# 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 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 and 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 overall accuracy. 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 offer.

*\* mask quality is referred to as the level of similarity to the original/enhanced 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”)*. The essence of these methods is to create a filtered version of the input image and use the most significant differences to create binary mask. This mask is used as an index to replace those pixels in the original with the value of those in the filtered image. Similar success was seen with a more unconventional approach utilising 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**

Give details on each stage of your proposed pipeline.

Explain what is the purpose of each stage in the image processing pipeline.

Explain what algorithm / function is used in each step. Why was it used? What parameters were used and why?

Steps

P1

-slice select

-binarize

-mser

P2

-denoising

-thresholding

-contrast

-dark

-morphing

P3

Composition

Give details on each stage of your proposed pipeline.

Explain what is the purpose of each stage in the image processing pipeline.

Explain what algorithm / function is used in each step. Why was it used? What parameters were used and why?

**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 employed (e.g. for classifying noise types). A script for processing every given image was also created to aid of these validations.

# Results

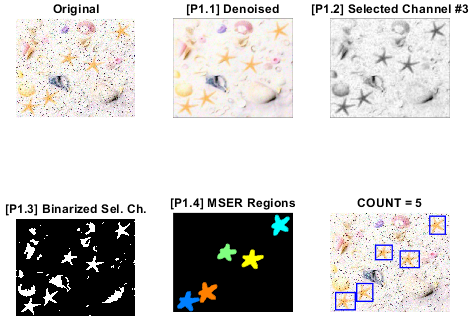
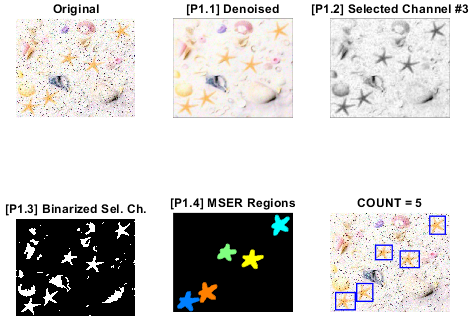


Figure : p1.m Processing steps with starfish.jpg.

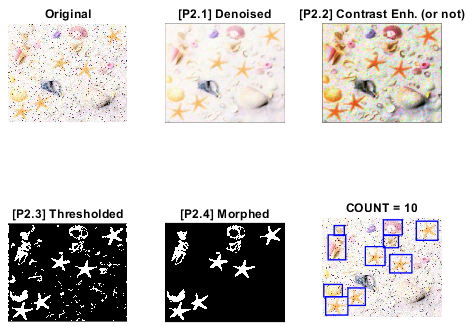
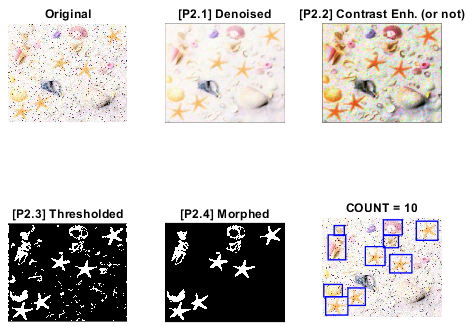


Figure : p2.m Processing steps with starfish.jpg.

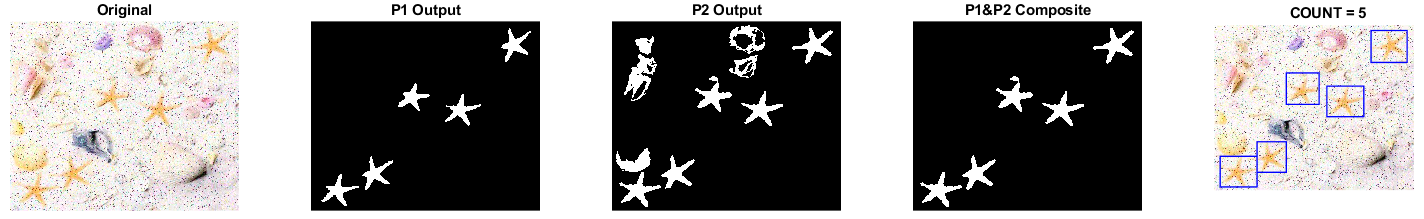


Figure : Final output of the proposed pipeline, with starfish.jpg.

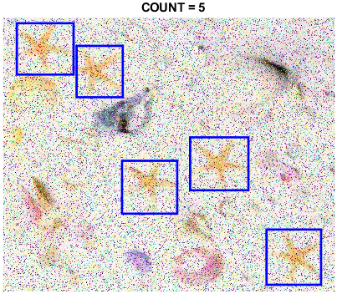
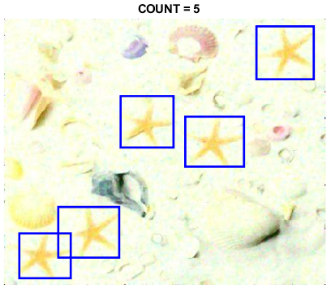
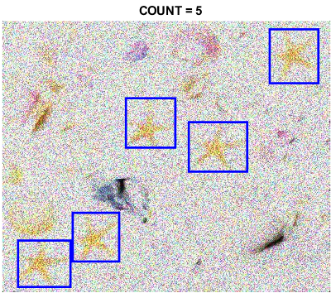
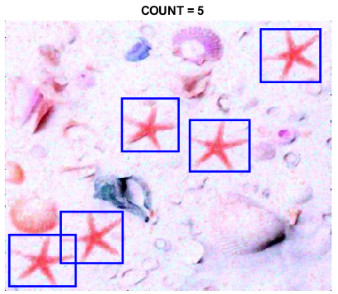


Figure : Final output for \_map1.jpg, \_noise9.jpg, \_map2.jpg and \_noise5.jpg.

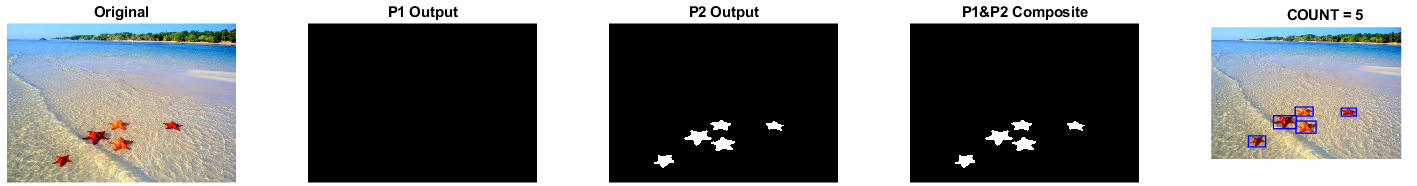


Figure : Composition fallback mechanism in action, with starfish\_5.jpg.

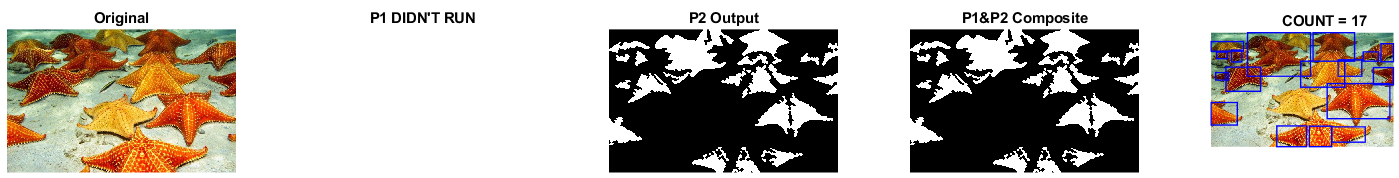
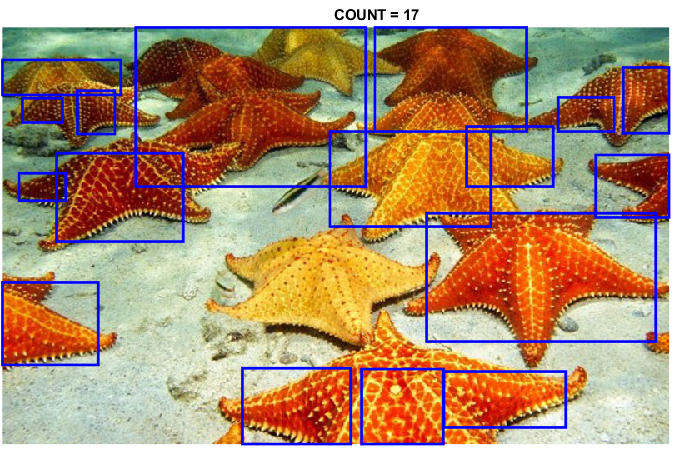
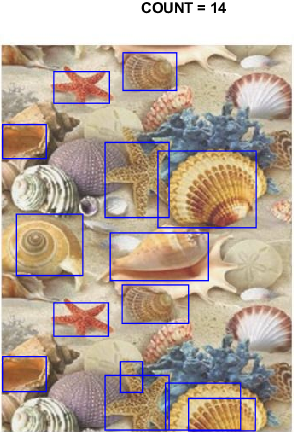
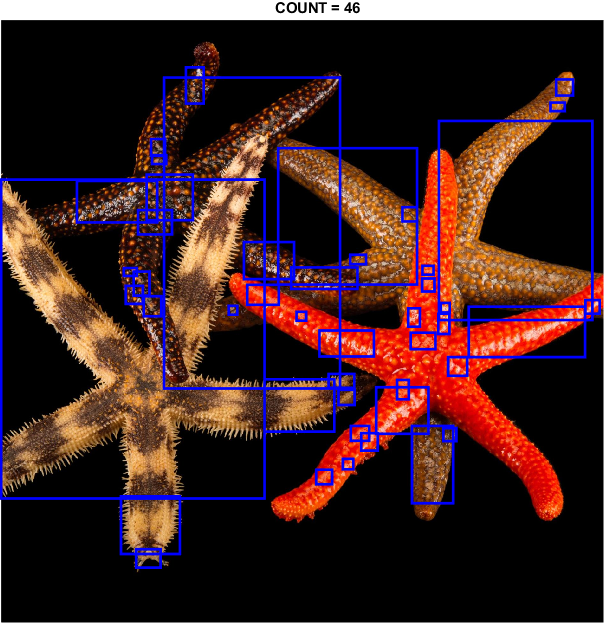
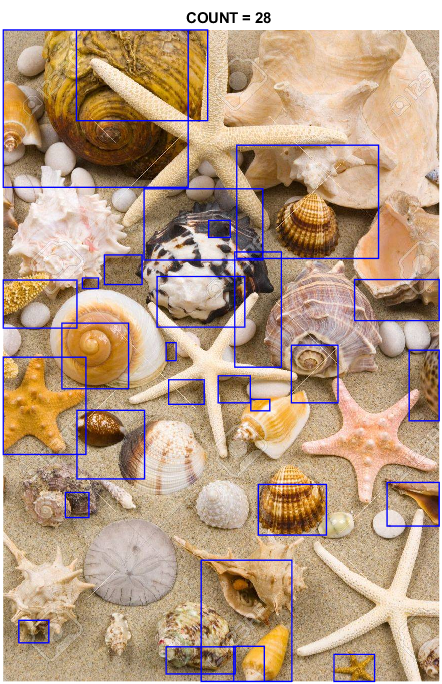


Figure : Non-linear approach in action; skipping p1 because starfish\_17.jpg is cluttered/occluded.

Figure : The detections on more complex images.



ALL FINAL OUTPUTS (MASK AND ROIS) – maybe go in the appendix

# 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

* colour thresholding
* all channel hists
* Noise detection & denoising with edge filters
* Textures

…Useless

* + Acquisition
  + Figures

Test Outputs and Additional Images

