Computer Vision 600100

Counting Starfish

Student ID: 201601628

Date: **May 19, 2020**

Deadline: Tuesday 5th May 2020 by 2pm (+14d)

Image Processing Pipeline

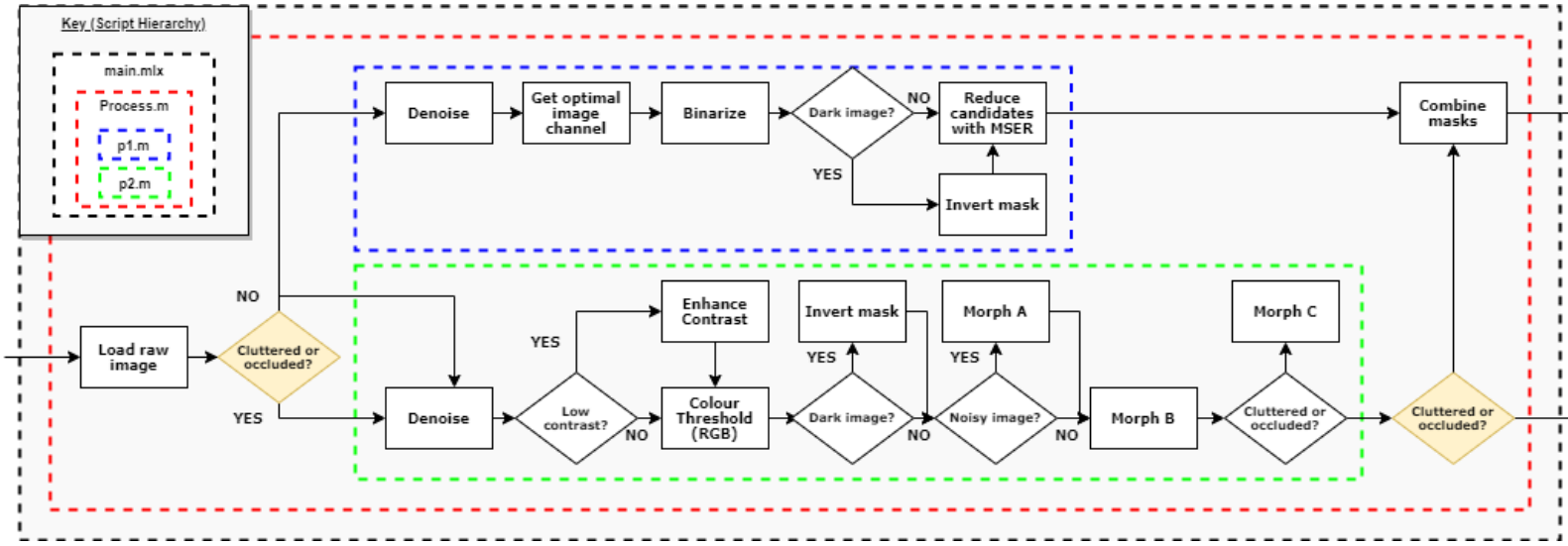


Figure 1: High-level overview of the implemented pipeline architecture (full-size version included in code submission).

Introduction

The proposed pipeline is a non-linear composition of p1.m and p2.m. Each is a pipeline within itself. P1 is focused on the isolation of blobs without too much concern for segmentation *quality**, whereas P2 is focused on the quality of masks - as long as all starfish are detected. In other terms, P1 is geared towards isolating distinct/separate (i.e. non-occluded) objects in an image, and P2 is better generalised at the cost of many false positives. By cross referencing (combining) the more accurate detections of P1 with those of P2 in a 'super' pipeline, the proposed architecture aims to produce higher quality masks than either pipeline can independently create.

** quality in this sense is considered as the similarity of the mask shape to the original object.*

Exploratory Work

The design is influenced by extensive exploratory and analytical work, which helped to develop an understanding of the provided data and potentially applicable (and inapplicable) techniques. This, for example, involved the use of MATLAB's colour thresholding app, studying the histograms of images, and research into localised denoising methods. Some examples of this exploratory work are included in the appendix, and associated scripts are included in the project folder, "*. /appendices*"

Most notably, this exploratory work is what revealed the possibility of creating a relatively successful pipeline (for the given starfish image variations) by simply taking a single image channel and binarizing it (*see "./appendices/EzMode.mlx")*. **This is the basis for p1.m and what inspired the more complex architecture as to explore more than just the most basic techniques for this project.**

Other work included alternative methods for noise detection and reduction. The concept of removing salt and pepper noise by creating a mask of all min and max pixels (i.e. pixel intensity of 0 or 255) showed impressive results for restoring images, although more destructive smoothing techniques proved more useful in segmentation (*see "./appendices/DenoiseS&P.mlx")*. Similar success was seen with a more experimental

approach, using Laplacian edge detection and convolution to create the 'diff mask' (*see "./appendices/NoiseDetectLaplacian.mlx").* Variations of these experiments informed the *myDenoise.m* function in the proposed pipeline.

Proposed Pipeline Stages

1. Detect cluttered/occluded images
    I. This check determines whether to use the P1 route in the pipeline since images with occlusion or clutter (i.e. the 'advanced' images) do not work well with P1. A custom method (*./functions/IsCluttered.m*) was created to this end, which checks the percentage of black and white pixels in an Otsu's threshold mask of the original image. The classification is then just a matter of thresholding the percentage against a value observed to identify all the 'cluttered' images in the given data (0.37).
2. P1 (if the image is not cluttered/occluded)
    I. **Denoising**: every image is median filtered using *medfilt2*, per channel, with a 3x3 kernel, followed by a mean filter in the same way (with *imfilter* and padding equal to 'same' as to not reduce image size). Although this less tailored approach to denoising the images results in perceptively less detailed images, it was found to improve segmentation in all cases. As informed by exploratory work, a third denoising step conditionally and locally enhances the image further, using a threshold of >30 to identify noise from the 'diff mask'.
    II. **Optimal channel selection**: as discussed in the Exploratory Work section, it was found that the 'optimal' channel for use in the creation of an initial binary mask could be found by taking the channel with the max variance (when using the RGB colour model). This was implemented as a custom function (*./functions/GetOptimalChannel.m*).
    III. **Binarization**: because the selected channel is so 'optimal' (has high contrast between objects and background), the binarization step simply uses the default Otsu's method of *imbinarize*, as implemented by MATLAB.
    IV. **MSER**: The binarization step successfully isolates objects – but more than just starfish. The MSER phase uses thresholds for extent, solidity, eccentricity (values determined by observing table of outputs) to isolate starfish in the majority of the given images.
3. P2
    I. **Denoising**: *same implementation as in P1.*
    II. **Contrast enhancement**: with liberal smoothing applied in the previous step, it was found that the best way to increase edge contrast in low-contrast images (rather than sharpening) was to enhance the contrast. For this, *imadjust*, *histeq*, and *adapthisteq* were explored, with the best segmentations resulting from the latter, which locally increases contrast rather than globally.
    III. **Colour thresholding**: using the MATLAB colour thresholder, the RGB colour model was found to be the best all-rounder for segmenting starfish in the given images and proposed pipeline. The thresholding is applied dynamically, based on the distribution of colour channels. Conditional behaviours (and values) were informed by manual analysis of the differences between the distributions of colour channels in images and are otherwise random/as determined by trial and error to achieve the best segmentations.

IV. **Morphological operations**: the morphological phase of P2 has 3 distinctions. Very noisy images are identified and operations selected to reduce noise and less aggressively increase mask sizes (as to not reconnect with noisy blobs), Less noisy images are processed more aggressively, with the a lower risk in that respect; aiming to reconnect fragments of the real object. If an object is deemed to be cluttered or occluded, (*IsCluttered.m*), then additional, increasingly aggressive morphological operations are performed in order to separate objects. The noise level of images is determined by a custom method (*./functions/GetNoiseLevel.m),* which calculates a relative but representative value for the severity of noise in an image, using methods similar to those in *myDenoise.m.* Finally, detections caused by borders etc. are removed by removing detections with bounding areas close to the size of the image area.

4. Composition of P1 & P2
   I. This process essentially checks if blobs from P2 collide with blobs from P1, and then keeps the larger of the 2. If the area of blob in P2 is more than 2.5x as large as in P1, it is assumed to be a conjoined blob in P2 and the P1 detection is used. Finally, if P1 did not detect any blobs, the P2 mask is used. This is justified since, P1 is not as well generalized as P2.

Testing

In addition to formative, exploratory work – some lightweight testing of implemented methods is included in the project, in the folder, "*. /testing*". The goal of this is to validate, demonstrate and justify the techniques described above as well as those ultimately not used in the pipeline (e.g. *./functions/GetNoiseType.m*). Some examples are included in Appendix 2.

Results

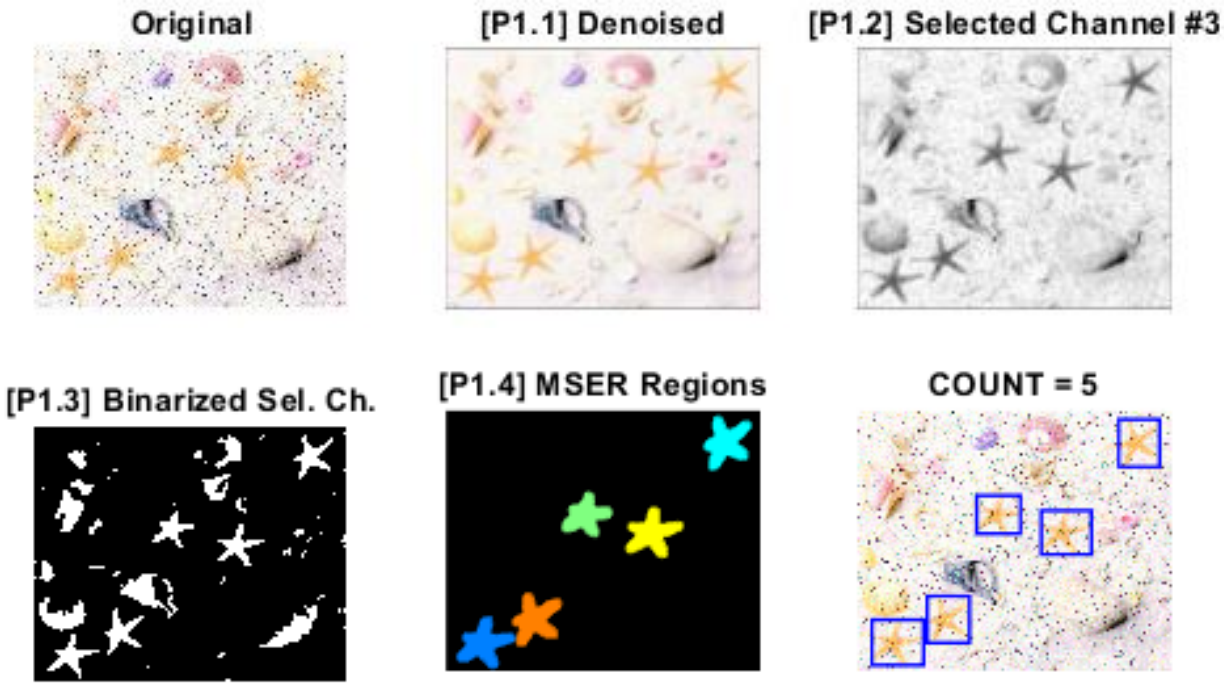


Figure 2: p1.m Processing steps with starfish.jpg.

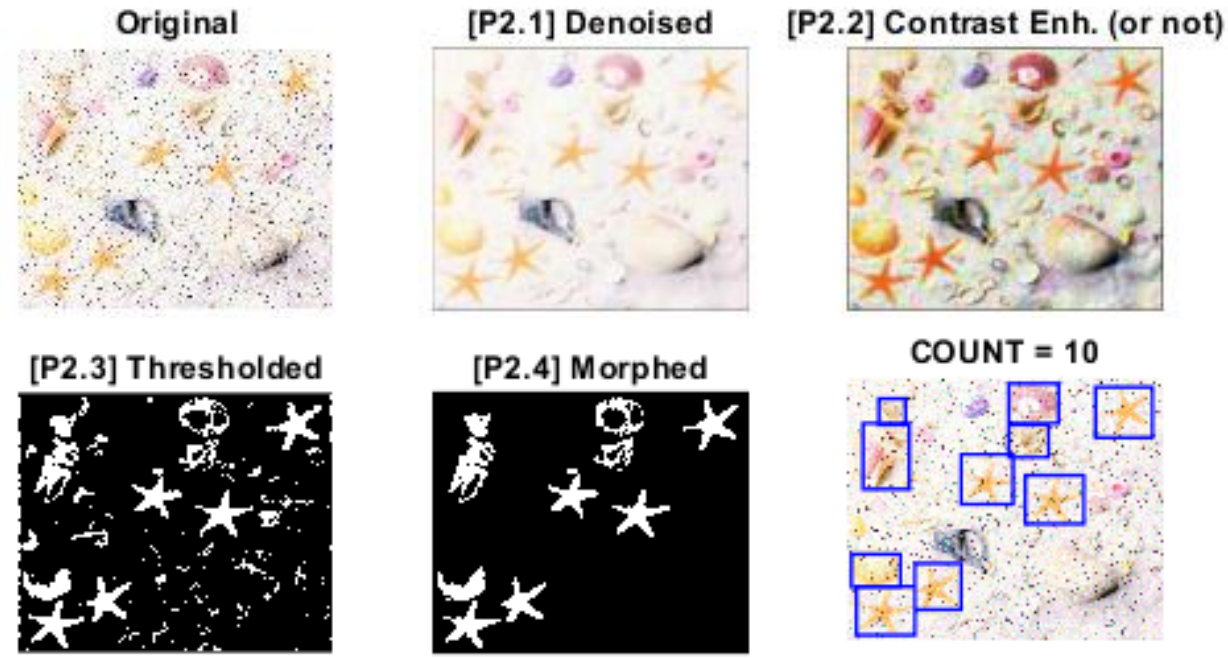


Figure 3: p2.m Processing steps with starfish.jpg.

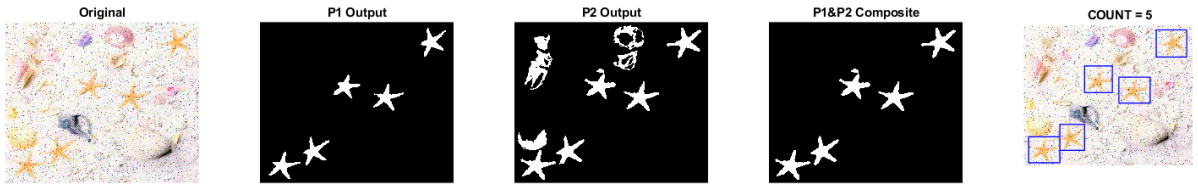


Figure 4: Final output of the proposed pipeline, with starfish.jpg.

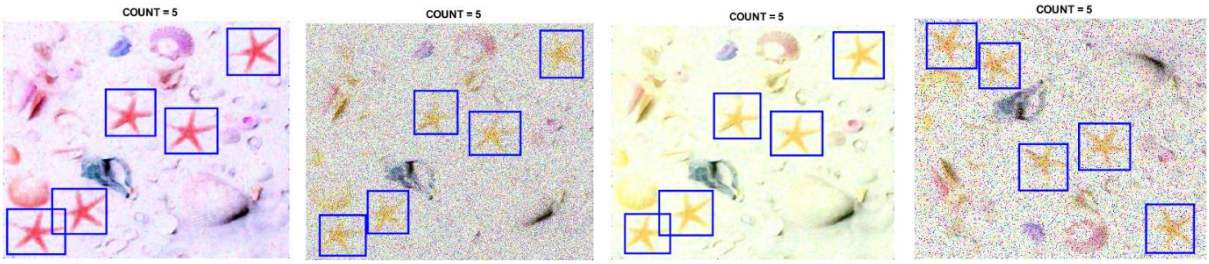


Figure 5: Final outputs for _map1.jpg, _noise9.jpg, _map2.jpg and _noise5.jpg.

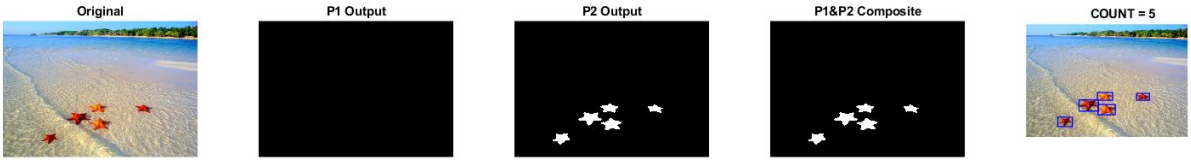


Figure 6: Composition fallback mechanism in action, with starfish_5.jpg (final output).

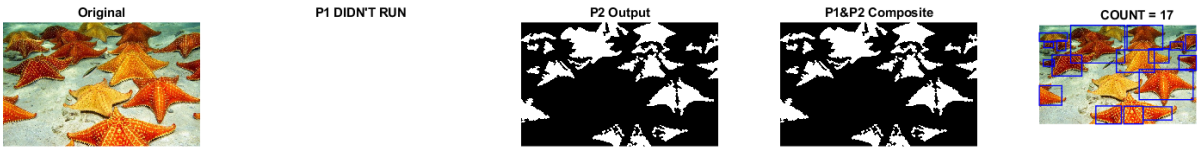


Figure 7: Non-linear approach in action; skipping p1 because starfish_17.jpg is cluttered/occluded (final output).

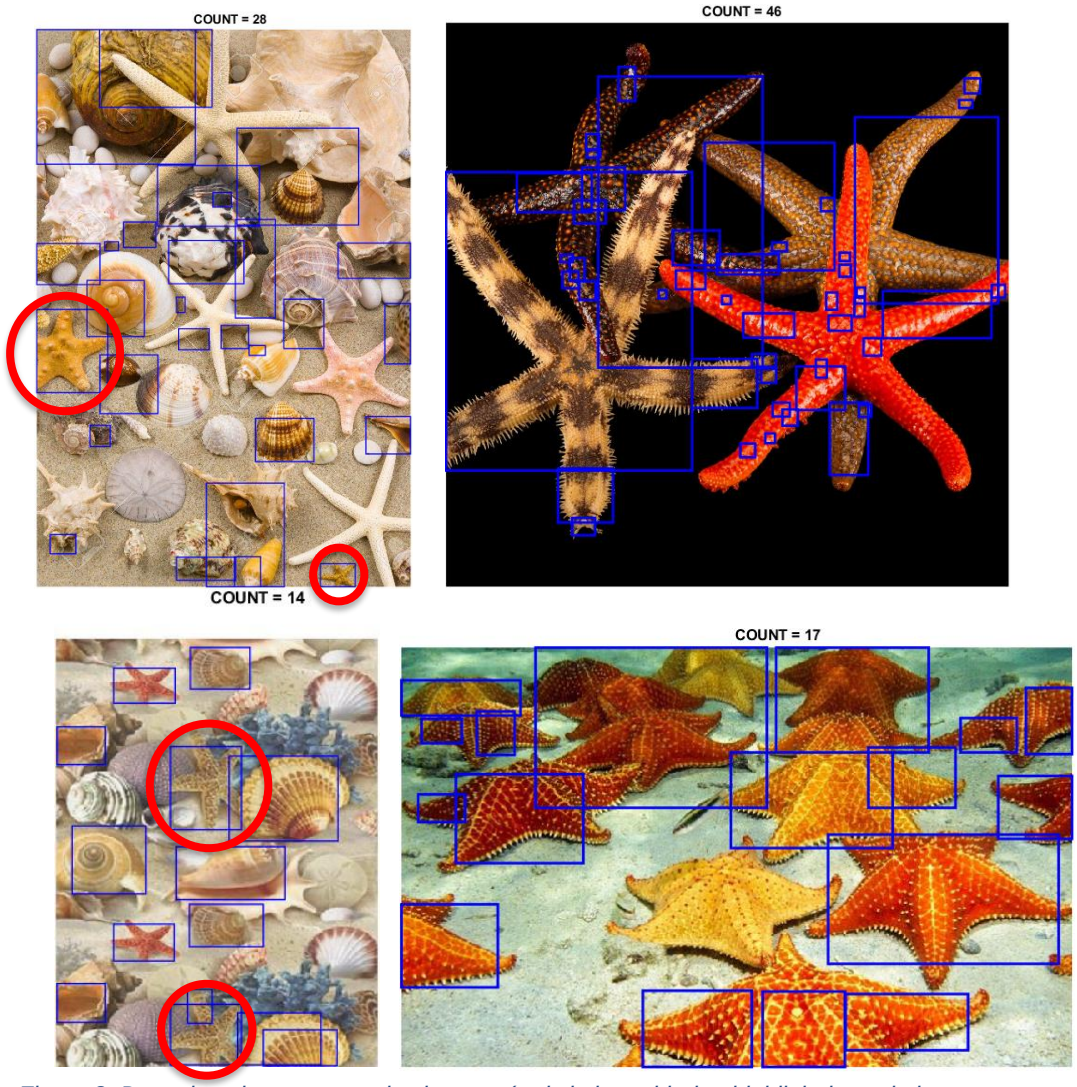


Figure 8: Detections in more complex images (red circles added to highlight less obvious successes).

Discussion

The pipeline processes outlined in figures 2, 3 and 4 are representative of the steps to achieve all other illustrated outputs. Observing figures 4, 5 and 6, the pipeline appears to excel situations where starfish are relatively small to the image size and texture is minimal. Additionally, reviewing figure 8 supports this characteristic and suggests that the pipelines biggest weaknesses are textures, and occlusion.

In respect to textures, this is likely a result of the primary methods of generating the masks in P2, which is used in the advanced images: colour thresholding. Looking at the 2 right-hand images of figure 8 shows many instances of semi-erroneous detections – i.e. detections that *are* starfish but only partial. In terms of occlusion, the pipeline does surprisingly well, considering the relatively primitive methods in place. Looking specifically at the starfish_overlaid.jpg image in figure 8, the exposed limbs of the top right starfish are all detected. The question this raises is whether this would be considered a ‘correct’ segmentation in the problem domain, or if there would be a need ‘patch-up’ the separate

components. Once more, looking at starfish_17.jpg in figure 8, where several starfish are recognised as one object; there is arguable some human difficulty in visually separating the objects.

Perhaps the worst segmentations of all the images are those in the left-hand side of figure 8 - shellfish_four.jpg and seashell_five.jpg – which include a wider variety of objects in a cluttered/occluded scene. The erroneous detections in these images, however, are again very texture-rich objects. It therefore stands to reason that integration of texture thresholding or detection methods could significantly improve the detection accuracy of more complex images like these.

Texture measures were implemented and extracted both manually and automatically (based on correct segmentations) but analysis of this data did not tell of any significant differences that might be used in the pipeline (see Appendix 1). However, this is possible due to the inclusion of the excessively noisy starfish.jpg variations. More meaningful data may have been acquired if only the non-noise, complex images were used.

In summary, considering that the domain is not something life threatening or mission critical, the proposed pipeline may be sufficient. Alternatives like CNNs might be overkill for such an application, but could certainly improve detection rates and probably generalise much better. Better generalisation of the proposed pipeline might be achieved by exploring additional texture measures or further de linearity of the pipeline.

Appendices

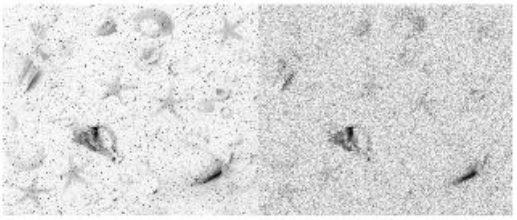
Appendix 1: Various Texture Statistics

SET		AvgGrayLvl	AvgContrst	Smoothness	ThrdMoment	Uniformity	Entropy
A	Correct/autodetected regions (wholestarfish)						
	nsamples	101					
	mean	220.3125233	36.57759306	0.020793641	-1.34724701	0.025735327	6.319948841
	std	9.606263592	6.844377469	0.00712311	0.643432944	0.010486921	0.410058349
	min	199.4109402	19.55017556	0.005843536	-3.088208534	0.009367171	5.366895744
	max	238.6797065	51.95383734	0.039855785	-0.309577353	0.061823616	7.031396357
B	SET#A + manual advanced image regions						
	nsamples	149					
	mean	204.8875525	40.44167429	0.026651996	-1.023450421	0.029218358	6.407772073
	std	41.93459516	13.07617967	0.018410123	2.067321099	0.033911886	0.624220136
	min	40.93327851	19.55017556	0.005843536	-4.584418833	0.004396147	4.843240556
	max	238.6797065	96.67969707	0.125678594	12.0185868	0.243327181	7.908471886
D	Correct and partial regions in advanced images only						
	nsamples	60					
	mean	123.2488932	62.64670602	0.059699598	0.661346394	0.023291403	7.270085107
	std	39.75647034	16.50205465	0.028526061	3.43704143	0.052357762	0.685245844
	min	41.15073382	27.76874119	0.011719584	-4.658081552	0.004345557	4.865342813
	max	220.6436911	104.4482389	0.143669136	10.21866321	0.240968465	7.917558147
D	Incorrect manual regions						
	nsamples	61					
	mean	168.1650861	46.84149477	0.036333422	-0.668484964	0.014834773	6.988777483
	std	49.34500329	17.7342365	0.026323324	1.40445567	0.015566826	0.66532977
	min	32.12442297	9.795517095	0.001473445	-3.901403085	0.00426849	5.080255821
	max	238.5833333	98.52904527	0.129902136	4.008444793	0.111302331	7.936933669

Appendix 2: Custom Methods as Demonstrated in Tests.mlx

TEST #2 STARTING : GetNoiseLevel() is representative

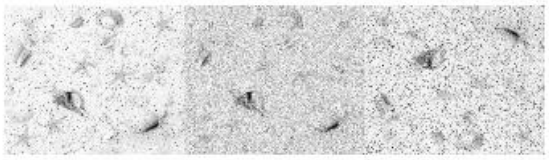
im1NoiseLevel = 0.2171
im2NoiseLevel = 0.7563



TEST #2 PASSED

TEST #3 STARTING : GetNoiseType() is representative

im1 = "Impulse/Isolated"
im2 = "Gaussian/No Noise"
im3 = "Speckle/Mixed"



TEST #3 PASSED

TEST #4 STARTING : IsDark() is representative

im1IsDark = *logical*
0
im2IsDark = *logical*
1

TEST #4 PASSED

TEST #5 STARTING : IsCluttered() is representative

im1IsCluttered = *logical*
0
im2IsCluttered = *logical*
1

TEST #5 PASSED
