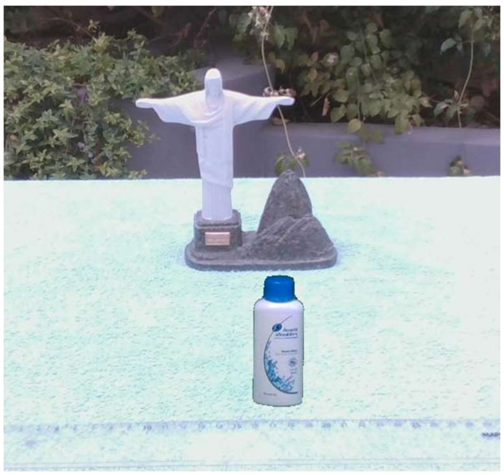
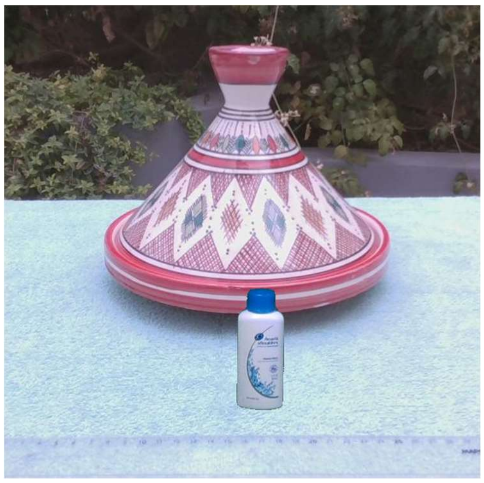
Image with background removal:



The next function called NewBackground() takes the image with the removed background and the new background image as parameters. Next the new background is resized to the size of the extracted image, finally the extracted image and background image are combined as an np.array due to the multiple colour channels.

Object with new background:



The shampoo bottle has blended greatly with the new backgrounds, there is minimal green colour from the previous background that can only be seen when looking closely.

Part 2 – Image Inpainting

The goal of this part of the assignment was to ‘delete’ an object given its mask, by inpainting what should have been behind it. The COTS dataset [2] which contains images with and without objects. These were then used to compare the obtained inpainting results to how they were supposed to be. Two inpainting methods were used during this process, these are the ‘*Telea*’ and ‘*NS*’ inpainting methods.

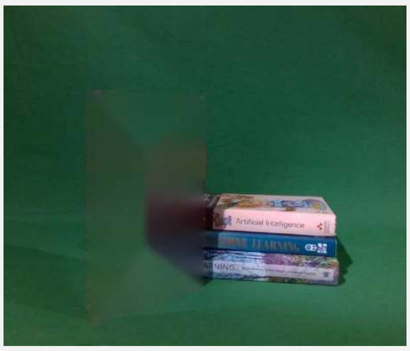
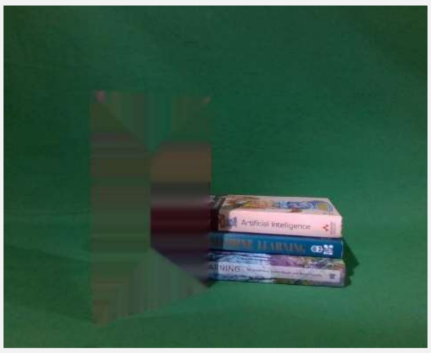
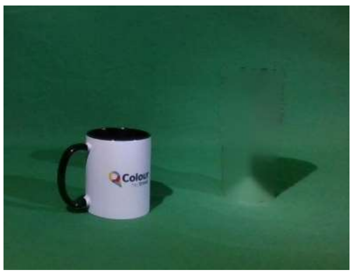
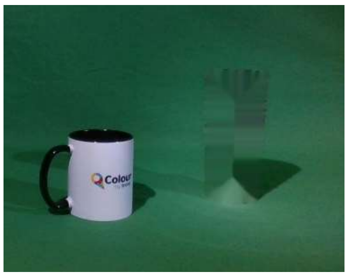
The Telea method starts from the outer edges of the mask to be inpainted, since these will be the most accurate and fills in the pixels depending on their neighbouring pixels depending on their weight, moving on to the inner pixels afterwards. [1]

The NS method preserves edge like features which tries to map them from were they start at one end to were they end, which may not always be a straight line. This method uses a heuristic to accomplish this since mapping curved lines may not be an easy feat. The rest of the mask is drawn in by matching colours with equal intensities. [3]

1.2.1 Inpainting with a Simple Background

To accomplish this, an OpenCV [1] function was used, this is the ‘*inpaint*’ function. It was tested on six categories from the dataset, being ‘*Academic*’, ‘*Footwear*’, ‘*Mugs*’, ‘*Shooter*’, ‘*Statues*’, and ‘*Tech*’*,* using both of the previously mentioned methods. The function was given the original image to inpaint, the mask for the object to be inpainted, a radius of 20 and one of the inpainted methods as parameters. A radius of 20-pixels was selected since selecting a small number of pixels as a neighbourhood for 1280 x 720-pixel images would have fewer pixels to base its ‘prediction’ on, and hence being less accurate.

Some examples of the results are shown below:

TELEA	**NS**
	
	

The results based on the variance from the original image were obtained using the results of the CompareResult function used in Part 1 of the assignment which returns the Mean Squared Error and the Sum of Squared Distance Error.

Mean Squared Error(MSE) works by by first obtaining the distance of the predicition from the regression line and squaring it to remove any negative numbers. These valuse are then summed and divided by the number of prediction which results in the average error, hence the name Mean Squared Error. Sum of Squared Distance Error is similar to the fact that sum of the square is returned but no averaging is done.

The results for the 6 categories are as follows:

Academic

	MSE	*SSD*
Telea	326.55285	60084036
NS	379.89933	62151316

Footwear

	MSE	*SSD*
Telea	1518.22525	66051304
NS	1503.99133	66550990

Mugs

	MSE	*SSD*
Telea	76.4463	39278766
NS	85.49626	41437830

Shooters

	MSE	*SSD*
Telea	59.85807	34284561
NS	59.56621	34618458

Statues

	MSE	*SSD*
Telea	459.77335	116140559
NS	516.36609	116953580

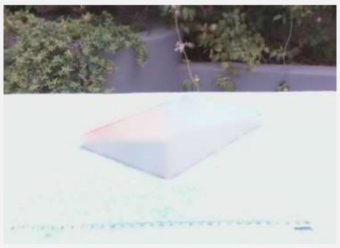
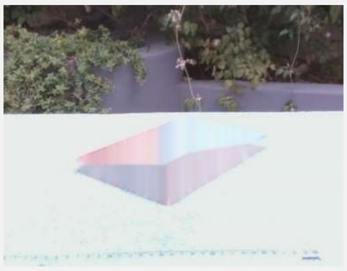
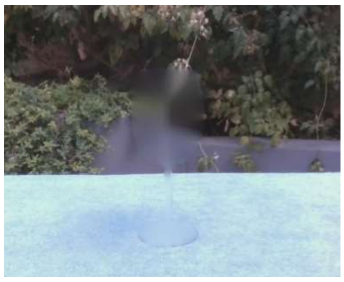
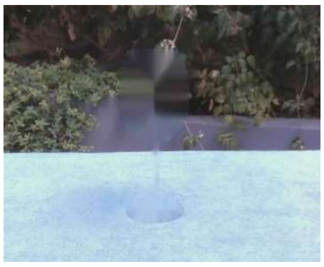
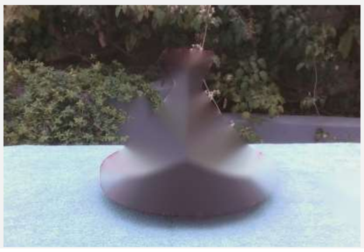
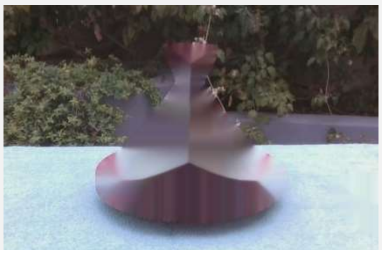
Tech

	MSE	*SSD*
Telea	119.56662	39895768
NS	146.37125	41270141

Both methods yielded reasonably good result, with some having minimal differences between the two methods. However, the Telea method proved to produce better results, both visiually and also by the error values, besting the NS method in four out of the six images. The Telea images produced a smoother and more suttle version than the straight and ‘polygonal style’ of the NS, which might be why some images had a lower score.

1.2.2 Inpainting with a Complex Background

The second part of this section was to repeat this process with a dataset that had a more complex background for seven categories, which are ‘Books’, ‘Bottles’, ‘Cups’, ‘Electronics’, ‘Food’, ‘Souvenirs’, ‘Statues’. The following are some of the produced results:

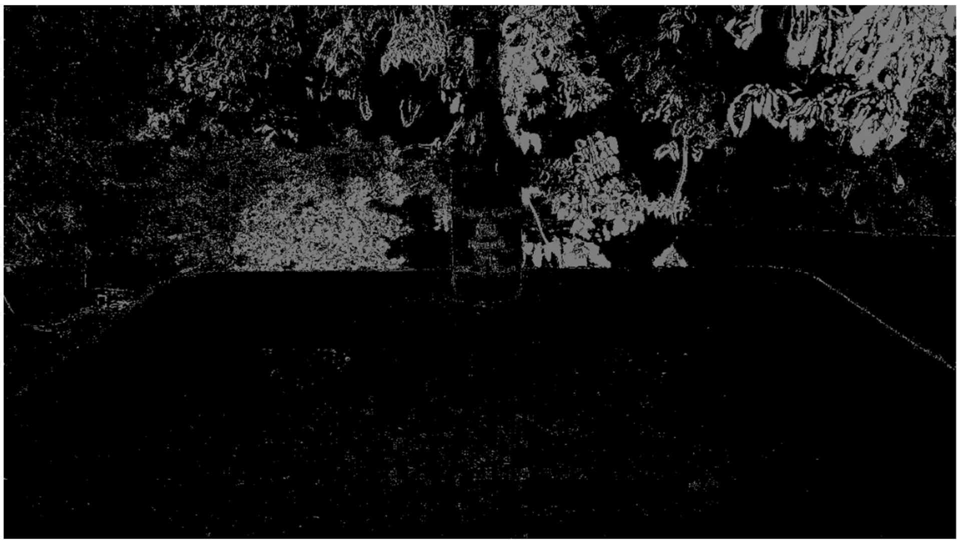
TELEA	**NS**
	
	
	

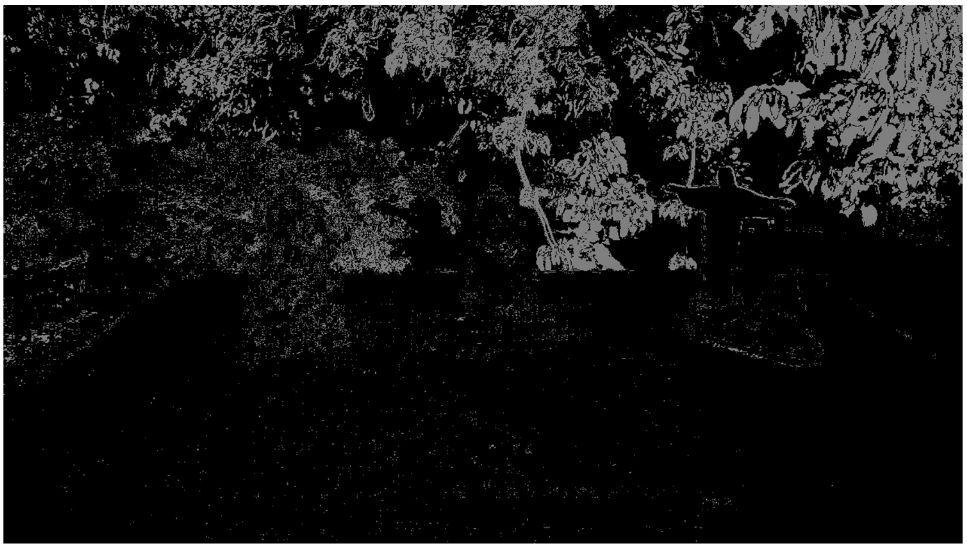
Having a more complex background proved to be a harder feat for both methods producing an average MSE for the seven categories of 1,709.2298 and 2,102.74485 for the Telea and NS methods respectively. This differs greatly from the 426.73707 and 448.61507 MSE averages produced previously by the more plain and simple background. The results also varies by the size of the object to be inpatined, producing better results for smaller ones which is as expected. Furthermore, it strengthened the fact that the Telea method seemed to produce a better result overall.

1.2.3 Changing Background

The last part of this section of the assignment was an evaluation for changing or movement in background was required. This made use of a background subtractor, createBackgroundSubtractorMOG2() [4], in brief terms subtracts an image from another, resulting in those pixels that remained static as black, while giving a colour for those that did not.

This was then used to detect the difference between images with and without wind. This method can further be used for Image Segmentation and Object Detection to mask those pixels which are changing and marking them as the foreground. In our case, the main objects which were effected are the leaves which are clearly visiable in the images.





References

[1] OpenCV Library, “Home,” *OpenCV*, 09-Feb-2021. [Online]. Available: https://opencv.org/. [Accessed: 18-Jun-2022].
[2] D. Seychell, *COTSDataset: A Multipurpose RGB-D Dataset for Inpainting and Saliency Applications*.
[3] S. Mallick, “Image inpainting with OpenCV (C++/Python),” *LearnOpenCV – OpenCV, PyTorch, Keras, Tensorflow examples and tutorials*, 02-Apr-2019. [Online]. Available: https://learnopencv.com/image-inpainting-with-opencv-c-python/. [Accessed: 18-Jun-2022].
[4] “OpenCV: cv::BackgroundSubtractorMOG2 Class Reference,” *Opencv.org*. [Online]. Available: https://docs.opencv.org/4.x/d7/d7b/classcv_1_1BackgroundSubtractorMOG2.html. [Accessed: 18-Jun-2022].