Computer Vision
Workshop 6
Extracting Properties from Regions

Aims of the workshop

The aim of this session is to follow on from Workshop 5, where we explored some connected components and morphology, introducing region properties from segmentation masks.
As such, **Workshop 5 must be completed prior to these exercises**. These exercises will utilise concepts taught in recent lectures around features.

Some code will be provided in this workshop, please see 'Useful Information' below on how to look up MATLAB functionality.

Feel free to discuss the work with peers, or with any member of the teaching staff.

Useful Information

MATLAB Documentation

MATLAB functionality mentioned during lecture or workshops is fully-documented and available at: https://uk.mathworks.com/help/matlab/

You may find the “Getting Started” section of this documentation to be useful: https://uk.mathworks.com/help/matlab/getting-started-with-matlab.html

In addition, MATLAB Answers may be useful in finding common solutions to any issues you may encounter: https://uk.mathworks.com/matlabcentral/answers/help?s_tid=gn_mlc_ans_hlp

Useful functions: bwareaopen, imfill, regionprops, zeros, disp, fprintf, rectangle, circle, line, ismember, min, max.

Reminder

We encourage you to discuss the contents of the workshop with the delivery team, and any findings you gather from the session.

Workshops are not isolated, if you have questions from previous weeks, or lecture content, please come and talk to us.

The contents of this workshop are not intended to be 100% complete within the 2 hours. This session involves substantial comparisons and critical thinking, and as such it's expected that some of this work be completed outside of the session. Exercises herein represent an example of what to do; feel free to expand upon this and make your own comparisons, in addition, if you wish.

For Loops

Today's session will look at region properties from a connected components matrix. When dealing with connected components, we need a way of iterating through each identified component; making some decisions about them. Is it round enough? Is it big enough to be counted as a coin? To achieve this we can use a for-loop within MATLAB over these regions.

When looking at region properties from our segmentation masks, we often get a structure which contains as many rows as we have components, where each column is a metric we have selected. This might look something like the following:

properties_table = 5×23 table

	Area	Centroid		BoundingBox			
1	11297	116.5319	388.5541	57.5000	327.5000	118	122
2	11253	151.0052	136.0194	77.5000	61.5000	147	149
3	10654	355.1482	351.6170	286.5000	263.5000	155	196
4	8094	487.0262	160.9291	406.5000	79.5000	161	145
5	10856	637.0403	367.7617	574.5000	227.5000	125	253

We can obtain the number of components either directly from our connected components function, or by using *size(our_region_props_variable, 1).*

```
for i=1:numberOfRegions
    % Calculate Area, Perimeter of each region.
    regionArea = our_region_properties_variable(i).Area;
    regionPerim = our_region_properties_variable(i).Perimeter;

    % Calculate some metrics for our given region.
    roundness = regionPerim^2 / 4 * 3.141 * regionArea;

    % Either store these for later use, or display them, etc.
end
```

Here we iterate from 1, to the size of our region properties (how many regions we have). Remember the : notation when we were indexing?

Using this we can directly index the specific row by performing our_region_properties_variable(row_we_want). For example *rprops_var(4)*. We can then directly access the named attributes of this row, similar to how you might do in other languages (using dot).

Once we can access the given attributes of a specific component (from our segmentation mask), we can then use these to calculate other additional metrics. In the case of perimeter and area, we can calculate the roundness of our object to choose the best circle.

If we wanted to store these for use outside of the loop, we might create an empty matrix using zeros. We can then make this multi-dimensional, to store as many metrics as a want. E.g *metrics = zeros(numberOfRegions, 27);*

This will create a 4 by 27 matrix (if numberOfRegions = 4, in our example). We can store 27 metrics per component.
To assign to this we can use *metrics(i ,1) = roundness;* Where roundness is our first metric. Or *metrics(i, metric_number) = metric_value;* in the general case.

Note: In these workshops, and the ACW, you can work in either .m or a .mlx “live script”. Feel free to experiment with using either, and see which you find most useful.

The below tasks require Exercise 5 of Workshop 5 to be completed beforehand (Connected Components).

The input to the following tasks should be the *bwlabel* output from exercise 5. This is a binary (logical) image of 1’s and 0’s - representing the segmented coins from the previous workshop. This is NOT the RGB labelled components image.

Exercise 1: Look at the documentation for **bwareaopen**. This function accepts connected components (NOT labelled connected components) and filters these based on size/area.

1. Use this to remove any accidental or erroneous small components from your final mask.
2. If there are any holes in your detected coins, due to the effect of noise in your morphology processes, consider using *imfill(image, ‘holes’);* to make these regions whole.

Do not worry if there are no erroneous detections, holes, or bridges. If this is the case, and you are correctly identifying the number of coins, you may wish to purposefully make a bad morphology step, in order to complete this exercise.

Exercise 2: Using *regionprops*, gather a list of centroid locations, areas, and bounding boxes; printing them.

Hint: We will need to iterate over each row, and get our values. Consider using *fprintf* with the following template string:

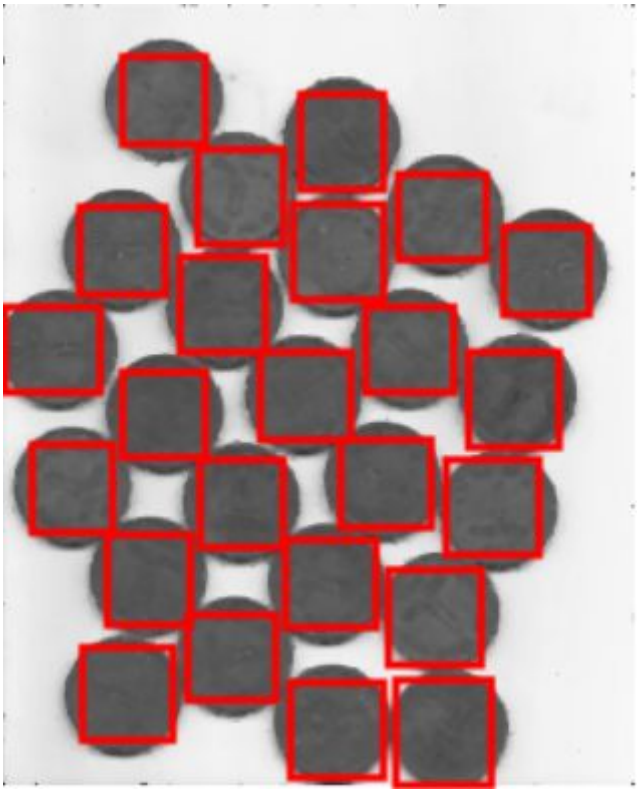
```
'%i %0.1f %0.1f %i %0.1f %0.1f %0.1f %0.1f\n'

%i = integer
%0.N f = number of floating points. Eg 0.3f = 0.002 (precision).
\n = newline.
```

Output should look something like the following:

```
ID Centroid-x Centroid-y Area Bbox1 Bbox2 Bbox3 Bbox4
1 21.9 138.5 1059 1.5 121.5 38.0 34.0
2 28.9 194.0 1004 12.5 176.5 32.0 35.0
3 48.6 99.4 1035 31.5 81.5 34.0 35.0
4 49.7 276.4 1053 32.5 257.5 36.0 37.0
```

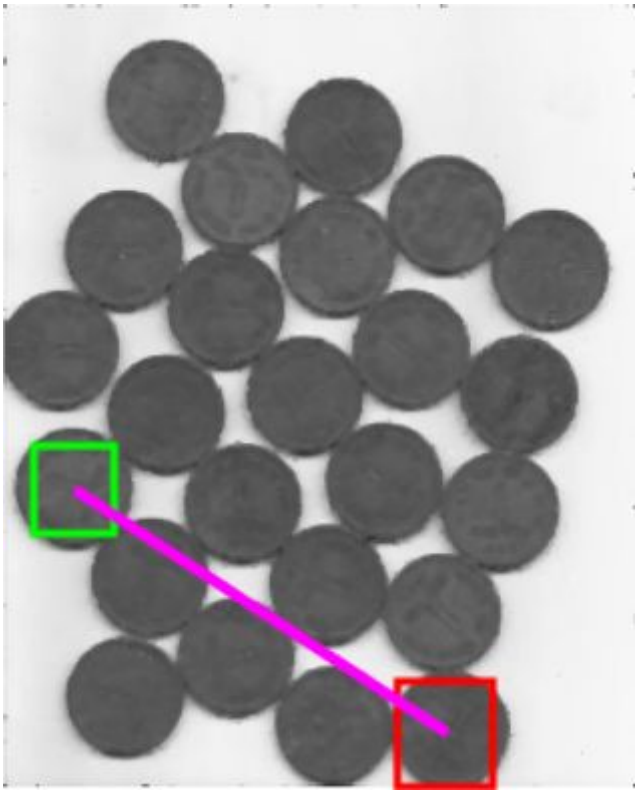
Exercise 3: Using the bounding box information, overlay boxes over the de-noised coins image. (You may have to gather this from last week's workshop solutions yourself). Remember to show the original image, and use *hold on* to then draw rectangles over that image.



Exercise 4: Find the largest, and smallest, coin by filtering the Area property of each connected component.

1. Display the segmentation mask of the largest coin. This will be a binary image, where 1's represent just the largest coin.
2. Overlay a Red bounding box for the largest coin onto the de-noised image.
3. Overlay a Green bounding box for the smallest coin onto the de-noised image.
4. Calculate the distance between the largest coin, and the smallest coin.
5. Draw a magenta line between the center of the two coins.

Hint: *min* and *max* may help with this task, especially as it also can return the index. *ismember* will let you filter labeled connected components by a given index.



The Extended Exercises are **optional**, and are offered as an advanced supplement for those who have completed the existing work and wish to expand on their practical knowledge; challenging themselves further.

Extended Exercise 1: Using additional properties such as MajorAxisLength, MinorAxisLength, and Orientation:

1. Perform Ex 3 again; replacing the Axis-Aligned Bounding Boxes (AABB) given by regionprops with your own calculated Object Oriented Bounding Boxes (OOBB).

This will require an understanding of Major and Minor axes, and how these may form the dimensions of a bounding box, in conjunction with an orientation of the given shape. In addition, you will then need to generate the four coordinates required to create a 4-poly for display.

Consider the differences between axis aligned, and object aligned boxes. How might these impact computation time, and accuracy? When might OOBB be preferred over AABB?