INM460 / IN3060 Computer Vision

In this lab, we will use Matlab to perform image segmentation. As discussed in lecture, image segmentation partitions an image into multiple regions.

Resources

The lab will make use of Matlab, and the following data files on the Lab materials 04 folder on Moodle in the Session 4 area:

- OnTheBeach.png
- GroundTruth.png

Task 1: Finding starfish

Download the "OnTheBeach.png" image from the Session 4 area on Moodle, and write a Matlab script to display the image. You'll see the image contains a number of shells on a beach, including five orange-coloured starfish. Your goal in this task is to write a Matlab program that will automatically recognise the starfish in the image, and provide an accurate as possible segmentation.

There are many ways to complete this task; a suggested algorithmic workflow is presented in Figure 1.

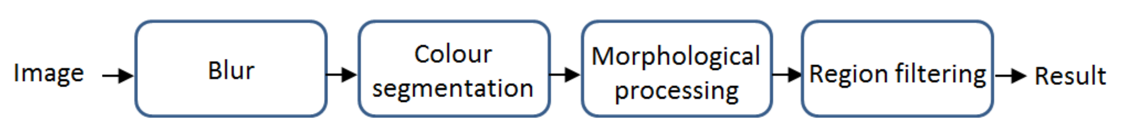
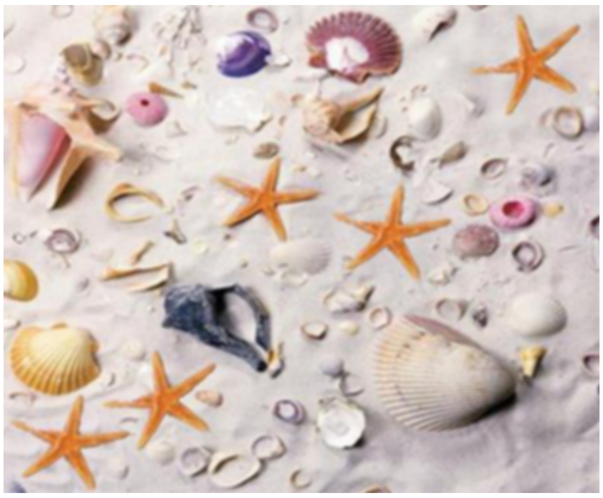


Figure 1. A suggested algorithmic workflow to segment the starfish from the image.

Task 1a: Blur

Images are corrupted by noise, which can be addressed using a denoising algorithm. A popular choice for denoising is blurring (a.k.a. low pass filtering), which reduces abrupt changes in image colour that is characteristic to noise. Whilst some blurring is often beneficial for image analysis, performing too much blurring can result in a loss of detail in the image. With this in mind, use the `imfilter` function, as discussed in Lecture 3, to blur the image by a small amount. Feel free to use any type of low pass filter, like a box or Gaussian filter, but don't excessively blur the image. My blurred image looks like this:



Task 1b: Colour segmentation

In this image, colour is one of the features that distinguishes the starfish from everything else in the image. With the blurred image shown in a Matlab figure, use the Data Cursor (shown in Figure 2) and click on some of the starfish pixels. This will give you an idea of the starfish colours in RGB colourspace:

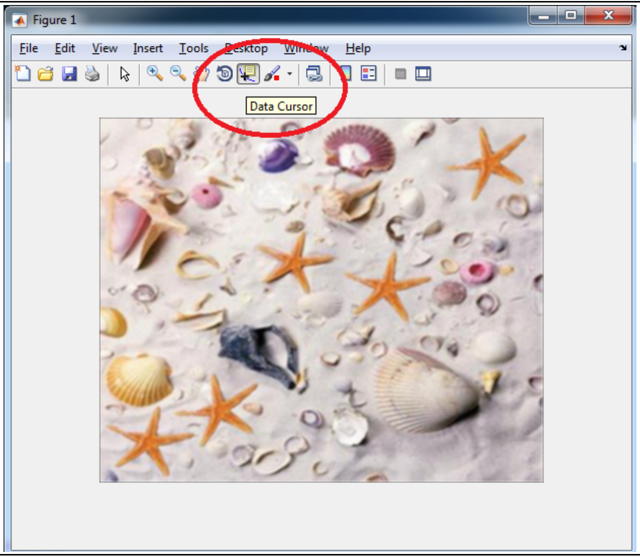


Figure 2. Using the data cursor

Try to find a (somewhat generous) range [R_low, R_high], [G_low, G_high], [B_low, B_high] that characterises the colours observed in starfish pixels. You can think of these ranges as a colour model. Based on this model, perform a colour-based thresholding of the image to find starfish-coloured pixels. It's like the thresholding we have seen in class, but applied with different values to each colour channel. The result should be a binary image J that has one for starfish-coloured pixels, and zero for non-starfish coloured pixels. If you're unsure how to do this, remember how you can use the logical operators (e.g. "&") to combine different binary images.

My colour segmentation looks like this:

Yours may look different; try to achieve a segmentation that has the starfish well segmented despite having other elements in the image still visible. We'll do some filtering later to remove the "clutter" from the non-starfish in the image.

Task 1c: Morphological processing

The boundaries of my segmented starfish are reasonable, but there are some holes in the starfish segmentations. This can be easily addressed using morphological processing. In fact, Matlab has a useful function `imfill` which can be used to fill holes in a binary image like this one. Use this function to fill holes in your colour segmentation. Store the output in a variable named K.

Task 1d: Region filtering

Most likely there is some clutter in your segmentation, with non-starfish objects part of the segmentation mask. The reason for this is that there are no single thresholds that uniquely identify starfish from non-starfish pixels in this image. So some additional processing is required in order to identify the starfish only.

Another feature that distinguishes starfish in the image is shape -- starfish have a star shape! Not only that, the starfish are fairly large compared to the smaller shells and "clutter" that appears in the segmentation mask. So in the last stage of processing we will apply a filter to remove regions that have incorrect shape characteristics or are too small.

Matlab has a very useful function called `regionprops` that computes basic shape characteristics for regions in image. The properties to compute must be set by the programmer as input arguments to this function. Examples include `'Area'`, `'Perimeter'`, `'Solidity'`, etc. – please see the documentation for the full list. The area of a region is approximated by the number of pixels in the region. The perimeter is the number of pixels on the boundary of the region. The solidity is the number of pixels in the region's segmentation mask divided by the number of pixels in the convex hull of the region. Recall the *convex hull* is the largest enclosing convex set that includes the region. An example is shown in Figure 3, where the outline of the convex hull is shown in red. In 2D, the convex hull is a convex polygon.

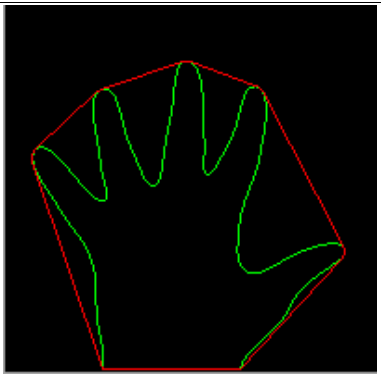


Figure 3. Convex hull (outline drawn in red) of a hand contour (shown in green).

Therefore, a convex region will have a solidity of one, since all its pixels are in the convex hull. However, a shape like a hand (or a starfish) will have a smaller percentage due to non-convexities in the shape. With this in mind, let's compute the region properties and filter regions to isolate the starfish regions. For example, if K is your binary colour segmentation mask, you can have regionprops compute the area and solidity of each region as

```
stats = regionprops(logical(K), 'Area', 'Solidity');
```

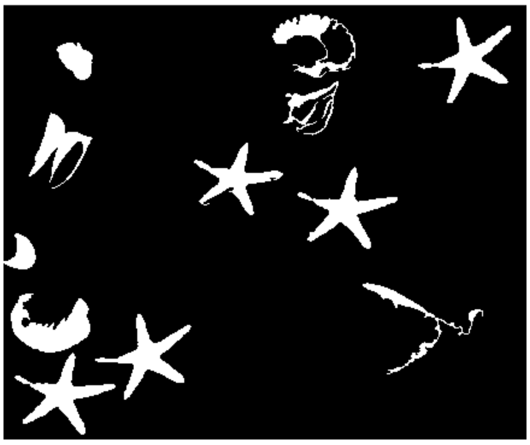
These measurements are returned in cell array (`stats` in the case above). In my case, I have 440 regions with measurements:

```
stats =

440x1 struct array with fields:

    Area

    Solidity
```

The example below shows how to produce a filtered output called result that only contains the regions that have an area greater than T (where T is a number).

```
ind = ([stats.Area] > T);
L = bwlabel(K);
result = ismember(L, find(ind));
```

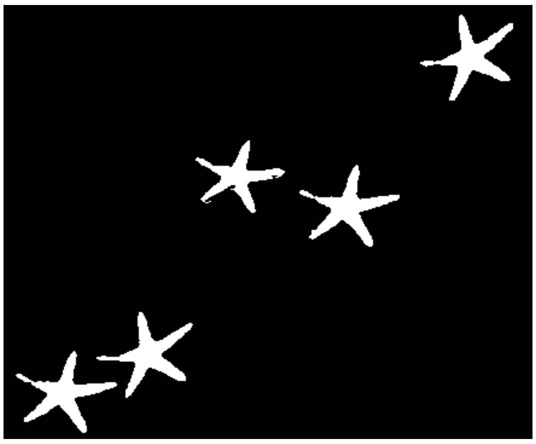
This finds the regions that satisfy the condition. It then finds the labelled regions that match the condition. Try setting a value for T, and visualising the result. My result looks like this:



You'll notice all the regions with a small area have been filtered out. Try including additional criteria on the line

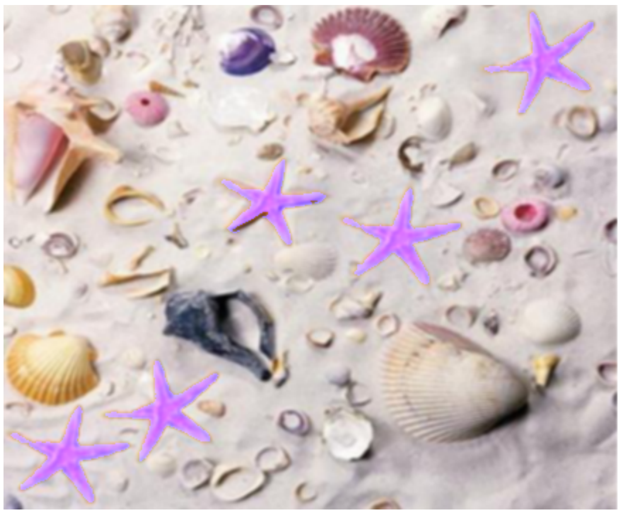
```
ind = ([stats.Area] > T);
```

to also filter based on low and high ranges for the solidity measurement. Based on this, my result looks like this:



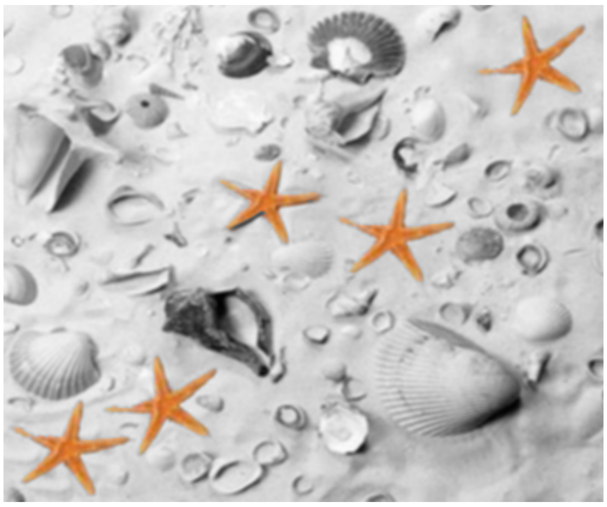
If your result is missing starfish, or includes extra clutter, try revisiting tasks 1a to 1d to get a segmentation with five starfish, or look for additional region properties to include in your filter.

Finally, recolour the original image based on the segmentation mask to change the colour of the starfish. For example, for starfish pixels, I set the blue channel to 255, but didn't change the red or green channels. The result looks like this:



Task 1e: Grayscale and colour

How would you create an effect where the image was grayscale, but only the starfish were in colour? This type of effect is sometimes used in pictures.



Note there are many ways this work could be expanded; for example, using different segmentation methods (active contour, graph cuts, k-means, superpixels). If we had a large number of images, it might be more practical to devise a classifier to identify "starfish". We will see how to do this in upcoming lectures.

Exercise 1: Use k-means (and some post-processing) to segment the starfish.

Hint: look at the lecture slides for code suggestions.

Task 2: How good is your segmentation?

As discussed in lecture, there are a variety of ways to determine the performance of a segmentation algorithm. A simple way is to compute a confusion matrix, and statistics like accuracy, sensitivity, and specificity.

Also in this week's Lab materials on Moodle is a ground truth mask, groundTruth.png. This is a mask which indicates which pixels should be part of the correct segmentation, and which are not. Load this image into the Matlab workspace. Use this image to compute the true positives, true negatives, false positives, and false negatives of your segmentation. If you're unsure how to do this, consult today's lecture notes. Once you have these, determine the accuracy, sensitivity, and specificity of your segmentation.

The performance of my segmentation algorithm is

Accuracy = 0.99086, sensitivity = 0.83718, specificity = 0.99935

What's yours (can you do better)? Why might the sensitivity be lower than the specificity?

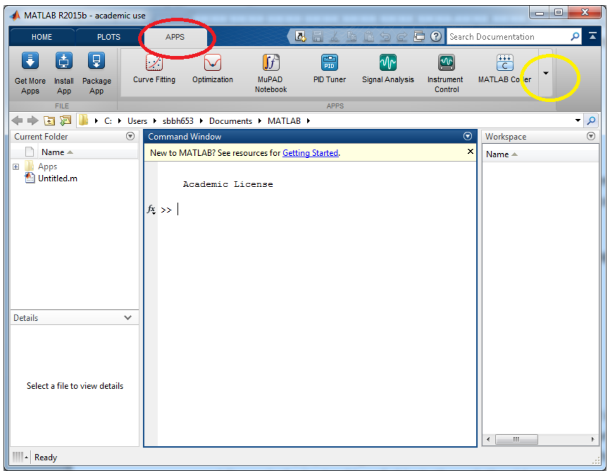
Note: the GroundTruth.png image was created manually (and rather hastily) by me.

Exercise 2: calculate the Dice-Sørensen Coefficient (DSC). While there is a Matlab function that does this automatically, try to implement it yourself and check the result.

Task 3: Active contour app

Matlab supports applications "apps" that provide a graphical user interface and run Matlab code. Matlab comes with a variety of apps you can use, and as a programmer, you can even create your own apps.

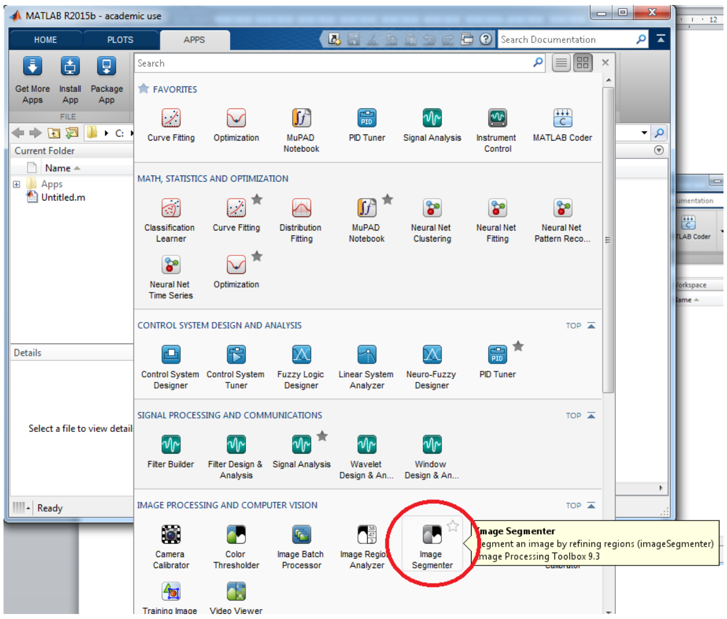
On the Matlab interface, select the "Apps" tab as indicated in the image below.



New to MATLAB? See resources for Getting Started.

```
Academic License

>> |
```

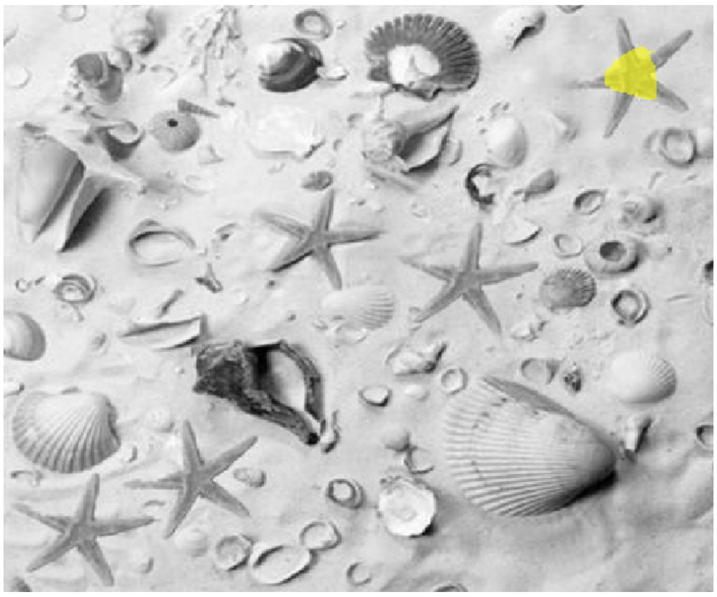
and then the Image Segmenter (you may have to push the down arrow button in yellow above).



mage Segmenter
egment an image by refining regions (imageSegmenter)
mage Processing Toolbox 9.3

This app runs the active contour model, and allows the user to provide an initialisation.

In the Image Segmenter app, Load the “OnTheBeach.png” image. You will get a warning that Matlab expects the input to be grayscale. Click ok. Then, click on the Draw Freehand button, select Draw Freehand, and draw around a starfish.



Press the Active Contours button. Then, click the Evolve button, which will run the Chan-Vese segmentation PDE (the default) for 100 iterations. You can try other starfish and other initialisations to see how the method works. Based on your understanding of the active contour code in Matlab, how could you automatically perform a Chan-Vese segmentation of the starfish in the image?