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

ARI2129 - Principles of Computer Vision for AI

Cristina Cutajar* (230802L), Britney Vella (188102L), Gabriel Vella* (400002L), Dylan Zerafa* (348002L)

*B.Sc. (Hons) Artificial Intelligence

Study-unit: **Principles of Computer Vision for AI**

Code: **ARI2129**

Lecturer: **Dr Dylan Seychell**

Plagiarism Form

FACULTY OF INFORMATION AND COMMUNICATION TECHNOLOGY

Declaration

Plagiarism is defined as "the unacknowledged use, as one's own, of work of another person, whether or not such work has been published, and as may be further elaborated in Faculty or University guidelines" (University Assessment Regulations, 2009, Regulation 39 (b)(i), University of Malta).

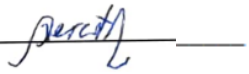
I / We*, the undersigned, declare that the [assignment / Assigned Practical Task report / Final Year Project report] submitted is my / our* work, except where acknowledged and referenced.

I / We* understand that the penalties for committing a breach of the regulations include 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.

* Delete as appropriate.

(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).

Cristina Cutajar	
Student Name	Signature
	
Student Name	Signature
Gabriel Vella	
Student Name	Signature
Britney Vella	
Student Name	Signature

ARI2129	Principles of Computer Vision for AI
Course Code	Title of work submitted

18/06/2022

Date

Part 1:

Stage 1:

Object Extraction

Object extraction is the first step of Object Blending where the mask of a specific object is used to obtain the object from a specific scene image. Since the mask would only contain 1 bit values for the intended object and 0 bit values for the unwanted parts of the image, the result of multiplying the mask of an object from scene 1 to the image of scene 2 would be an extracted image of the masked object.

In this case the extracted image is represented in Figure 2 which was taken from the scene in Figure 1 by using the target mask in Figure 3:

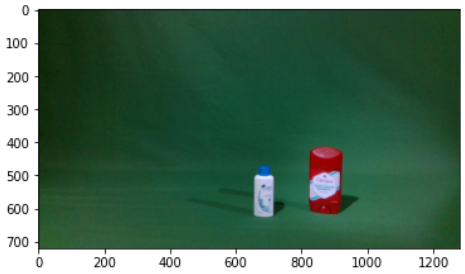


Figure 1

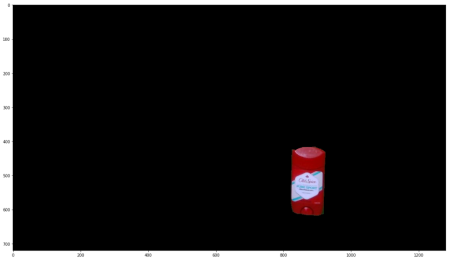


Figure 2

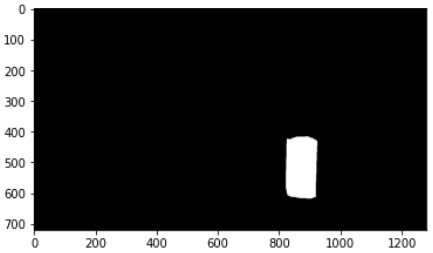


Figure 3

Image Filters

Three filters are then applied on the extracted object image by using different kernels. In this project the following kernels were used: Gaussian Blur, Sobel X, and Box Filter.

The Gaussian Blur is used for smoothing where it helps in reducing the noise in the image by altering the value of standard deviation. The OpenCV function 'GaussianBlur()' was applied on the extracted image with a chosen kernel size of '3x3'. This resulted in the image being blurry as seen in Figure 5. The Sobel X kernel is used to detect horizontal edges in an image. The OpenCV function 'Sobel()' was used to apply convolution automatically by the sobel kernel. This resulted in the image showcasing only the detected edges as seen in Figure 6. The Box Filter is also used to remove noise by smoothing the image but works differently to Gaussian as it makes use of averaging. This resulted in the image being blurry as seen in Figure 7. As you can see both blurry images shown in Figure 5 and 7 are similar to each other but computed differently.

In our program each kernel is assigned an index where 0 represents the original image (Figure 4), 1 represents the gaussian kernel, 2 represents the sobel x, and 3 represents the box filter.

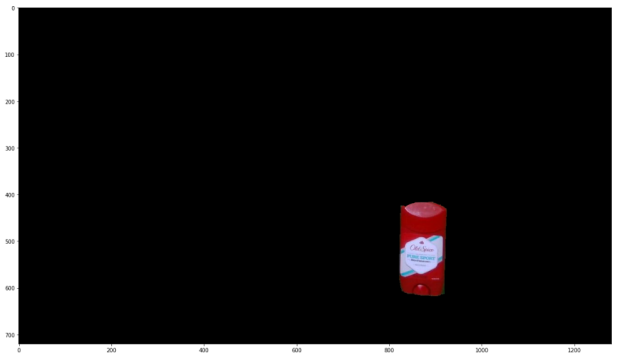


Figure 4

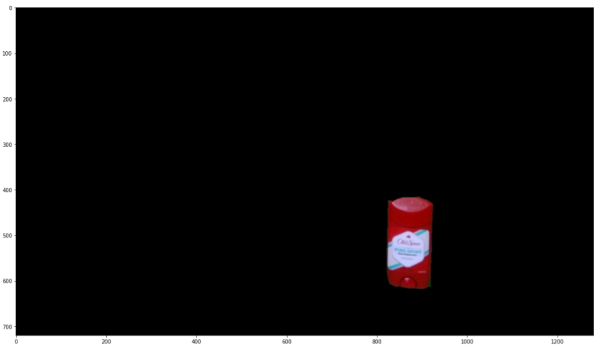


Figure 5

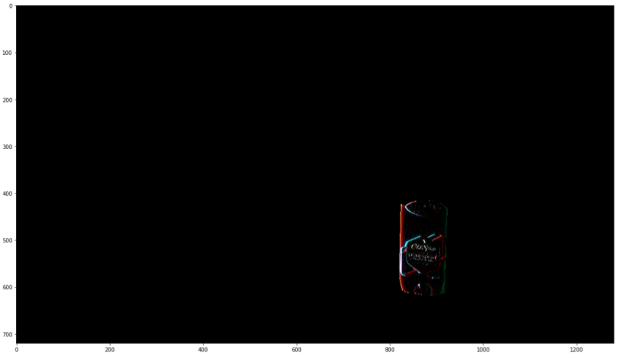


Figure 6

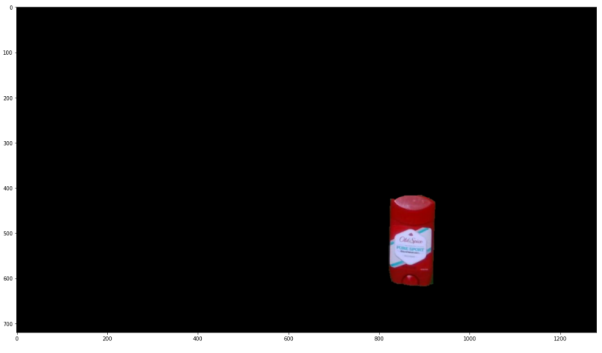


Figure 7

Object Blending

Object Blending is the process of combining the extracted image with the new scene to form a new image. This is done by using the Addition Blending technique which works by using two weight parameters for summation of images. By tweaking these parameters we can determine the level of opaqueness for each of the images until a good combination is shown. The function 'addWeighted()' from OpenCV was used where the final parameters ended up being 0.6 and 0.8. This process is applied for all the filtered images obtained from the previous section, resulting in images shown in Figures 8, 9, 10, 11.

Figure 8

Figure 9

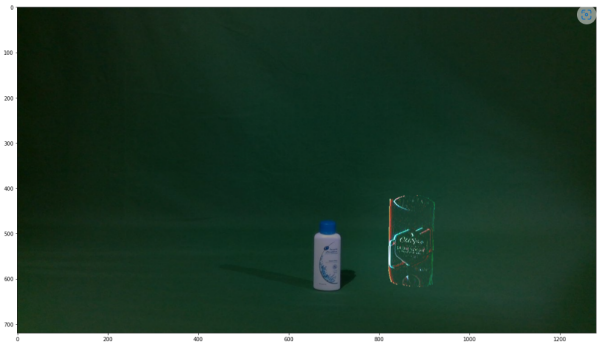


Figure 10

Figure 11

Comparing Results:

The blended result can then be compared to the original image having both objects by using error metrics like SSD (Sum of Squared Differences) and MSE (Mean Squared Error).

The results obtained are:

SSD:
Unfiltered Image: 272425228
Gaussian Image: 272427284
Sobel X Image: 272528257
Box Filtered Image: 272449608

MSE:
Unfiltered Image: 98.5334302662037
Gaussian Image: 98.53417390046296
Sobel X Image: 98.57069480613426
Box Filtered Image: 98.54224826388888

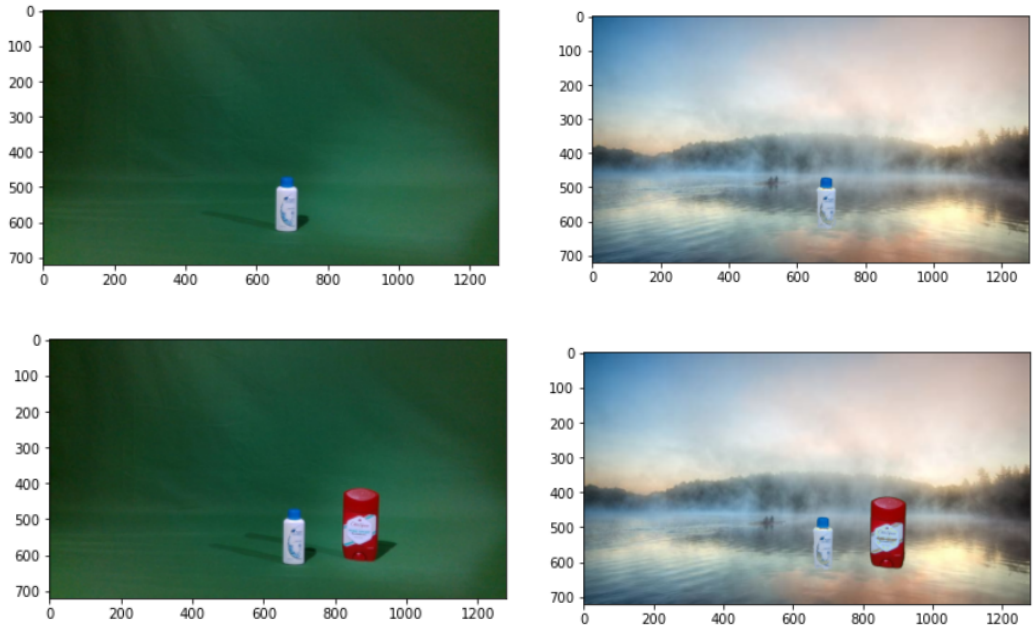
As you can see, the results obtained from MSE when compared to SSD all show that the blended images obtained are similar to their original ones since the closer the value is to 0 the more similar they are to their original image in MSE. Since the SSD measures similarity based on pixel by pixel intensity differences, the value is greater than that calculated by MSE as the MSE takes into consideration the corresponding pixels and calculates the square of difference of the pixel values.

Stage 2:

For this part of the assignment, it was required to create a function RemoveGreen() which returns the given image without the green background. This was done by first converting the image into a lab image. A threshold function was then applied and using the bitwise_and cv2 function, a mask containing the dark background is found. In addition, another mask with the white background was also found by setting the 0 values of the threshold image to 255. To remove the green shade along the border, the mask containing the dark background was converted to a lab and then normalised. Afterwards, another two masks were created using the threshold function. Finally, the lab image is converted back into bgr and the pixels that were dark in the final threshold image were set to white.

The NewBackground() function takes as input an image without the background and the new background image. The new background image is resized to the shape of the original image and then added on the original image where the values are equal to 255.

These functions were tested on both scene 1 and scene 2 images using 4 different backgrounds. Below one can see scene 1 and scene 2 and the results after changing to the first background image.



Part 2 :

Task A :

In this section of the assignment we choose 6 sets of images each containing a different object type. These where :

1. Academic Books
2. Statues
3. Mugs
4. Tech
5. Footwear
6. Shooters

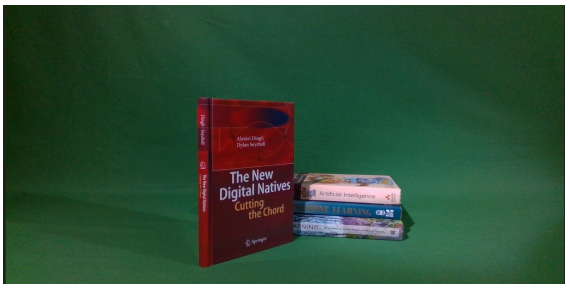
Each set contains 3 images in total.For example a Scene1 for Academic Books a Scene2 and a mask. The difference being between Scene1 and Scene 2 is that Scene1 contains one less object then Scene2. We are making use of OpenCv's inpaint() function .

```
def image_inpainting(image,mask,choice):
    if choice==1:
        Ours_Telea= cv2.inpaint(image, mask,3,flags=cv2.INPAINT_TELEA)
        return Ours_Telea

    if choice==2:
        Ours_NS = cv2.inpaint(image, mask,3,flags=cv2.INPAINT_NS)
        return Ours_NS
```

The user can choose between two options, based on how he would like whether it's TELEA or NS. Scene2 and its mask is passed and the returned Image is an image Scene1 since through inpainting the extra object that was present in Scene2 is now removed. This holds for both when cv2.INPAINT_TELEA and cv2.INPAINT_NS.

Original Image Scene1 : **Original Image Scene2 :**



Example of output image , using cv2.INPAINT_TELEA :

Example of output image , using cv2.INPAINT_NS :

As can be seen, the red book is removed and the resultant image is similar to Scene1. However one can notice between the resultant image when using cv2.INPAINT_TELEA and cv2.INPAINT_NS, this is due to the way each respective algorithm works.

cv2.INPAINT_TELEA :

This Algorithm starts from the boundary of this region and goes inside the region gradually filling everything in the boundary first. It takes a small neighbourhood around the pixel on the neighbourhood to be inpainted. This pixel is replaced by the normalised weighted sum of all the known pixels in the neighbourhood. Selection of the weights is an important matter. More weightage is given to those pixels lying near to the point, near to the normal of the boundary and those lying on the boundary contours. Once a pixel is inpainted, it moves to next nearest pixel using Fast Marching Method. FMM ensures those pixels near the known pixels are inpainted first, so that it just works like a manual heuristic operation.

cv2.INPAINT_NS :

This algorithm is based on fluid dynamics and utilises partial differential equations. Basic principle is heuristic. It first travels along the edges from known regions to unknown regions (because edges are meant to be continuous). It continues isophotes (lines joining points with same intensity, just like contours joins points with same elevation) while matching gradient

vectors at the boundary of the inpainting region. For this, some methods from fluid dynamics are used. Once they are obtained, colour is filled to reduce minimum variance in that area.

Comparison :

By making use of the Mean squared error we are then comparing the difference between Scene1 and the output of images which are attained after inpainting. A value close to zero indicates that there is very little difference between the two photos

Task B:

For this section, we were required to use the code implemented in Task A on a new part of the dataset. We made use of 6 sets to evaluate the inpainting algorithms as well as to experiment with different visualisation methods of background changing. All the images for each set were downloaded from the complex background folder of the COTS dataset. Each set contains different instances that include “NW”, “W” and “NO” images, as required.

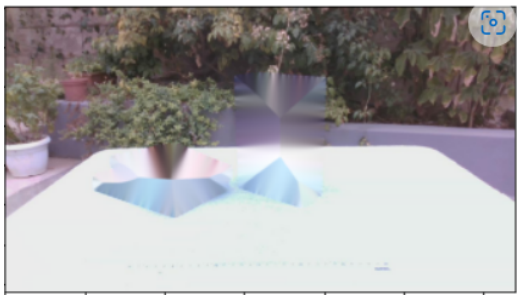
The 6 sets used for this task were:

1. BooksA
2. BooksB
3. BottlesA
4. CupsA
5. ElectronicsA
6. StatuesB

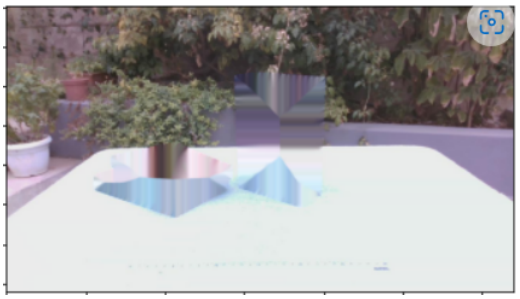
Each set contains 4 images in total; an image with NW background, another instance with W background and their respective masks.

The masks for the images were used to achieve the first part of this task which included evaluating the inpainting algorithms developed in Task A on these 6 sets. The image_impainting() function was used to perform image inpainting evaluation using both Ours+Telea and Ours+NS on all of the “NW” and “W” instances images found in the six sets.

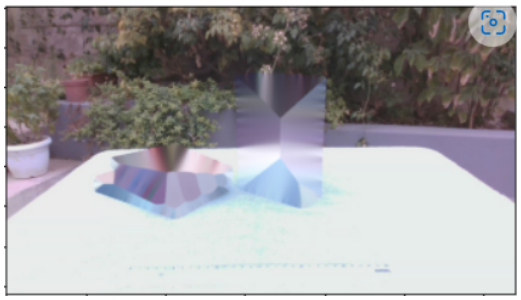
BooksA NW Ours+Telea:



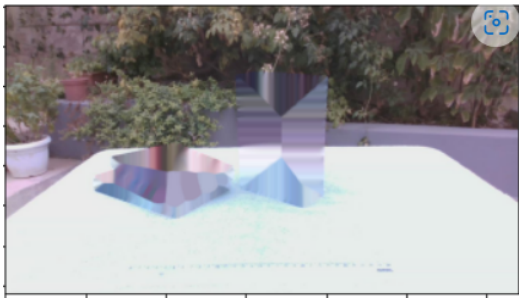
BooksA NW Ours+NS:



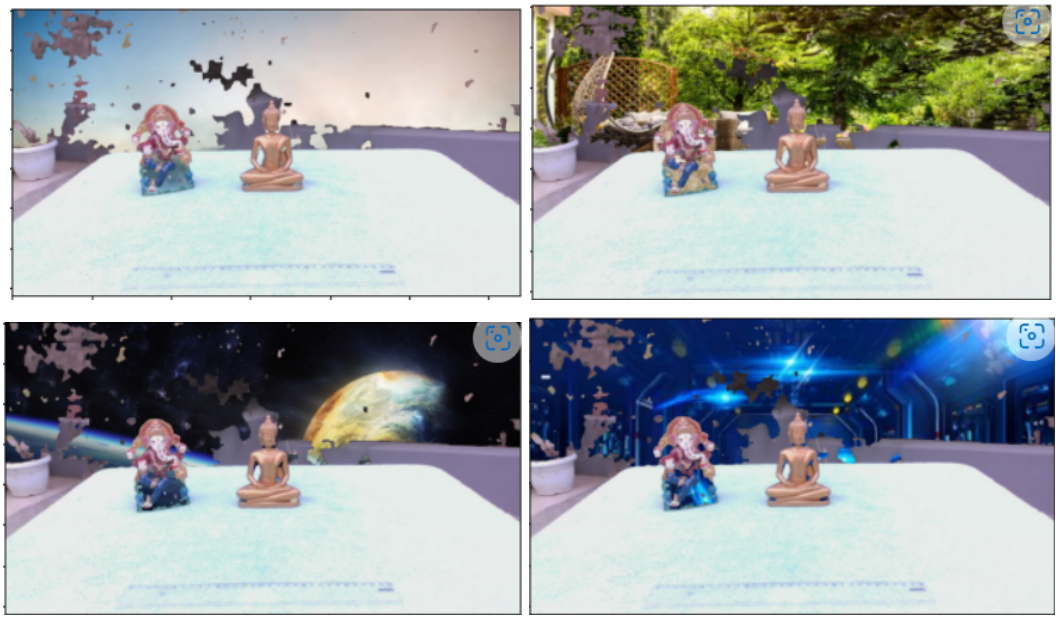
BooksA W Ours+Telea:



BooksA W Ours+NS:



The second part of this task consisted of experimenting with different visualisation methods to demonstrate the changing background effects. In order to achieve this we developed a new function called remove_and_change_background() which first performs background subtraction and afterwards, the function calls the NewBackground() function to change the image's background to the given background image. The following is an example of the result after applying four different background images to the StatuesB image with the "W" instance.



References :

[1] “OpenCV modules.” docs.opencv.org. https://docs.opencv.org/3.4/index.html (Accessed May. 12, 2022)

[2] "Blending images using OpenCV" Medium. 2021 [Online]. Available: https://medium.com/featurepreneur/blending-images-using-opencv-bfc9ab3697b7#:~:text=OpenCV-Python%20uses%20the%20addWeighted%20%28%29%20function%20to%20blend,and%20its%20parameters%20are%20as%20follows.%20dst%3Dcv.addWeighted%20%28src1%2C (Accessed: May 12, 2022)

[3] "OpenCV: Sobel Derivatives", Docs.opencv.org. [Online]. Available: https://docs.opencv.org/3.4/d2/d2c/tutorial_sobel_derivatives.html (Accessed May 12, 2022)

[4] "OpenCV Python Image Smoothing - Gaussian Blur", TutorialKart. Available: https://www.tutorialkart.com/opencv/python/opencv-python-gaussian-image-smoothing/#:~:text=OpenCV%20provides%20cv2.gaussianblur%20%28%29%20function%20to%20apply%20Gaussian,sigmaX%20%20%20dst%20%5B%2C%20sigmaY%20%5B%2C%20borderType%3DBORDER_DEFAULT%5D%5D%5D%29 (Accessed May. 12, 2022)

[5] "OpenCV - Box Filter", Tutorialspoint.com. Available: https://www.tutorialspoint.com/opencv/opencv_box_filter.htm (Accessed: May. 12, 2022)

[6] rishikagupta1. “Background subtraction – OpenCV.” geeksforgeeks.org. https://www.geeksforgeeks.org/background-subtraction-opencv/ (Accessed May. 24, 2022)

[7] J. Luke. “Remove green background screen from image using OpenCV/Python.” stackoverflow.com. https://stackoverflow.com/questions/51719472/remove-green-background-screen-from-image-using-opencv-python (Accessed May. 24, 2022)

[8] A. Rosebrock . “ Image inpainting with OpenCV and Python.“ pyimagesearch.com https://pyimagesearch.com/2020/05/18/image-inpainting-with-opencv-and-python/ (Accessed May. 28, 2022)

[9] P. Samaratunga. “Change the Background color of an image set using OpenCV.” stackoverflow.com. https://stackoverflow.com/questions/69060426/change-the-background-color-of-an-image-set-using-opencv (Accessed Jun. 1, 2022)

[10] ali_m “ Image SSD calculation error value” https://stackoverflow.com/questions/29642488/image-ssd-calculation-error-value (Accessed May. 28, 2022)

[11] A.Olafenwa “Change the Background of Any Image with 5 Lines of Code.“ towardsdatascience.com https://towardsdatascience.com/change-the-background-of-any-image-with-5-lines-of-code-23a0ef10ce9a (Accessed Jun. 1, 2022)