CS4302 Signal Processing Practical 2

Audio and Image Analysis

Overview

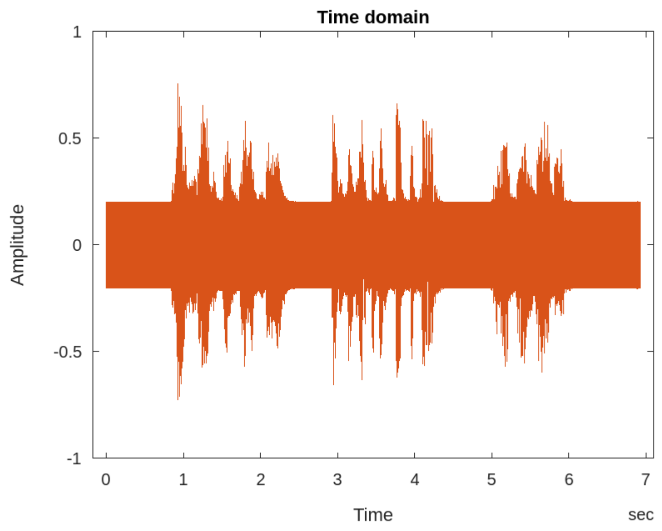
I have identified the noise frequencies of the three given audio clips and removed the majority the noise using MATLAB. For the image processing part of this practical, I have tried every task (tasks 1-6). Tasks 1-4 have good accuracy up until the extreme difficulty image, however task 5 only has good accuracy on the easy image. I have combined my methods in task 6 with similar results.

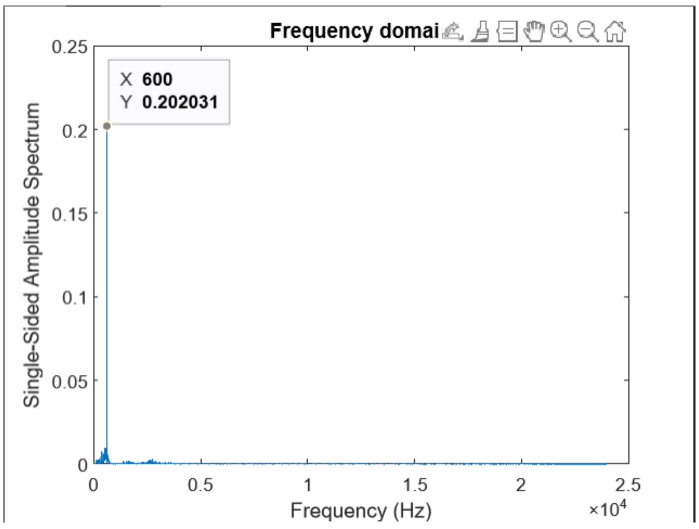
Solutions to each task have been split up into their own script files, e.g., “Q1”.m, “Q2.m”, etc. that generate their own figures in their corresponding directory, e.g., “Q1”, “Q2”, ...etc. A run script has been provided to run all the tasks with one command, simply run the “run.m” script with the “run” command in MATLAB. This will take a little bit of time to run because the program will be saving figures and processing the data. I have not shown all the figures in this report to make it shorter and easier to read, however, they can still be viewed in files. The figures that are included are those I believe to be most useful in understanding the methods used.

Audio Signal Processing

Task 1 Identifying the Frequency of the Noise

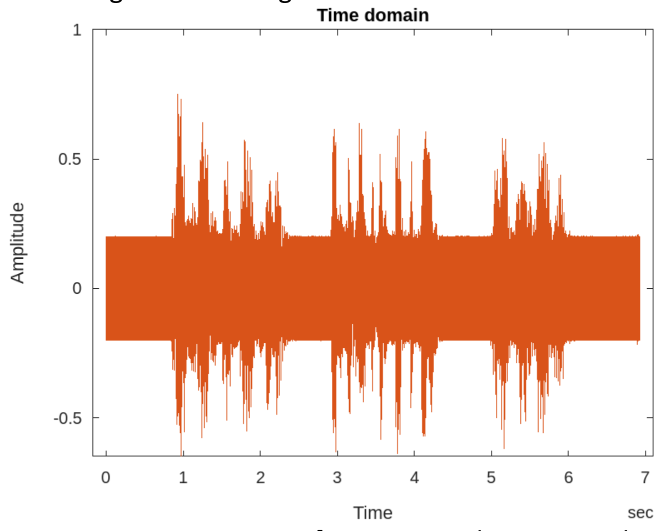
I identified the frequency of the noise by plotting the frequency as a single-sided spectrum, this made the plot easier to understand as it would only show the positive frequencies that would have otherwise been mirrored in the negative frequencies. The figures below show how I identified the noise frequencies in MATLAB by analysing the figures generated by Q1, as well as a plot for the audio signal prior to removing the noise.

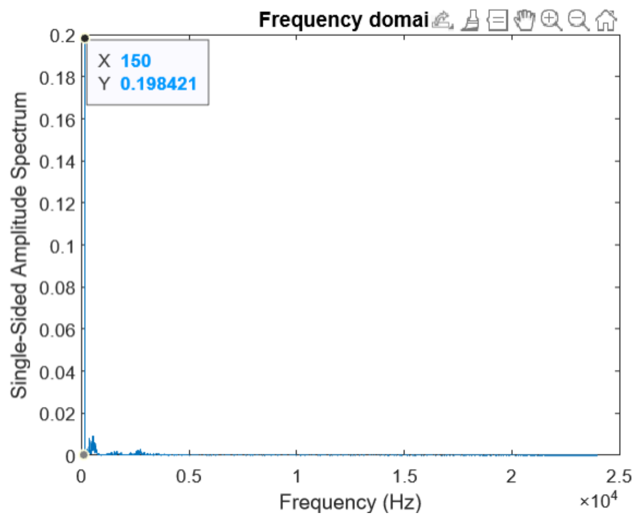




[Figures 1 and 2: Time and Frequency domain plots for Audio 1.]

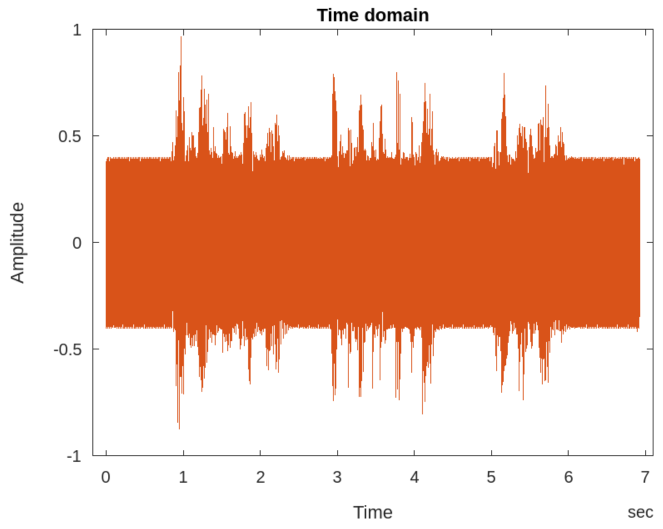
audio_in_noise1.wav has a 600Hz noise frequency, this is shown in figure 2.

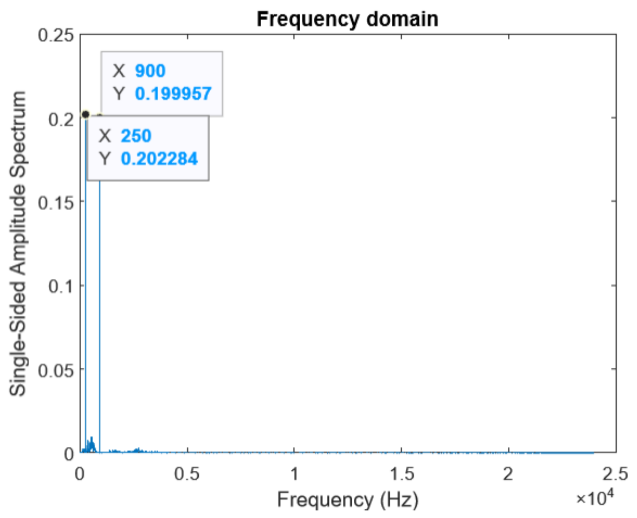




[Figures 3 and 4: Time and Frequency domain plots for Audio 2.]

audio_in_noise2.wav has a 150Hz noise frequency, this is shown in figure 4.

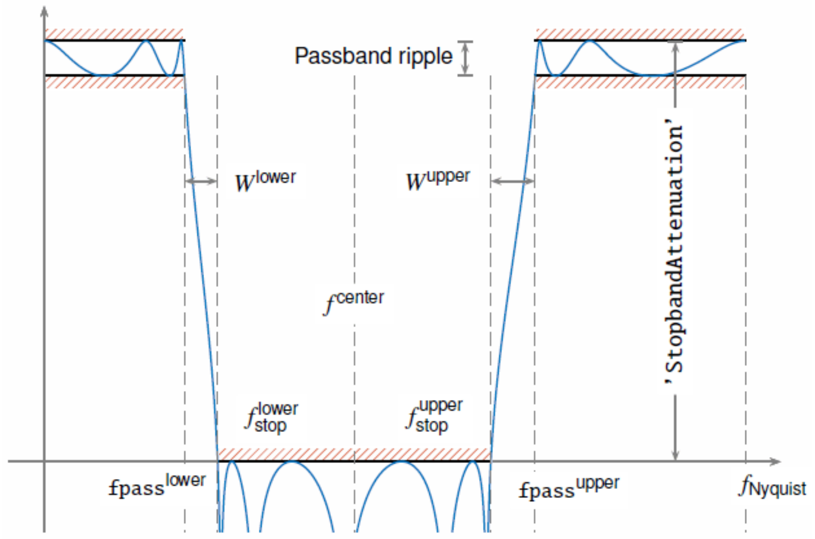




[Figures 5 and 6: Time and Frequency domain plots for Audio 3.]

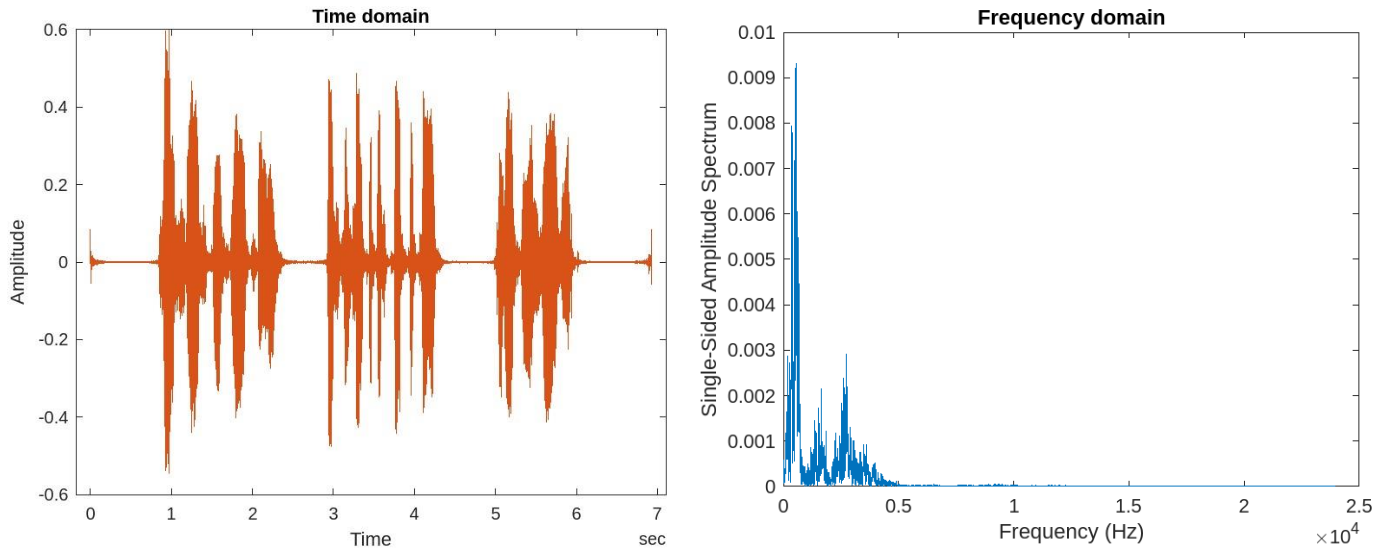
audio_in_noise2.wav has 250Hz and 900Hz noise frequencies, this is shown in figure 6.

Task 2 Frequency Filtering and the Sounds



[Figure 7: "Bandstop Filter Steepness"[1].]

To remove the previously identified noise frequencies, I used a band-stop filter. The band-stop filter explained in the lectures works by multiplying frequencies in a signal either by 1 or 0 – only allowing the desired frequencies through. The MATLAB band-stop filter function also uses steepness to reduce frequencies at the edges of the range less for the length of W^{lower} and W^{upper} which defaults to 0.85 – 85% of (f^{center} – $fpass^{lower}$). This can be seen in the above figure from the MATLAB documentation.[1] I used the default steepness as it appeared to have the best effects on audio quality and used applied the band-stop to frequencies with a range of 50Hz with the values discovered in task 1 as f^{center}.



[Figure 8: Example audio plots after noise reduction in the time and frequency domain.]

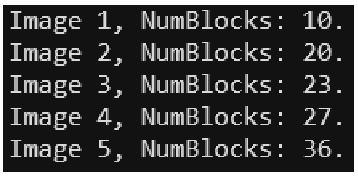
The audio with the noise removed, which sounds like a person saying "Hello CS4302, best of luck with this practical. I hope you're enjoying it", has some leftover noise artefacts. The noise previously sounded like an extended ringing sound, which is now largely removed. The left noise artefacts are because a signal of a person talking has many different frequencies and may suddenly start and stop. "Fourier analysis is usually not able to detect those events [abrupt changes]"[2].

[1] MathWorks, bandstop, Date Accessed: 24/11/2023

[2] MathWorks, Detecting Discontinuities and Breakdown Points, Date Accessed: 24/11/2023.

Image Analysis

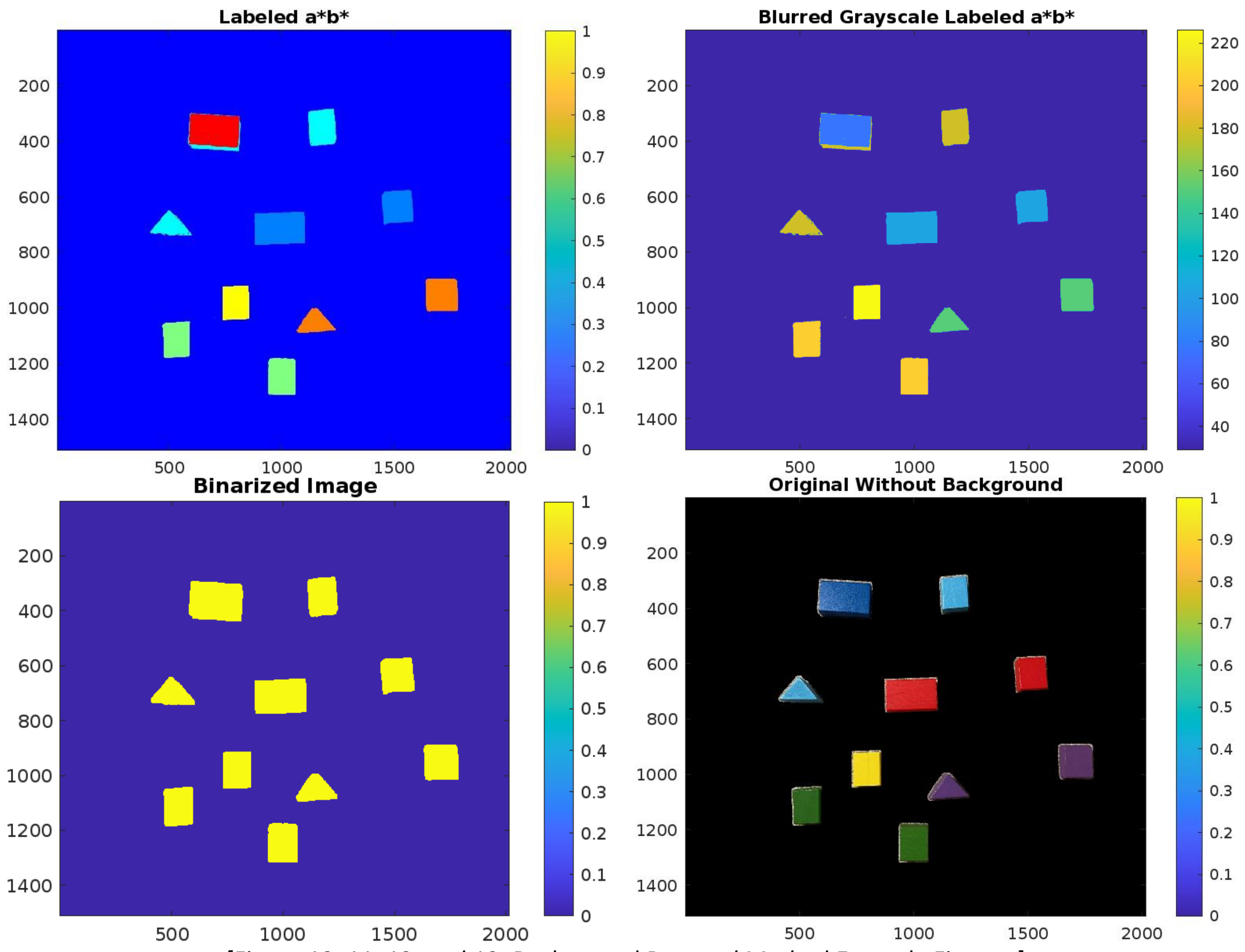
Task 3 Block Counting



```
Image 1, NumBlocks: 10.
Image 2, NumBlocks: 20.
Image 3, NumBlocks: 23.
Image 4, NumBlocks: 27.
Image 5, NumBlocks: 36.
```

[Figure 9: Program output for block counting.]

My program correctly counts the number of blocks in the first 4 images. This is shown in the above printed output for Q3. The last image should have 27 blocks counted; however, my program counts 36 – some shadows and areas between shapes are counted as their own shapes. Later I will explain how this could be improved using processing developed in later tasks (colour classification). My program was given the number of colours in each image to count the number of blocks with k means clustering– this helped with the accuracy of which I was able to remove the background carpet.



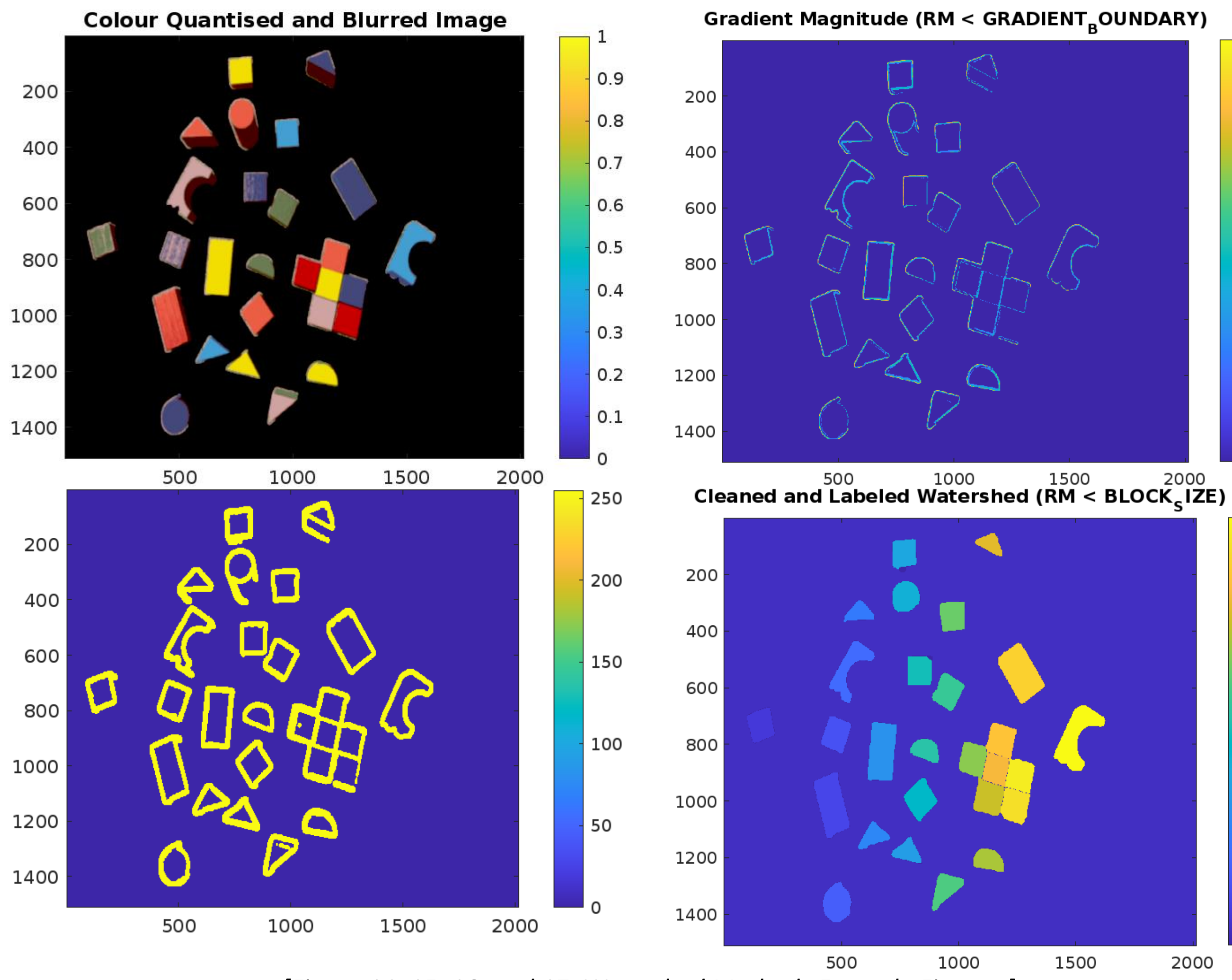
[Figure 10, 11, 12, and 13: Background Removal Method Example Figures.]

The first step I took in counting the blocks in each image was by remove the background carpet. To do this I converted the image from RGB to the L*a*b* colour space and then used K-means clustering[3] to segment the objects by colour. This colour space takes into consideration the luminosity, red-green values, and blue-yellow values of a colour to classify pixels.[4] I chose to use K-means clustering because I could specify how many different colours there are in the image to segment it. I was planning on using this technique instead of watershed for colour classification,

[3] Mathworks. Color-Based Segmentation Using K-Means Clustering. Date Accessed: 24/11/2023.

[4] Mathworks. Color-Based Segmentation Using the L*a*b* Color Space. Date Accessed: 24/11/2023.

however it did not distinguish well between colours that were similar, e.g., vivid red and dull red. Instead, I used this to make a binary image that can be used as a mask to remove the background carpet. For the images where blocks are not touching, this can be used to count the number of blocks by how many connected areas there are.



[Figure 14, 15, 16, and 17: Watershed Methods Example Figures.]

Counting blocks that were touching required much more work as I had to segment the image based on smaller changes in colour. The first step was to make the image smoother so that when I get the gradient boundaries there is less unhelpful noise. Blurring the image, quantising the colours (so that there were only 10 colours), and removing small changes in gradient, made the gradient boundaries much more useful. I used all the gradient boundaries that met these criteria to create a binary image that was watershed. In the final cleaned and labelled watershed image (figure 17) blocks that were too small were removed, i.e., less than 4600 pixels. This threshold value was discovered increasing/decreasing the value by increments of 100, until the best results were found.

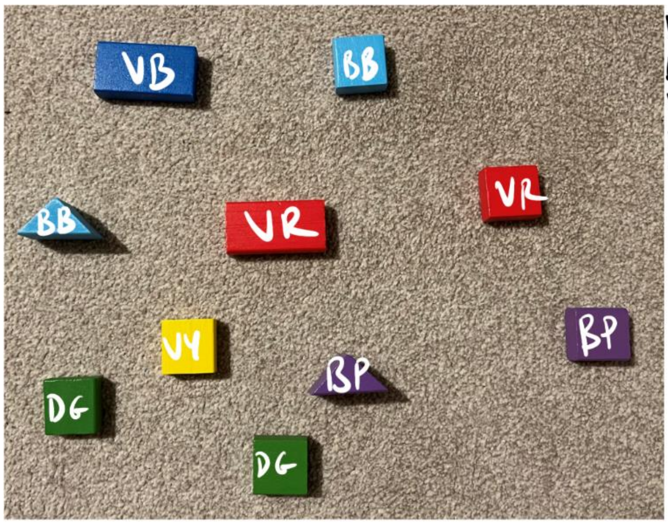
Task 4 Coloured Block Counting

The Color Blind Pal mobile app was used to identify colours and label them in the below image. This image was used as a reference and tool for checking the accuracy of my program.

KEY
VB = Vivid Blue
BB = Bold Blue
VR = Vivid Red
VY = Vivid Yellow
BP = Bold Purple
DG = Deep Green
DR = Dull Red
BO = Bold Orange

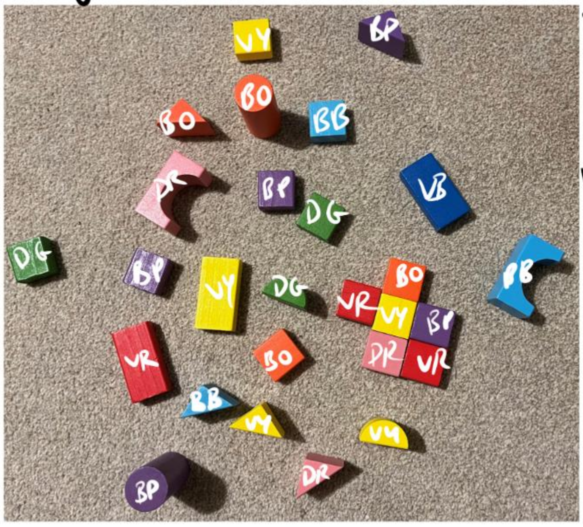
T = Total Blocks
N = Num Colours

easy



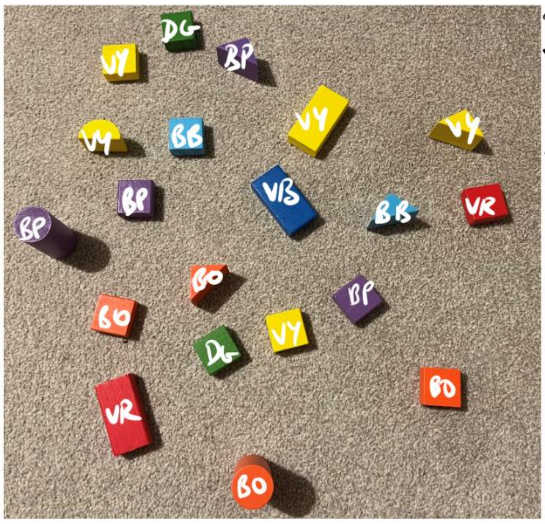
VB=1
BB=2
VR=2
VY=1
DG=2
BP=2
T=11
N=6

very hard



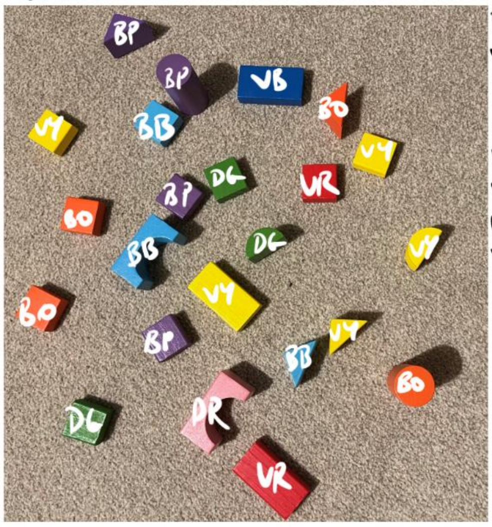
VY=5
BP=5
BO=4
DR=3
BB=3
DG=3
VR=3
VB=1
T=27
N=8

medium



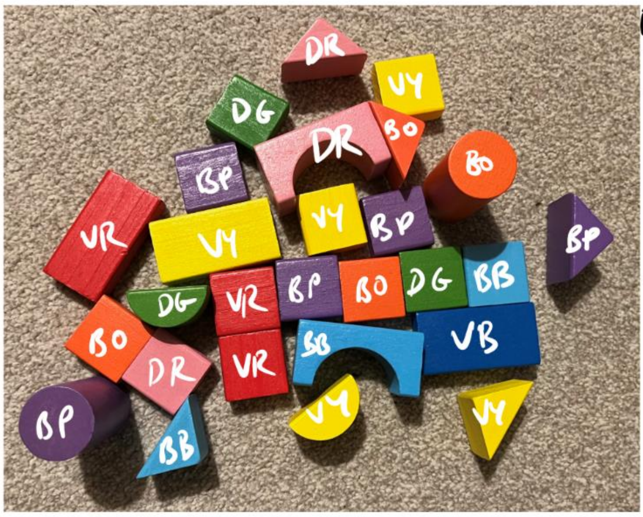
DG=2
VY=5
BP=4
BB=2
VB=1
BO=4
VR=2
DG=1
T=21
N=8

hard



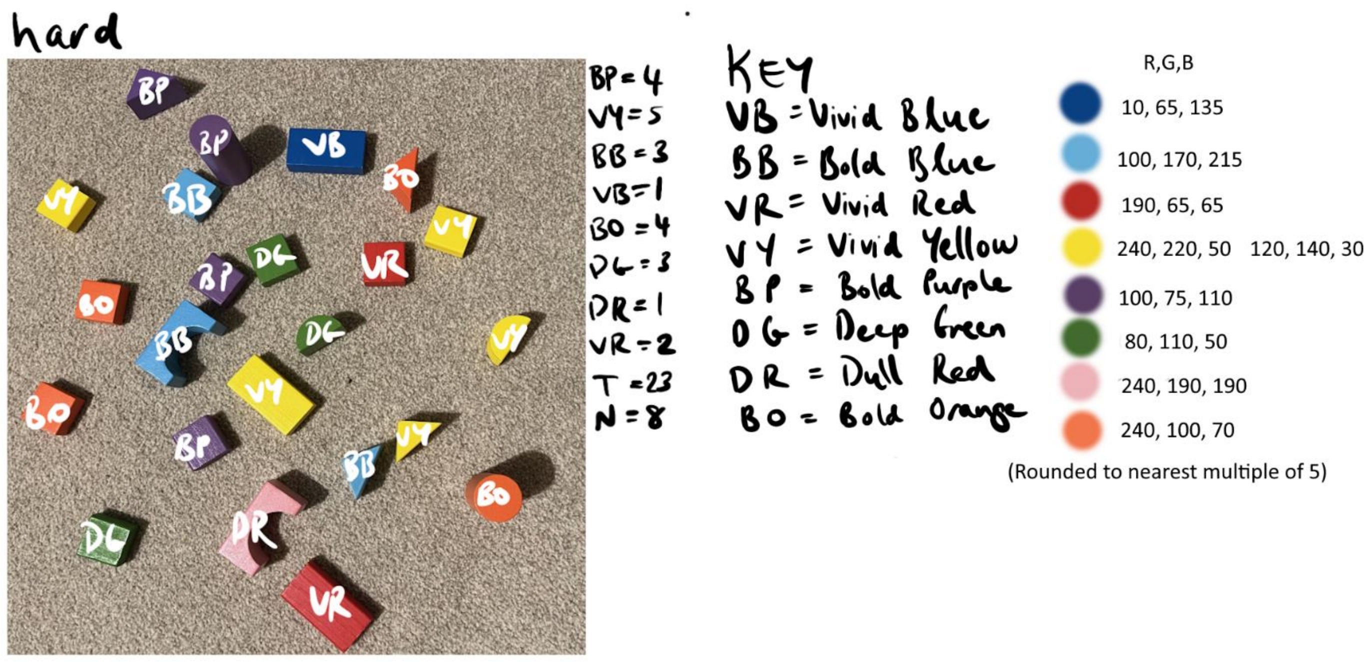
BP=4
VY=5
BB=3
VB=1
BO=4
DG=3
DR=1
VR=2
T=23
N=8

extreme



DR=3
DG=3
VY=5
BP=5
BO=4
VR=3
BB=3
VB=1
T=27
N=8

[Figure 18: Labelled images dataset.]



[Figure 19: Picked colours data set.]

After labelling the colours in figure 19, I used the colour picker tool in paint.NET[5] to determine the RGB values of the blocks, these colours where then given to my program to classify colours. I found that providing a shadow shade for yellow (where the RGB values change quite drastically) greatly improved the accuracy. To improve the accuracy of my program further, I could have provided more sample shades for each colour, for example, a shadow shade and a highlight shade.

To determine the colour of a block, I took the mean RGB values for an area and compared them with the colour data set in figure 19. A COLOUR_VARIANCE_VALUE and distance heuristic was used to determine which colour was the closest. The COLOUR_VARIANCE_VALUE is a limit in the difference between a single RGB value to a named colour's corresponding RGB value. The distance is the difference between each value added together. For example (2, 5, 9) to (9, 2, 4), has the difference value 14 (7 + 3 + 4).

The colour with the smallest distance within the COLOUR_VARIANCE_VALUE threshold is what is used to label a given colour. In the extreme image, where areas of carpet and shadow are provided, this results in an "Uncertain" colour label being given as the RGB values are too different from the colour data set – they are not within the threshold values.

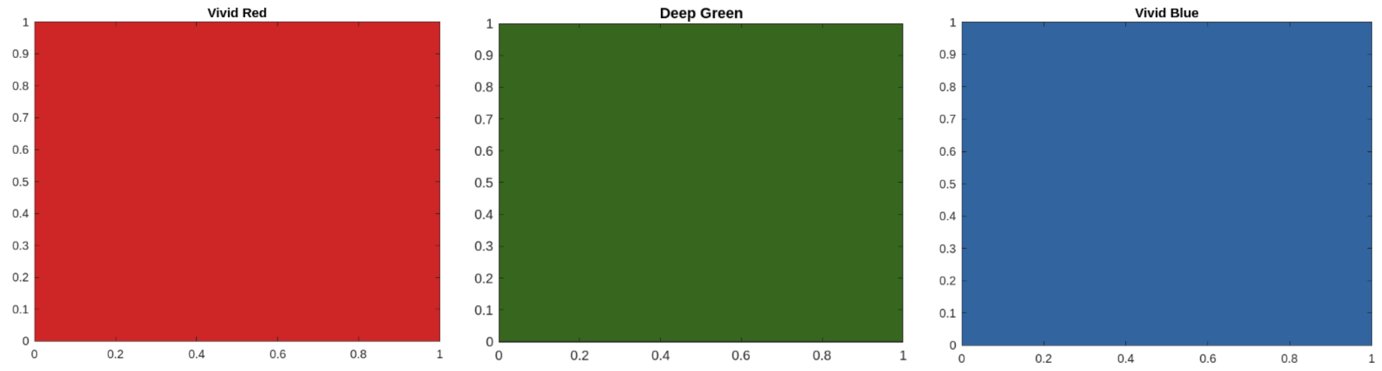
Image Num	Vivid Blue	Bold Blue	Vivid Red	Vivid Yellow	VY (Shadow shade)	Bold Purple	Deep Green	Dull Red	Bold Orange	Uncertain	Total
1	1	2	2	1	0	2	2	0	0	0	10
2	1	2	2	4	1	4	2	0	4	0	20
3	1	3	2	5	0	4	3	1	4	0	23
4	1	4	3	5	0	4	3	1	6	0	27
5	1	3	3	5	1	6	3	3	4	7	29

[Figure 20: Colour Classification Results. Key: Correct classifications, Incorrect Classifications.]

Generally, my program is quite accurate when it comes to colour classification and starts to fail on harder difficulties. It has some trouble determining the difference between bold blue and bold purple, and dull red and bold orange.

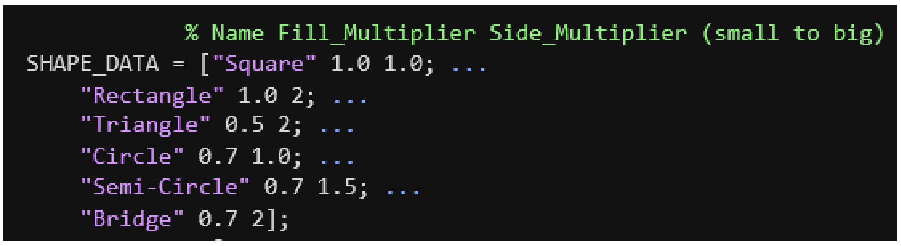
[5] dotPDN LLC. paint.NET. 2023.

Colour classifications have improved the accuracy of block counting, as can be seen in the uncertain category where areas that are not blocks have been separated.



[Figure 21, 22, and 23: Example colour classifications.]

5 Shape Detection



```
            % Name Fill_Multiplier Side_Multiplier (small to big)
SHAPE_DATA = ["Square" 1.0 1.0; ...
    "Rectangle" 1.0 2; ...
    "Triangle" 0.5 2; ...
    "Circle" 0.7 1.0; ...
    "Semi-Circle" 0.7 1.5; ...
    "Bridge" 0.7 2];
```

[Figure 24: Shape data set.]

My shape detection methods use a set of a data on the given shapes, this includes their name, fill multiplier, and side multiplier. The fill multiplier is how much of the area's window is filled by the shape – the window is the width of the area the shape is in multiplied by the height. For a square it is expected to be filled in a perfect scenario as the area of a square is the width multiplied by the height, so the fill multiplier is 1.0. However, for a triangle, this is expected to be half because the area of a triangle is the width multiplied by the height divided by two.

The Side multiplier is how much the smallest side should be multiplied by to be approximately equal to the larger side. For example, for a square where the width and height should be the same, the side multiplier is expected to be 1.0, and for a rectangle it is 2.0.

For some of these measurements I have made a rough estimate by looking at the shapes, rather than measuring them in the images, which I did not have time for. The side multiplier would likely be a much more accurate metric if I rotated the shapes so that they were in the smallest possible dimensions. I started implementing this (which can be seen in my code) but did not have time to finish it. I predict that this would greatly improve shape classifications for harder difficulties.

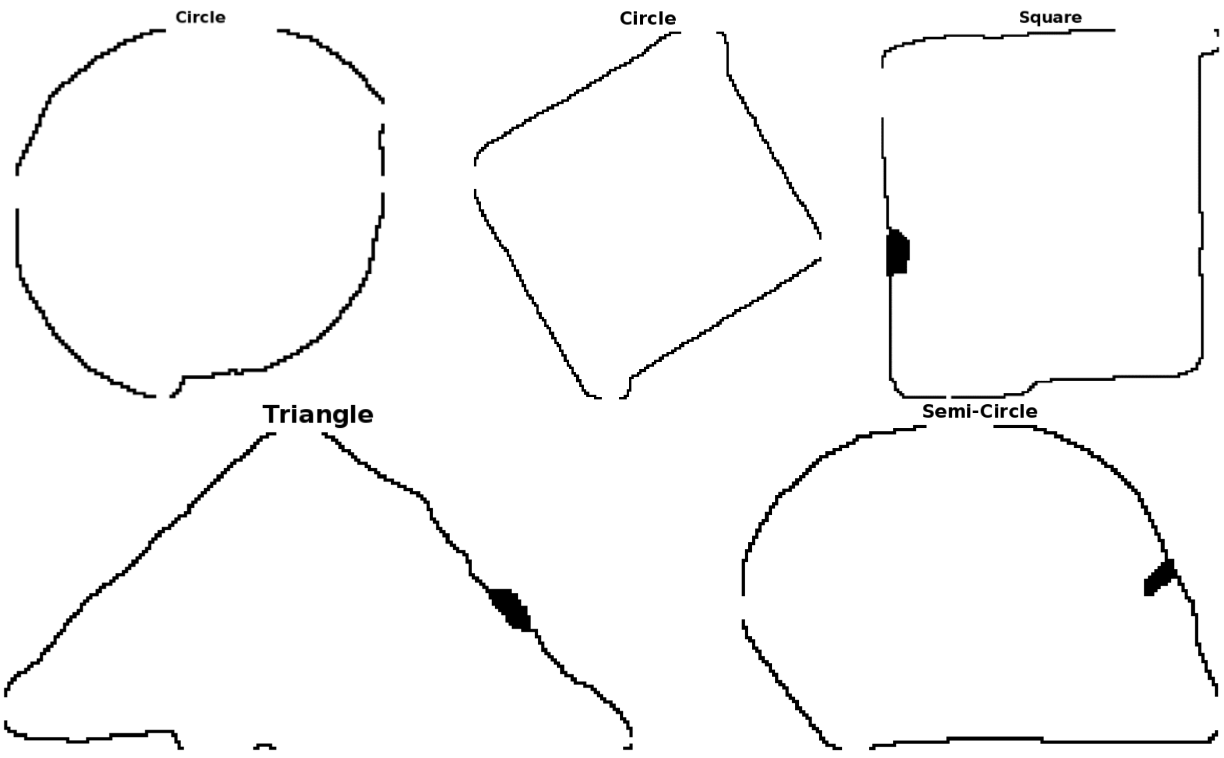
A difference heuristic is used to determine which shape name matches best, which compares expected and real values with these multipliers. This method for shape detection is correct for the first image, but then becomes much less accurate because of the increased image complexity – namely the rotation of blocks. This can be seen in the below figures. In figure 26, I have shown the difference between the number of a shape counted in an image and the number of expected shapes. There are negative values for undercounting, and positive values for overcounting.

Image Num	Square	Rectangle	Triangle	Circle	Semi-Circle	Bridge	Total
1	6	2	2	0	0	0	10
2	5	0	1	10	4	0	20
3	1	1	3	14	3	1	23
4	3	0	2	14	8	0	27
5	0	1	14	16	3	2	36

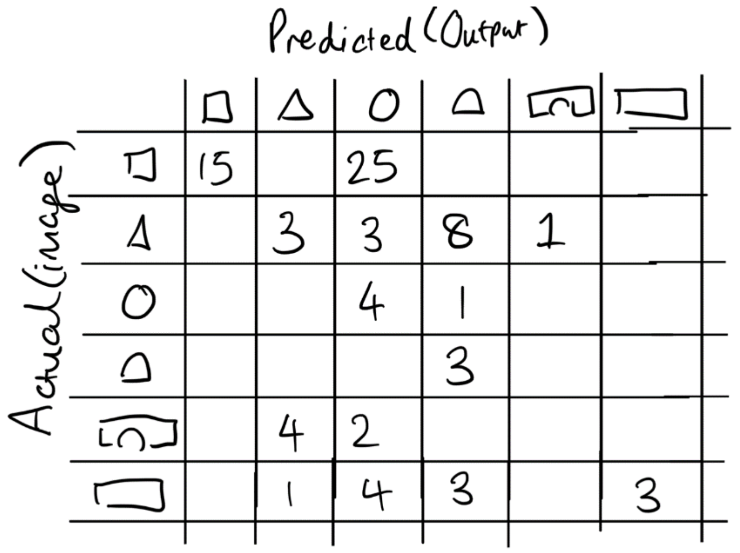
[Figure 25: Number of shapes counted in each image. Key: Correct classifications, Incorrect Classifications]

Image Num	Square	Rectangle	Triangle	Circle	Semi-Circle	Bridge	Total
1	0	0	0	0	0	0	0
2	-5	-3	-3	8	3	0	0
3	-9	-2	-1	12	1	-1	0
4	-10	-3	-3	12	6	-2	0
5	-13	-2	9	14	1	0	9

[Figure 26: Difference between shapes counted and expected count. Key: Correct classifications, Incorrect Classifications]



[Figure 27, 28, 29, 30, and 31: Example Shape Classifications.]


Predicted (Output)
Actual (image)

	□	△	○	⌓	⊓	▭
□	15		25			
△		3	3	8	1	
○			4	1		
⌓				3		
⊓		4	2			
▭		1	4	3		3

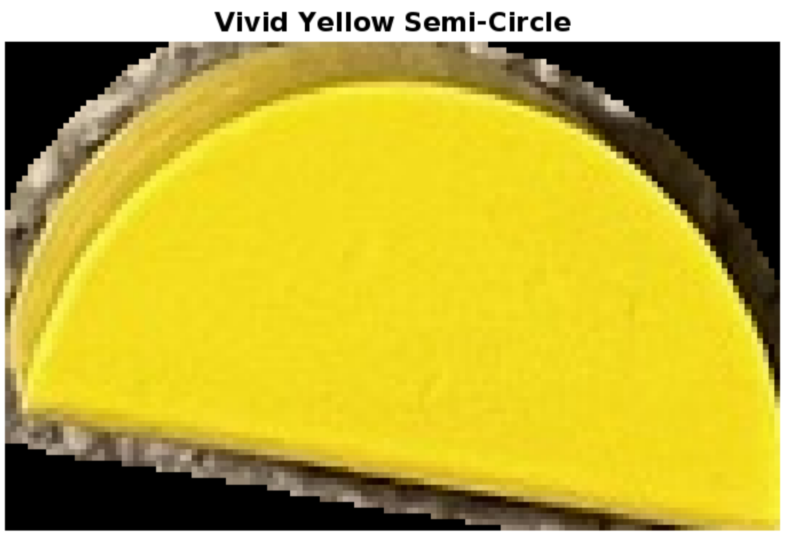
[Figure 32: Shape Classifications Confusion Matrix, counting segmented images from the first 4 original images.]

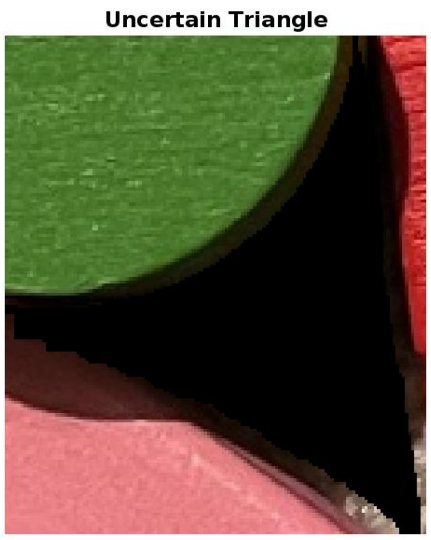
Figure 32 is a confusion matrix for my shape classifications that I made by looking at the output of my program for the first 4 original images. This matrix supports my conclusions about struggling to read shapes if they are at an angle

– which would most impact the longer shapes, i.e., the rectangle and bridge shape. On the top left to bottom right diagonal are the correct classifications.

6 Combining of Colour and Shape Detection

I combined my methods for colour and shape detection together and only had time to make them work separately from one another, an example of an outputted classification can be seen in the below figures. An improvement of this would be to do colour recognition first as it is more accurate to remove blocks that are not one of the colours in the given colour scheme. This would prevent strange classifications such as the one in figure 34.





[Figure 33 and 34: Example Coloured Shape Classifications.]

References

dotPDN LLC. paint.NET. Released 2023. Available from https://www.getpaint.net/.

MathWorks. Bandstop. Available from https://uk.mathworks.com/help/signal/ref/bandstop.html. Date Accessed: 24/11/2023.

Mathworks. Color-Based Segmentation Using the L*a*b* Color Space. Step 2: Calculate Sample Colors in L*a*b* Color Space for Each Region. Available from: https://uk.mathworks.com/help/images/color-based-segmentation-using-the-l-a-b-color-space.html. Date Accessed: 24/11/2023.

Mathworks. Color-Based Segmentation Using K-Means Clustering. Available from: https://uk.mathworks.com/help/images/color-based-segmentation-using-k-means-clustering.html. Date Accessed: 24/11/2023.

MathWorks. Detecting Discontinuities and Breakdown Points. Available from https://uk.mathworks.com/help/wavelet/ug/detecting-discontinuities-and-breakdown-points.html. Date Accessed: 24/11/2023.

Vincent Fiorentini. Color Blind Pal. Google Play. Available from https://play.google.com/store/apps/details?id=com.colorblindpal.colorblindpal&hl=en_GB&gl=US. Date Accessed: 24/11/2023.