# 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
   1. 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)
   1. **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’.
   2. **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*).
   3. **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.
   4. **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
   1. **Denoising**: *same implementation as in P1.*
   2. **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.
   3. **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.
   4. **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

**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*).

# Results

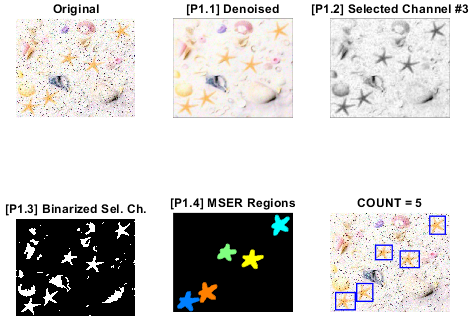
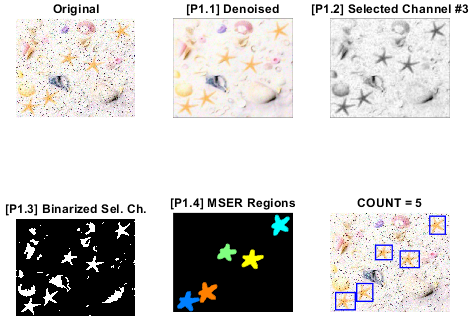


Figure : p1.m Processing steps with starfish.jpg.

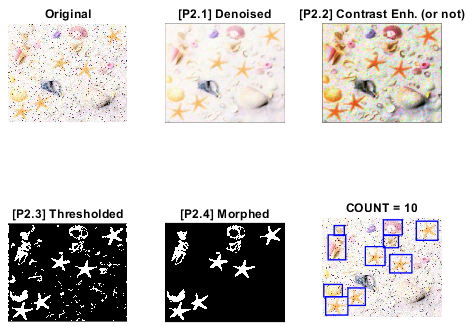
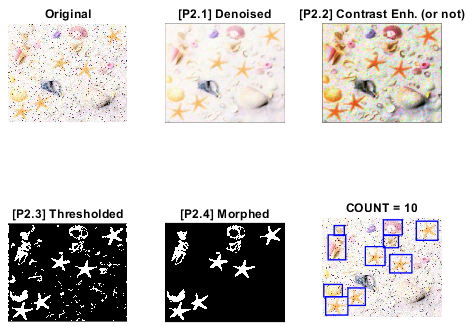


Figure : p2.m Processing steps with starfish.jpg.

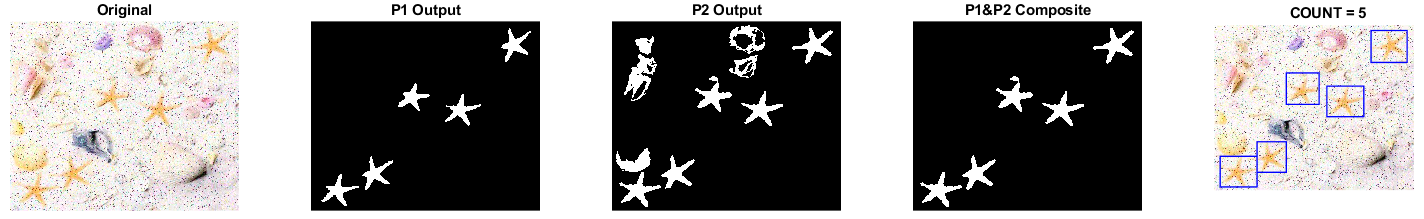


Figure : Final output of the proposed pipeline, with starfish.jpg.

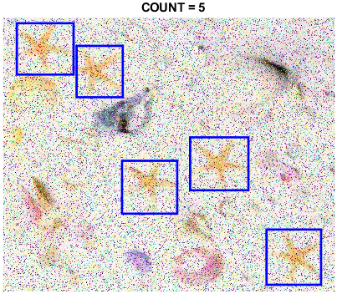
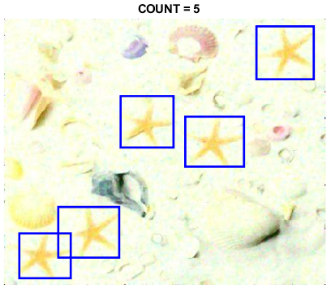
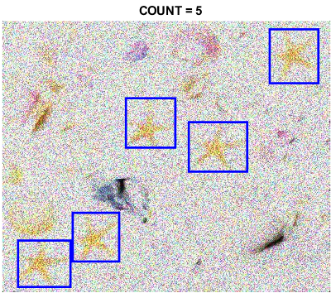
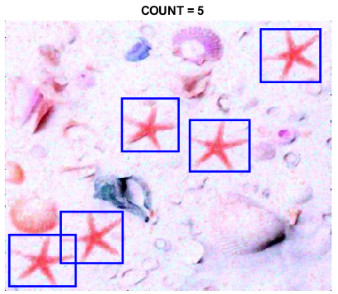


Figure : Final outputs for \_map1.jpg, \_noise9.jpg, \_map2.jpg and \_noise5.jpg.

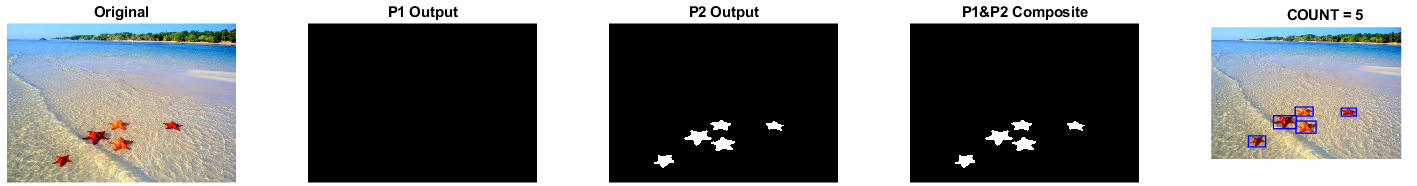


Figure : Composition fallback mechanism in action, with starfish\_5.jpg (final output).

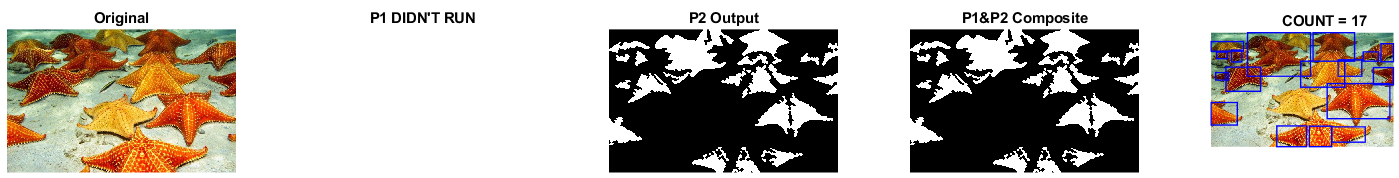
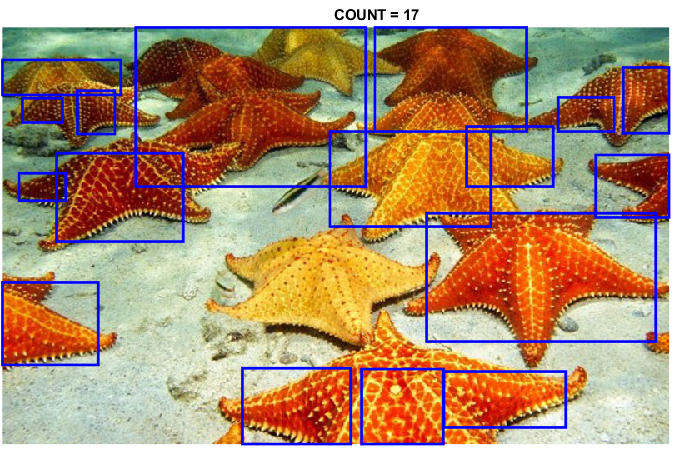
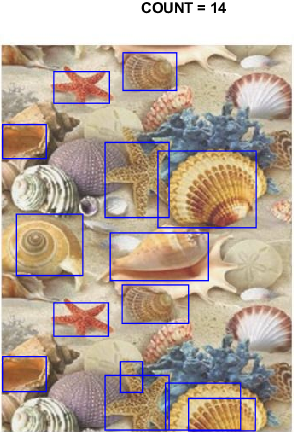
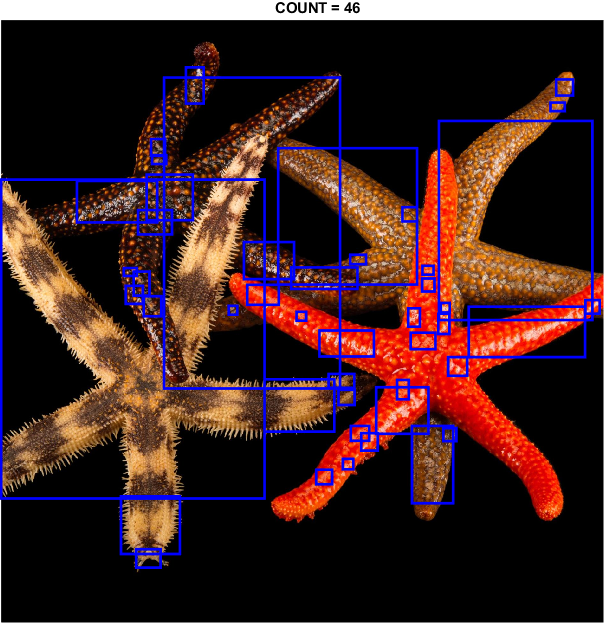
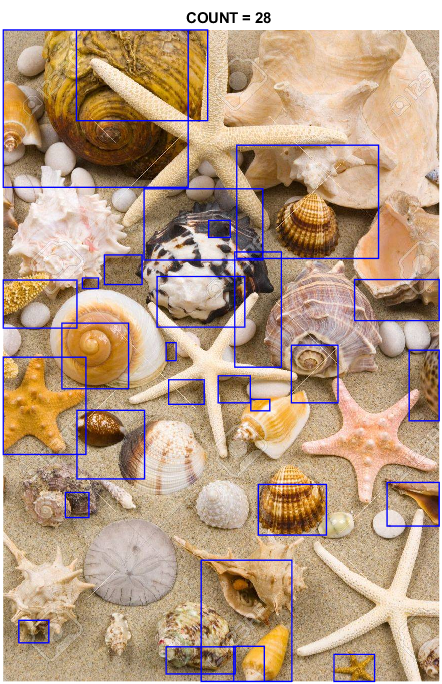


Figure : Non-linear approach in action; skipping p1 because starfish\_17.jpg is cluttered/occluded (final output).

Figure : The detections on more complex images (red circles added to highlight less obvious successes).



# Discussion

Discuss the results presented in the previous section. What works, and what doesn’t work; including why it may or may not work.

Considering the domain, P1 may be a sufficient pipeline in itself.

P2 has the benefit of actually denoising the original image, should that be useful. It is also better suited to more complex image where the objects to detect aren’t as well isolated.

The combined pipeline aims to find some middle ground between the simplicity of P1 and the additional utility and generalisation of P2.

P1s largest limitations are the

How can the design of your image processing pipeline and code be improved? Are there any alternative functions / algorithms / approaches which may have been more suitable in hindsight. Is there evidence to support this?

CNN, TEXTURE (CORREMAT), EDGE DETECTION (arguably not with noise ims), SURF/FAST?\*. Possibly a similar composite pipeline to detect more advanced images, e.g. detect NOT starfish and combine masks, or combine different colour models etc.

Remove contiguous space (borders) around images

Consider each stage of the image processing pipeline. Consider variations in noise, colourmaps, and image types, including the more challenging images with occlusion and clutter.

# Appendices

