ARI 2129 – Principles of Computer Vision for AI

Daniel Grech [daniel.grech.19@um.edu.mt](mailto:daniel.grech.19@um.edu.mt) Owen Agius [owen.agius.19@um.edu.mt](mailto:owen.agius.19@um.edu.mt)

## Part 1 – Computer Vision Functions

The task at hand is separated into two different sections where the first part tackles image blending and the second part tackles a chroma key implementation. Image blending is achieved in four unique steps. Consider Scene 1 [*Figure 1*] to contain only one object in the image and Scene 2 [*Figure 2*] to contain two or more images in the images.

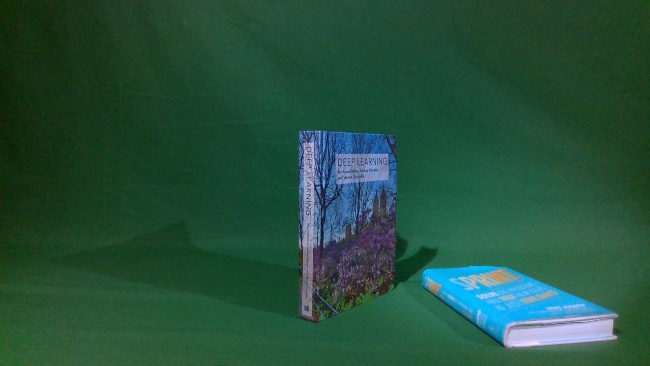


Figure – Scene 2

Figure – Scene 1

1. Extracting an object from Scene 2
2. Applying a filter to the object
3. Blending the object extracted to Scene 1
4. Comparing the blended image to Scene 2

Step 1:

The object is extracted from scene 2 with a simple implementation of a bitwise and operation on Scene 2 and a corresponding mask [*Figure 4*]. The illustration of the output of this stage can be seen in *Figure 3*.

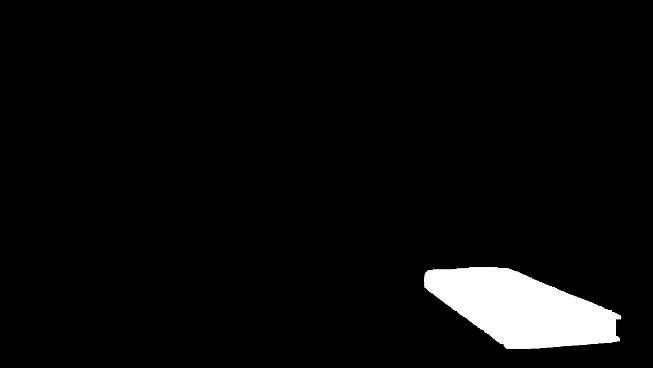


Figure – Output of Step 1

Figure – Mask

Step 2:

The three filters implemented were chosen to be the Bilinear, Gaussian and Sobel X filters. The Sobel X and the Gaussian filters are achieved via an implementation of convolution. Meanwhile, the Bilinear filter uses a hard coded array to achieve its results. The respective results can be found in the below images.



Figure – Image with Bilinear Filter

Figure – Image with Gaussian Filter

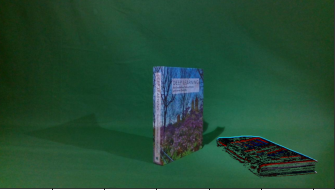


Figure – Blended Image with Sobel X Filter

Figure – Image with Sobel X Filter

Step 3:

Object blending is the focal point of this task where the aim is to add the extracted images found in *Figures 5-7* to Scene 1 [*Figure 1*]. The addition of the object to Scene 1 would result in an output image very similar to that of Scene 2 [*Figure 2*]. The object blending procedure is processed as illustrated in the flowchart below.

*Figure 8* represents the final and resultant image of all the processes defined above combined, producing a satisfying result of object blending. As it can be seen, the image in *Figure 8* is very similar to that of Scene 2 [*Figure 2*]. The only difference that can be spotted is the occlusion of the shadow of the extracted object. Although a short coming of the object detection algorithm, this was expected since the extraction procedure only captures the image rather than both the image and its shadow.

Step 4:

Finally, we can measure the error metric between the blended image [*Figure 8*] and Scene 2 [*Figure 2*]. Although to the naked eye one can spot the differences between the images, a more formal and mathematically proven approach has to be implemented. For the error metric the SSD and MSE error metrics were applied.

Sum of Square Differences is one of the measures of a match that is based on pixel-by-pixel intensity between the two images. [1][2] The summation of the square of the product of pixel subtraction is calculated between two images or rather defined by the equation below.

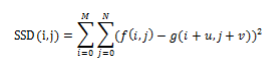
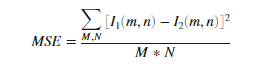


Figure – SSD Equation

Figure – MSE Equation

The result of SSD between *Figure 7* and *Figure 2* returns a value of 201. This value suggests a rather close matching between both images but as predicted earlier, the value is larger due to the occlusion of the shadow.

Comparative to the SSD, MSE defined as the Mean Square Error represents the cumulative squared error between the compressed and original image defined as the *Figure 8*. [3]

The value for MSE on the same images defined in the SSD implementation results to be 15.5145. The value is much more satisfying since the algorithm for MSE computes over the whole image at once rather than iterating through all the pixels as the SSD does. The below table denotes the values of the error metrics represented when blending the extracted image of a certain filter.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | Sobel X Filter | Gaussian | Bilinear | Normal Image |
| MSE | 20.078 | 18.346 | 16.113 | 15.515 |
| SSD | 184 | 208 | 176 | 201 |

Part B of Task 1:

Contrastingly to the above functions, the below takes a scene and converts the green background into a background of our choice. This is achieved via two functions who respectively remove the green colour from the image and replace the blank space with a predefined background.

Step 1: Remove Green:

This function sets two sets of 3 values representing BGR values. The two sets act as a threshold for a lower value and a higher value of green pixels. Any colour within the range of these two sets will be defined as an intense green colour and hence occluded from the image. As seen in both *Figures 11 -12*.

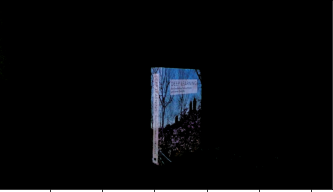
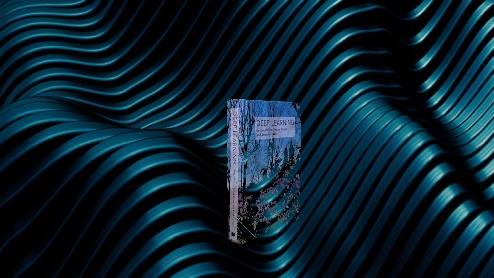


Figure – Chroma Keyed Image

Figure - Chroma Keyed Image

Step 2: New Background:

The new background is added via a simple addition blending technique as seen in the images below.



As can be seen on the images to the left, when attempting to add the new background to the extracted book, some features are left out from the book. This was expected since the remove green function removed the green colouring on the cover of the book. Although, this reciprocates when testing with a shoe. The shoe has no green elements, ensuring a loss-less green screen effect.

## Part 2 – Implementing a CV Research Paper

In this part, results from the research paper [4] are replicated using Python and OpenCV. Task A tackles replicating the results on the mentioned research paper (specifically Fig. 3 and Table 1). Task B is split into two parts where the first part simply involves applying the functions in Task A on images from the new addition to the COTS dataset (Complex Background) and the second part involves visualising images having a moving or unmoving background.

### Task A

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Scene** | **S2** | **Mask** | **S1** | **Telea** | **NS** |

In this section, results from the Inpainting research paper mentioned in the introduction are replicated. In order to make the functionality cleaner and easier to implement for Task B part 1, the required images for S1, S2 and masks of the object to remove are stored in the subfolders *original\_scene1*, *original\_scene2* and *original\_masks* respectively within *inpaint\_imgs*. The function create\_inpaint\_table then iterates through the images in these folders. In this function, the inpaint algorithms by Telea [5] (cv2.INPAINT\_TELEA) and Bertalmio *et al* [6] (cv2.INPAINT\_NS) where applied to each image in S2. The images within this function are then plotted as with the research paper using the *matplotlib.pyplot* python module. The function returns two lists containing inpainted images using each algorithm as well as another list with the ground truth (S1) images. These are used to calculate the MSE.

|  |
| --- |
| Statues |
| Shooter Glasses |
| Academic Books |
| Footwear |
| Mugs |
| Tech |

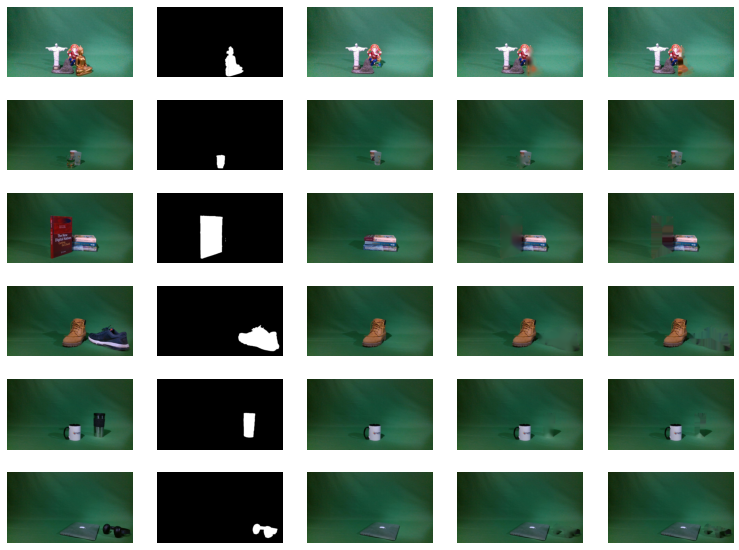


Figure - A visual representation of Inpainting results where S2 is the original scene and S1 is the ground truth which the inpainted images will be compared to in the evaluation.

To obtain the results in Figure 13, the inpaint\_radius in OpenCV’s inpaint function was set to 50 and the image S2 and mask of the object to remove where passed. The visual results show that the Telea’s approach yields a blurred output when compared to the approach by Bertalmio *et al.* which gives sharper results. On a subjective level, it seems that cv2.INPAINT\_TELEA performs better in the images where there is no occlusion since more of the green background replaces the inpainted object and the artifacts are less present. On the other hand, with occluded objects, it creates less realistic outputs because of the blurring as opposed to cv2.INPAINT\_NS which gives crisp results and though they are not completely correct, give a better sense of continuity.

The Mean Squared Error was then calculated when comparing the inpainted images with the ground truth (S1). The function create\_mse\_table takes the lists of images returned by create\_inpaint\_table and uses the function CompareResult from Part 1 to calculate the MSE for every inpainted image. Below are the results.

|  |  |  |  |
| --- | --- | --- | --- |
|  |  | *Mean Squared Error (MSE)* | |
|  | **Occlusion** | **Telea** | **NS** |
| **Statues** | yes | 131.98347005208333 | 170.900166015625 |
| **Shooters** | yes | 19.108080873842592 | 19.785056785300927 |
| **Academic** | yes | 94.62069878472222 | 124.6414529079861 |
| **Footwear** | no | 20.95038013599537 | 29.786233000578704 |
| **Mugs** | no | 23.329866898148147 | 28.146632306134258 |
| **Tech** | no | 36.016528862847224 | 48.04720341435185 |

Even though the results are quite different from the original research paper, they are in the same ratio relative to each other and hence paint the same picture as those in the research paper. The highest MSE scores were obtained in the Statues and Academic images which both contained occlusions. This is expected since the inpainting algorithms are less effective to model the complex objects behind the inpainted object as opposed to a uniform green background. That being said, the Shooters had the lowest MSE, despite having occlusions, but this is probably because the inpainted object was small and transparent so there was not much difference between the inpainted images and the ground truth. In all cases, *NS* had a higher MSE than *Telea* meaning that based solely on the MSE, *Telea* performs better, at least with images from the COTS dataset.

*Table 1 – MSE for images inpainted using cv2.INPAINT\_TELEA and cv2.INPAINT\_TELEA using images from S1 as the ground truth.*

### Task B

The first part in task B involved reusing the code from task A but with images having complex background. The required images for S1, S2 and masks of the object to remove are stored in the subfolders *new\_scene1*, *new\_scene2* and *new\_masks* respectively within *inpaint\_imgs*. The relevant masks were extracted from the masks in the dataset by using the cv2.inRange function to convert each mask image to a binary image containing only the green masks which corresponded to the object to be inpainted.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Scene** | **S2** | **Mask** | **S1** | **Telea** | **NS** |

|  |
| --- |
| Bottles |
| Electronics |
| Cups (no) |
| Statues |
| Cups (oc) |
| Books |

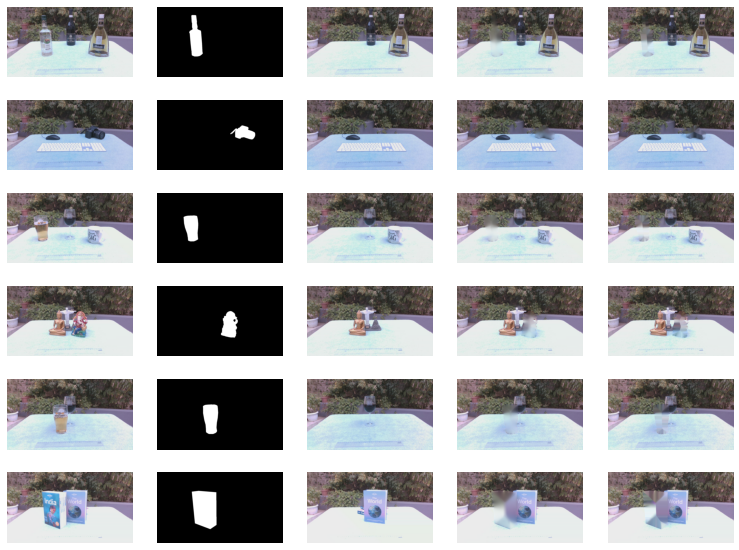


Figure - A visual representation of ipainting results for images with complex backgrounds.

The inpainting algorithms seem to produce less accurate results with the backgrounds which do not have a uniform surface.

|  |  |  |  |
| --- | --- | --- | --- |
|  |  | *Mean Squared Error (MSE)* | |
|  | **Tag** | **Telea** | **NS** |
| **Bottles** | No wind, no occlusion | 73.66781358506944 | 84.89556676793981 |
| **Electronics** | No wind, no occlusion | 84.67842918113426 | 145.5227560763889 |
| **Cups** | Wind, no occlusion | 117.59107277199074 | 123.95586588541667 |
| **Statues** | Wind, no occlusion | 147.5934790943287 | 156.44811234085648 |
| **Cups** | Wind, occlusion | 108.88221390335649 | 116.92723524305555 |
| **Books** | No wind, occlusion | 258.9800372540509 | 343.8428927951389 |

*Table 2 – MSE for inpainted images with complex backgrounds.*

As can be noted from the above MSE results, Telea performed better than NS just as it did with the images from the original dataset. In general, the Mean Squared Errors are higher, probably due to the complex background which makes it more difficult for the inpainting algorithms to fill in the missing area than with a uniform surface background. Images that had occlusion also had higher MSE values. The wind seemed to have an effect on certain results as was expected due to the moving background. Images tagged “Wind, no occlusion” had overall higher MSEs than images tagged “No wind, no occlusion”.

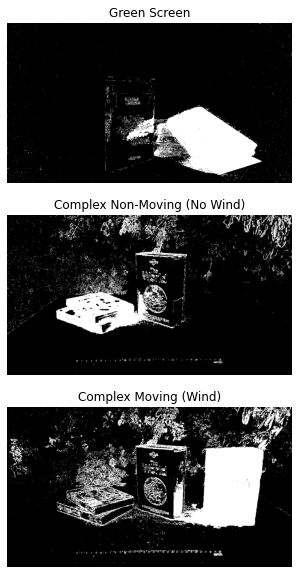
In the second part of Task B, two techniques were used to visually compare the backgrounds of images. The first technique involves using the MOG2 background subtractor which is based on [7] and [8]. Three images from a particular scene are fed into the background subtractor object and this allows it to learn and approximate what the background in the image is. The output is a mask where the foreground objects are white and the background is black, hence it is removed. The second technique uses the ImageChops.difference function from the *PIL* library. This function finds the differences within two images and removes similar elements in the output. Therefore, two images from the same scene were inputted to the function and if the background shows in the output, then it has changed between the images. Below is the result of these experiments.

Figure - Background subtraction with MOG2

Figure - Finding the differences between images using ImageChops.diff() in PIL library

In the above output, newly added objects to the scene have brighter pixels. In both Figures 15 and 16, the green screen background has been completely removed meaning that there was virtually no change in background between images. This shows that images from this dataset were taken in a controlled environment where the lighting was constant and the environment was unchanging. In the second case, the complex was not really removed with the MOG2 background subtractor indicating that the background subtractor either had problems removing it because of its complexity, or there was some slight change in it because the pictures were taken outside with little control over the environment. Background changes were barely detected with the second technique meaning that there were some slight changes in the background between scenes but they were very small. In the final example, were there was wind, the background showed up very clearly in the MOG2 output, showing that the background was not removed. The background can also be seen with the second technique showing that the scene background changed quite noticeably between images.

## References

[1] M. B. Hisham, S. N. Yaakob, R. A. A. Raof, A. B. A. Nazren and N. M. Wafi, "Template Matching using Sum of Squared Difference and Normalized Cross Correlation," 2015 IEEE Student Conference on Research and Development (SCOReD), 2015, pp. 100-104, doi: 10.1109/SCORED.2015.7449303.

[2] S. Ourselin, X. Pennec, R. Stefanescu, X. Pennec, and R. Stefanescu, “Robust Registration of Multi-Modal Medical Images : Towards Real-Time Clinical Applications, 2001.

[3] <https://www.mathworks.com/help/vision/ref/psnr.html#:~:text=The%20mean-square%20error%20(MSE,MSE%2C%20the%20lower%20the%20error>. [Accessed: 23/06/21]

[4] Seychell, Dylan & Debono, Carl. (2020). An Approach for Objective Quality Assessment of Image Inpainting Results. 226-231. 10.1109/MELECON48756.2020.9140597.

[5] A. Telea, “An image inpainting technique based on the fast-marching method.,” Journal of Graphics, GPU and Game Tools, vol. 9, no. 1, pp. 23–34, 2004.

[6] M. Bertalmio, A. L. Bertozzi, and G. Sapiro, “Navier-stokes, fluid dynamics, and image and video inpainting,” in Proc. of the 2001 IEEE Computer Vision and Pattern Recognition (CVPR), pp. 355–362, 2001.

[7] Z. Zivkovic, "Improved adaptive Gaussian mixture model for background subtraction," Proceedings of the 17th International Conference on Pattern Recognition, 2004. ICPR 2004., 2004, pp. 28-31 Vol.2, doi: 10.1109/ICPR.2004.1333992.

[8] Z. Zivkovic and F. van der Heijden, "Efficient adaptive density estimation per image pixel for the task of background subtraction", Pattern Recognition Letters, vol. 27, no. 7, pp. 773-780, 2006. Available: 10.1016/j.patrec.2005.11.005.