Auditory and Visual Signal Processing

Sami Hatoum

1 Overview

In a world where we have become increasingly reliant on digital data findings, it is imperative that the incoming signals we source are processed, cleaned, and classified so that they are suitable for any subsequent courses of analysis. This report will cover varying techniques and methodologies employed in the processing of visual and auditory data in MATLAB, specifically through identification and extraction of noise in the frequency domains for a set of signal inputs.

Both scripts can be run using the script *run_scripts.m*
All plots used in this report can be found under *Data/<Audio or Image>/*

2 Audio Signal Processing

2.1 Design Plan

This first half of this practical, Parts 1 and 2, regard auditory data; these audio files each consist of a talking voice as well as a continuous static frequency (or two) playing in the background. Tasked with filtering noise from these files, it must first be established what the noise is considered to be. A form of frequency-range filtering can then be selected to attune certain sections of the frequency spectrum.

Noise, in the general sense, depends on context and the listener's intent. In either case, it can be argued one side of the audio is disrupting the other. If the static frequency is noise, it is because the hum/hiss is obscuring the human voice. Conversely, perhaps if the context equipment testing, human speech can be obscuring the diagnostics. In either situation the subsequent analysis will show it does not matter, simply the direction of thresholding must be flipped. The context chosen here is that the human speech will be the audio to preserve.

2.2 Discrete Fourier Transform

Of these two parts, the former requires performing a Discrete Fourier Transform (DFT) to convert the signal from the time domain to the frequency domain, computing this signal can be done with a Fast Fourier Transform (FFT) algorithm. In this format, identifying the noise in the signal becomes a far more trivial task, as it is evident which aspects of the signal are most prominent, and thus match human speech the least. This process begins by importing the audio files into MATLAB using the **audioread** function, which returns two parameters:

- **y** - A matrix of all the sampled magnitudes of the signal
- **Fs** - The sampling rate

From these two parameters, we can derive all the constants necessary for plotting the signal in both the time and frequency domain. Extracting the length of **y**, gives us with the total number of samples, providing the x-axis on which temporal data is plotted, with each interval between [0-length(**y**)] being one of the samples – the total samples divided by the sampling rate.

Performing an FFT on the input signal is as simple as calling the FFT function in MATLAB and passing in the total samples. An additional step of analysis here, as to generalise further is to analyse each side of the signal separately and superimpose the results, as to account for stereo. Observations here however determined this not to be necessary, as examining the samples showed them to be symmetric (mono), thus we only examine one channel. From here, the magnitudes of each frequency can then be derived by taking the absolute value of each frequency.

An import distinction to be made is the difference between what magnitude represents in each domain. When plotted against time, the magnitude refers to the amplitude of the signal at that given point in time, thus it represents instantaneous sound pressure, or the loudness at any given moment. In the frequency domain, the magnitude corresponds to the amplitude of each frequency component. A higher amplitude here represents a more prominent frequency in the audio, or how much of that frequency there is, hence why the magnitude must be a positive value. Subsequently, plotting the frequencies, we must again take a subset of the total samples; here the number of samples and the sampling rate are both halved as to account for aliasing. The Nyquist theorem states that to avoid aliasing, sampling must be done at a rate slightly above double the highest frequency expected in the audio signal. Plotting this directly would produce a symmetric signal representing the positive

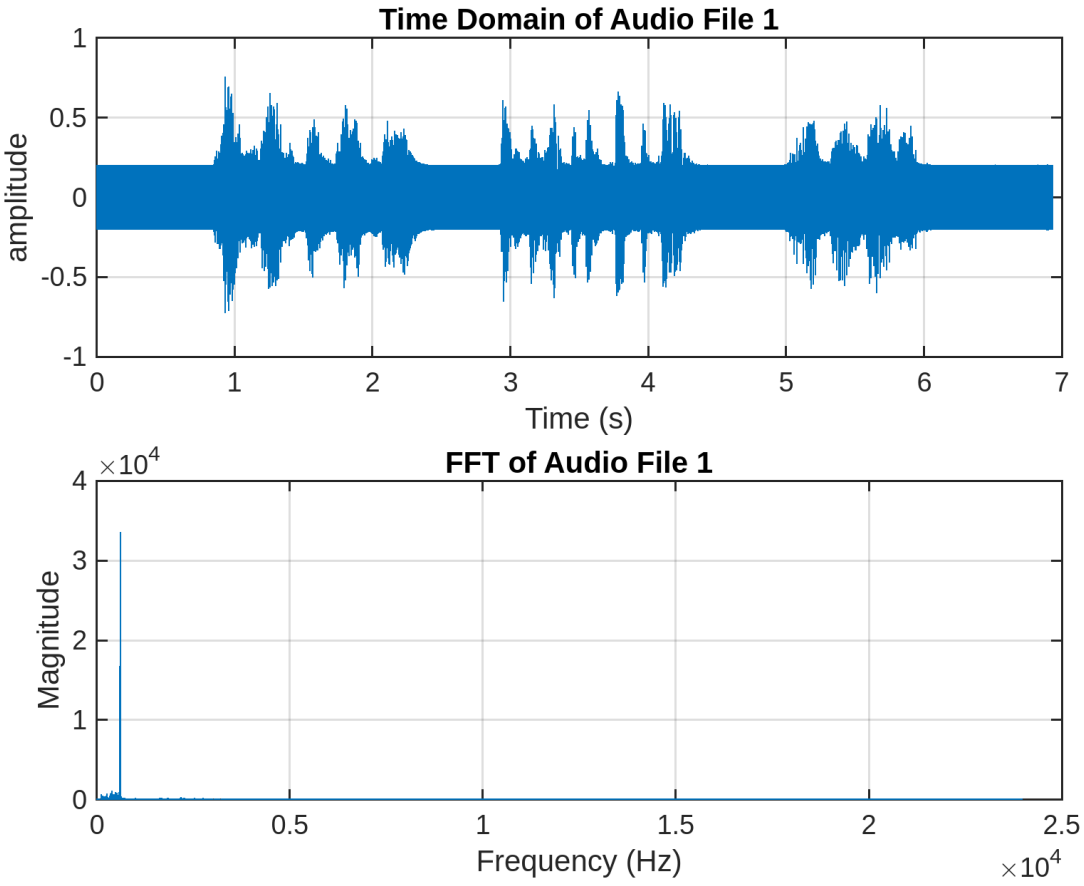


Figure 2.2.1: Audio 1, Time and Frequency Domain

and negative components of the frequencies, and their equal magnitudes. This second half of the frequencies, onwards of **Fs**/2, can be pruned (the amplitudes are also halved to reflect this change), leaving the original audio still intact.

2.3 Removing noise

Figure 2.2.1 shows the side-by-side comparison of *audio_in_noise1.wav* in the time and frequency domain. In both figures, it may seem overwhelmingly clear what the noise is, and so the question must be asked - why is the FFT necessary?

The answer to this becomes just as apparent when attempting to subsequently filter the noise; at no point on the plot are we able to surely say what component of the amplitude is due to noise and what is an intended part of the audio as the two are superimposed with one another, and both exist at the same points in time. Attempting to eliminate the noise regardless, will likely lead to either over-filtering or under-filtering, requiring surplus trial-and-error, over-fitting, and no generalisation possibilities for any future use cases. The latter however, using an FFT, not only allows us to identify the noise, if not in a clearer representation than the time domain, but also allows us to perform frequency filtering based on a certain range of frequencies in the signal.

Deciding on which type of filter to use ultimately came down to a question of versatility and precision; selecting the best filter meant finding one which is most capable of eliminating as little of the total frequency as possible, while simultaneously extracting the most noise. Eventually the notch filter was settled on – this filter provides functionality for isolating and attenuating frequency components in highly specific ranges, leaving the remaining signal intact. Thus filtering with a very low selected bandwidth, made it possible to preserve the full intended audio, as can be seen in Figure 2.2.2.

What can also be seen in the FFT of the audio signal is the distribution of frequencies of both the constant static noise and the natural fluctuations in human speech. The latter takes up a much broader band of the frequency spectrum. causing minor spikes of varying intensity in magnitude across a range of approximately 500Hz. At no point during the audio does the human speech exceed (or get near) 10% of maximum magnitude at any given frequency, thus making it a suitable threshold point for noise removal. Implementing this in MATLAB makes use of many of their built in functions related to filtering, starting with finding the indices of all frequencies in the input matrix which have a magnitude exceeding the threshold. These can then be iterated through, extracting a notch from the frequency spectrum in the range 10Hz below and above the targetted centre frequency. It should be prefaced the reason for iterating is to account for numerous sources of noise (as show in in *audio_in_noise3.wav*, so they can be handled on a case-by-case basis.

As before, the Nyquist frequency must be accounted for, the distinction here is that we divide by this value, as to produce a normalised value of the cut-off frequency between [0,1], putting the notch in a suitable format for MATLAB's **butter**

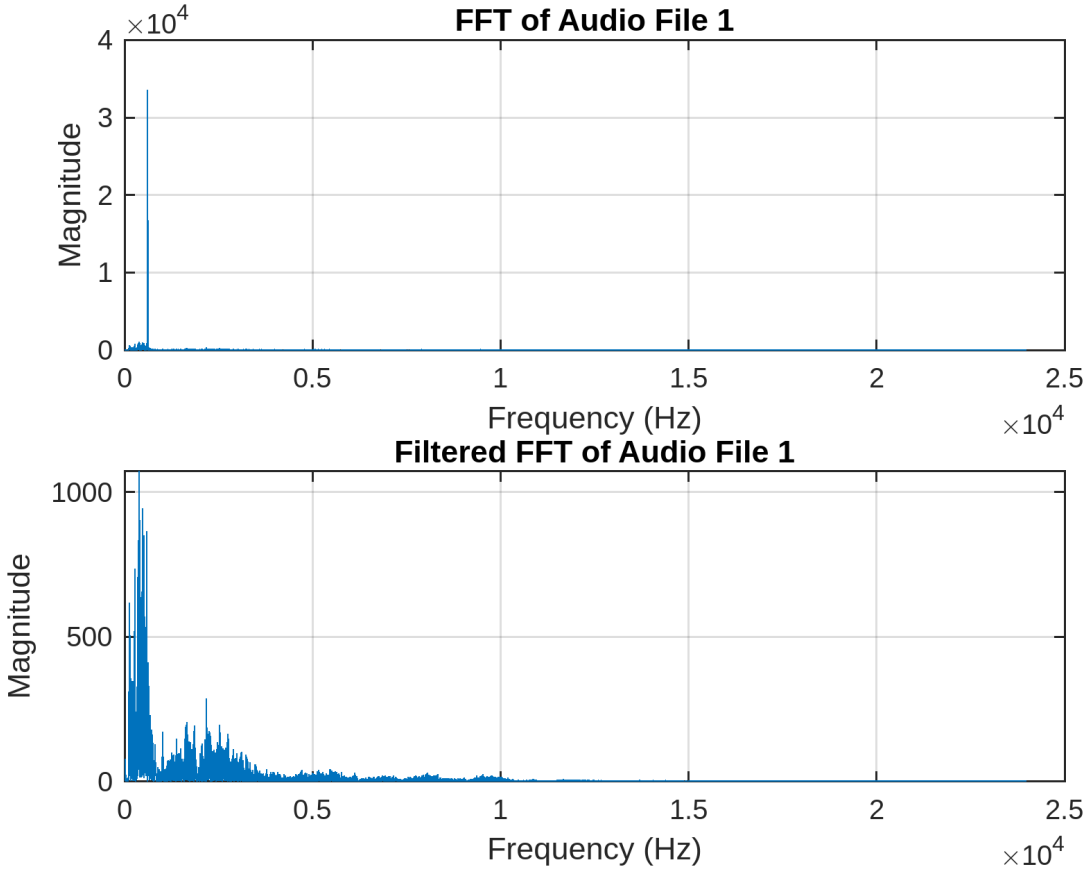


Figure 2.2.2: Audio 1, Frequency Domain Before and After Filtering

filtering tool, which takes a filter order (the smoothness of the filtering), a lower and upper bound, and the type of filter (in our case 'stop' corresponds to notch). The final filtered audio details a far more reasonable distribution of frequencies across the spectra, all of which aligning in the range expected of causal human speech; this range being most concentrated at 90Hz-155Hz for men, with a drop-off for all higher frequencies [1].

2.4 Evaluation

When considering how generalisable this method truly is, the flaw of the model boils down primarily to the specific adjusting which is made to parameters. This would not necessarily be considered over-fitting however, as in all instances in which the noise is a particular set of static frequencies, this model would perform exactly as intended. Where this model falls short is when the noise is not consistent; bouncing between many frequencies throughout the course of the audio will cause one of two scenarios:

- **No noise is attenuated** - In this instance, no particular frequency exceeds the 10% threshold previously established. This threshold could be lowered, but this comes at the risk of over-filtering, and so removing critical aspects of the desired signal.
- **Too much noise is attenuated** - As suggested, the alterative to lower the threshold, depending on the extent of this, could attenuate too much of the speech, as with each notch filter, a minimum of 20Hz of the total spectrum is removed, which is repeated for every instance of noise identified. Thus, for instances of inconsistent noise, this model is at high risk of not performing as intended.

2.5 Results

The full code for parts 1 and 2, audio processing, have been merged into one script - *audio_analysis.m*

Figures 2.5.2 and 2.5.3 detail the plots of remaining two audio files, detailing their FFT and the following filtering (Covering both parts 1 and 2). From these results it is clear that the remaining audio frequencies after the filtering operations appear near identical (aside from zooming out in file 2), allowing us to safely conclude the varying noise has successfully been removed from each of the audio files – these being a 600Hz, a 150Hz frequency, and a 250Hz & 900Hz pair for the respective audio files (in order 1-3). The reason for the files not being exactly identical is due to the behaviour of the notch filter; this filter has been designed to remove frequencies slightly greater/less than the centre frequency, thus depending on where the frequency lies is also where the narrow range of frequencies will be pruned in the spectrum. Listening to each of the files, this proves inconsequential, as they all appear to sound identical, that being human speech, without the static hum (or two)

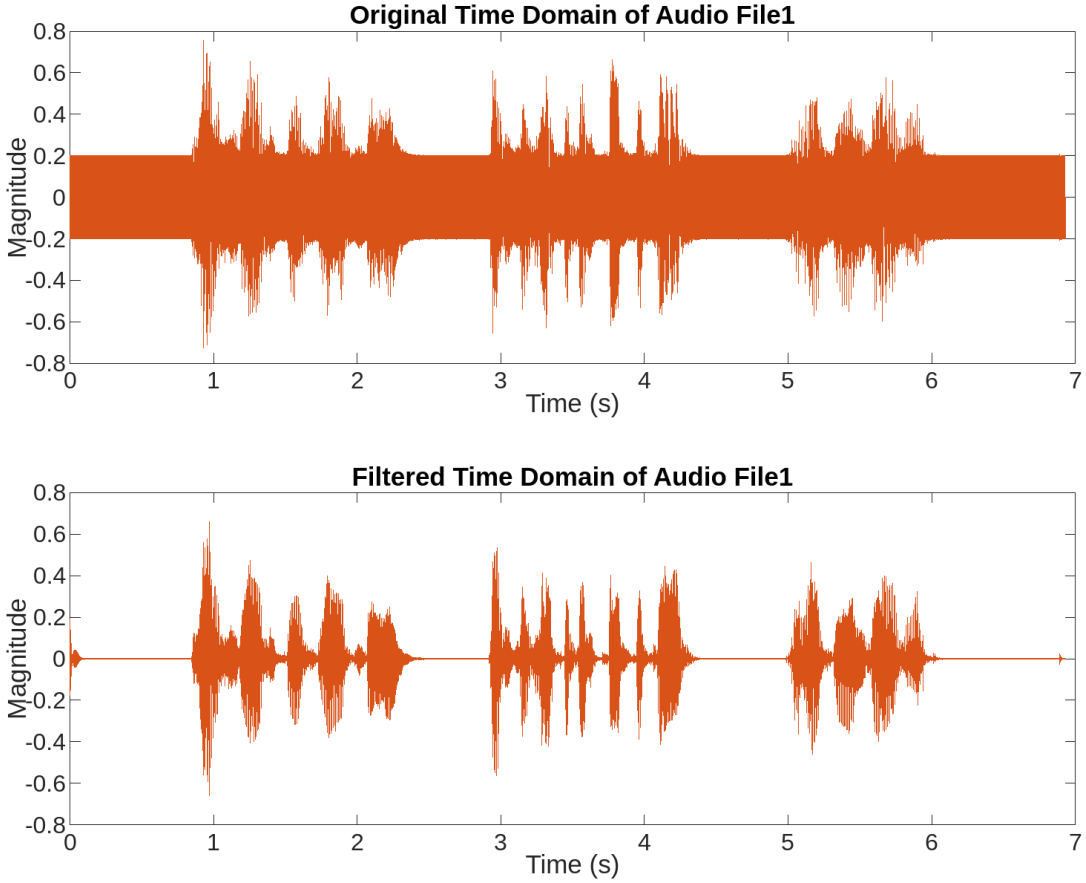


Figure 2.5.1: Audio 1, Time Domain Before and After Filtering

in the background. When can confirm these results further by examining the signal in the time domain, and how closely they match after filtering. As predicted, the three files match in this domain as well, Figure 2.5.1 shows the side-by-side comparison for file 1, the remaining two can be found with all other plots, under *Data/Audio/Plots/*. The filtered audio files can also be found under *Data/Audio/noise_removed<**fileNo**>.wav*.

3 Image Signal Processing

3.1 Design Plan

The latter half of the practical specification detail a hypothetical company's plans to design a software capable of performing object segmentation and classification in images, specifically for the "parts" of DIY projects – screws (of two sizes) and washers.

Supplied with a selection of images ranging in difficulty, each detail a mixture of these objects in varying orientations, with more difficult instances closing the gaps between the objects further. One implementation to tackle the image segmentation involves a "watershed" transformation, or drainage divide, a process whereby the grey-scaled image is converted into a set of minimas, representing the drainage basins. The watershed transformation treats the image it operates upon like a topographic map, with the brightness of each point representing its height, where high intensity denotes peaks and hills while low intensity denotes valleys; The algorithm then floods the image with different colours, starting from the valleys. As the colour spreads, it fills up the catchment basins until it reaches the boundaries of the objects or regions in the image, thus segmenting the image.

3.2 Pre-Processing

In order for the image to be put into a suitable format for image segmentation, it must first be grey-scaled, and filtered down to its most critical components; the further 'simplified' the image greyscale image is, the easier is it for the program to find markers, whereby the valleys of the watershed will begin [4].

1. **Importing** - Similar to the audio files, loading in an image will output a 2D matrix of image data, which can be converted into a normalised greyscale image. Doing so allows us to threshold the images based of the percentage intensity rather than any one specific value.

2. **Contrast Stretching** - Initially, thresholding was performed one-way; in easier difficulties, all screws and washers leaned towards the darker end of the spectrum, this made thresholding the lighter intensities (to 255, or 1 normalised) a

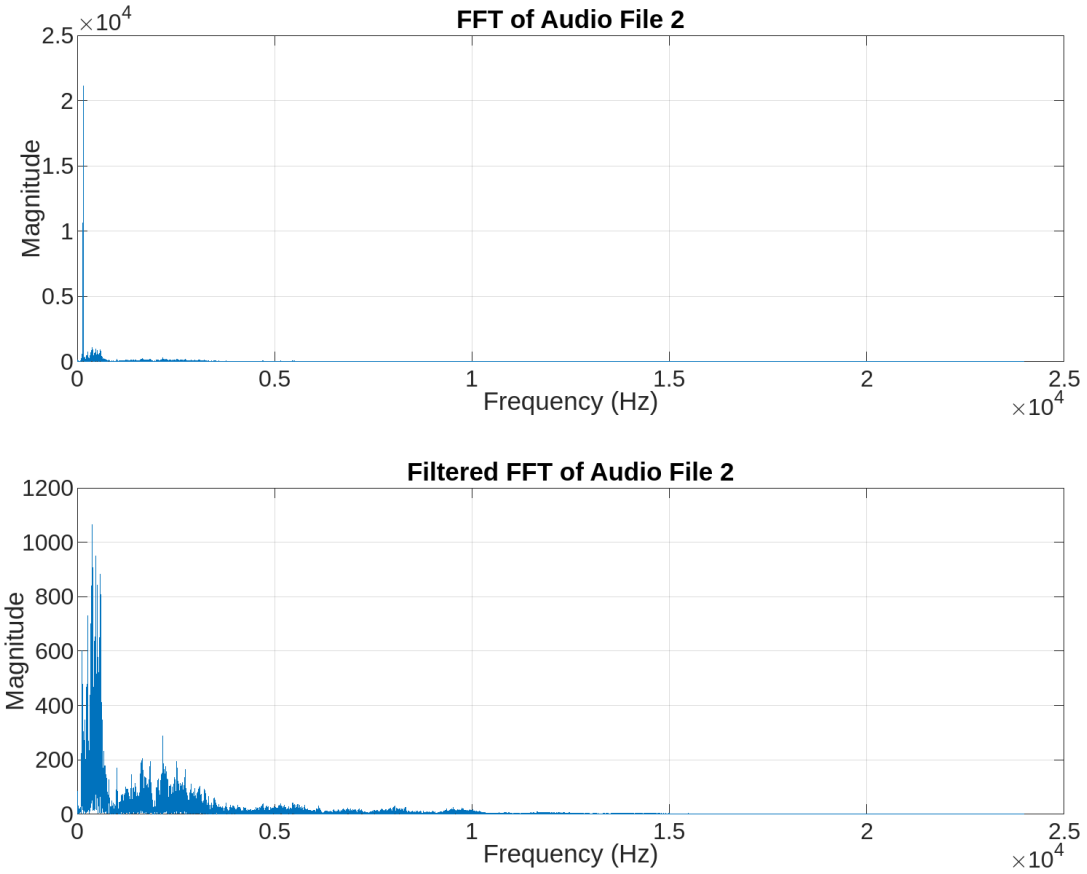


Figure 2.5.2: Audio 2, Frequency Domain Before and After Filtering

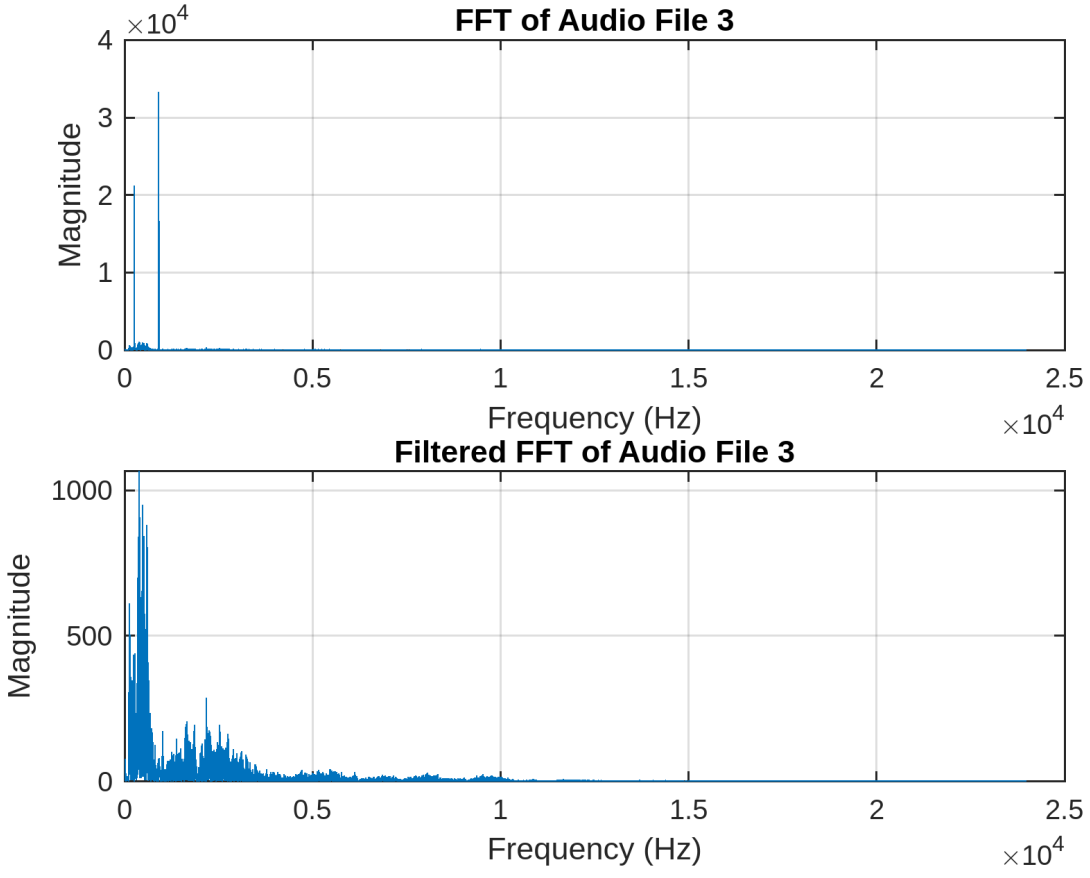


Figure 2.5.3: Audio 3, Frequency Domain Before and After Filtering

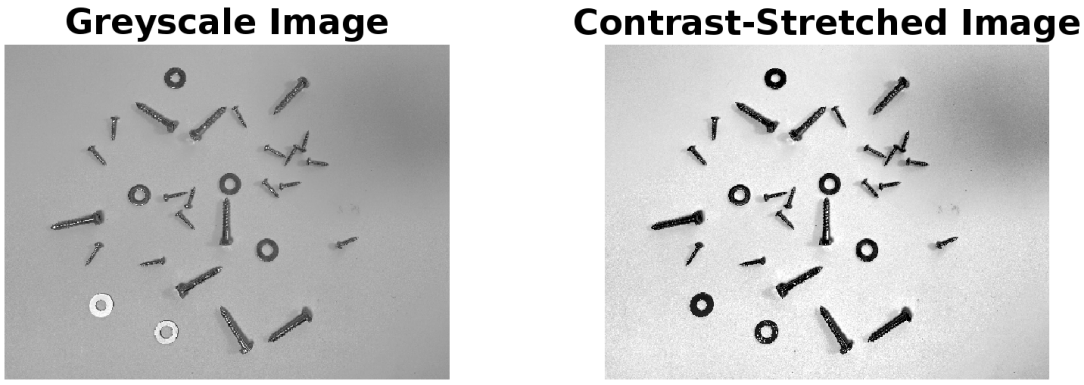


Figure 3.2.1: Hard Image, Greyscale Before and After Contrast-Stretching

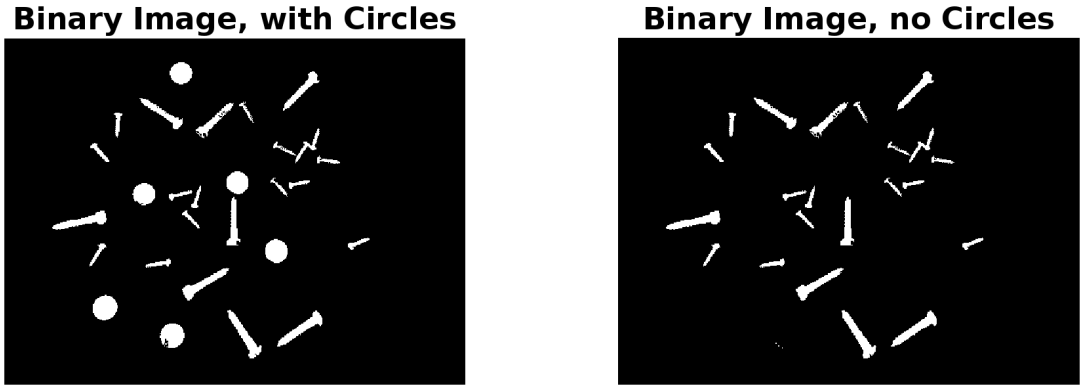


Figure 3.3.1: Hard Image, Washer Segmentation

sufficient process for eliminating shadow and any other noise which could be represented as many smaller valleys in the segmentation. This implementation was quickly aborted when encountering later inputs, containing objects brighter than the background, which were unintentionally omitted. To combat this, a logarithmic scaling of the brightness has been implemented. This takes the all bright portions of the image exceeding the bright threshold (0.84 was deemed sufficient here) and logarithmically scales them down, this value is halved again, placing the object intensity closer to 0 when normalised again. Figure 3.2.1 illustrates this conversion on *Hard.jpg* which contains two washers, initially brighter than the background, before and after contrast-stretching. On the other end of the spectrum, there was the consideration to widen the gap between foreground and background further by scaling up all aspects of the image which fell beyond a threshold, this proved inconsequential as "binarising" the image after negates these components regardless.

3. **Cleaning** - After smoothing out the image, as to avoid any localised pockets low intensity, the image is binarised, setting all values either to 1 or 0 based on a threshold. From here, larger components of the same objects are filled in and connected together, and any remaining specs are filtered out. As will be discussed later, the act 'closing-up' larger objects has a trade-off which cannot be combatted easily, resulting in either over or under-segmentation.

3.3 Washer Segmentation

One aspect of the process which came about towards the end of the implementation to optimise the segmentation (and following classification) was the isolation of washers, separate to the watershed transformation executed on the screws. At this stage of the process, washers are merely white circles on a black background, this makes their identification a far simpler task than that of the screws, and in isolating these, not only do we cover one third of the objects with near-absolute certainty, but it also makes all future watershed processes more likely to succeed, as there are less instances of clashing or merging objects.

Making use of MATLAB's built-in functions for circle detection, a range of radii can be passed which it will search against. This is paired alongside a measure of precision, setting this lower accounts for more imperfect circles, which is a possibility during image cleaning. After locating all circles on the image, they are each processed, setting all pixel values to true (on an initialised grid of all 0s, falses) which are contained within the circle or those which should be apart of the circle, this second condition is based on the pixel's value in relation to the Euclidean distance from the centre of the circle – the pixel must either already be a part of the mask or contained within the Euclidean distance. With the mask established, all circles are stored elsewhere, and segmentation can proceed on the screws.

3.4 Screw Segmentation

Screw segmentation follows a series information-extraction operations, slowly reducing the image to its necessary components, applying watershed transformations throughout until the remaining image is nothing but the separated screws. This process is iterative, it takes the results of previous watershed transformations to influence the subsequent operations. The first of these operations makes use of the distance transformation, an operation which computes the distance from each pixel in the binary image to the nearest pixel in the foreground. This produces a gray scale image, where the lowest intensity are the darkest points of an object, fading out as it reaches the edges of the image. While this alone would seem sufficient for watershed to produce the expected output, it is unable to take into account touching objects, as well as the multiple local minima within the same object, no matter how small; each of these would produce its own "valley", drastically over-segmenting the image. Shown below (Figure 3.4.1) is the side-by-side comparison of the distance transform and the first watershed iteration of *Hard.jpg*. Not only has this produced more valleys than expected, it is also entirely unintelligible, for classification purposes.

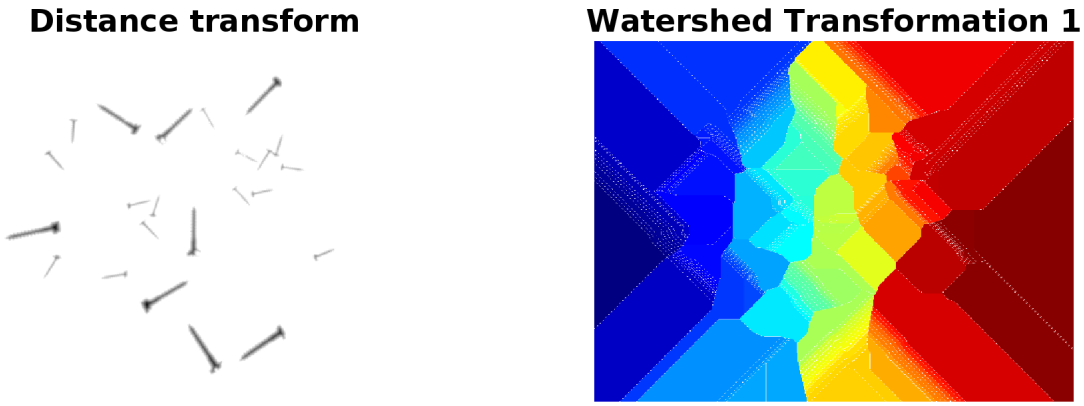


Figure 3.4.1: Hard Image, Distance Transform and Watershed Transform

From here the ridge lines are set to zero (white) and overlayed on the original binary image, this serves no purpose outside of guiding future marker placement, Figure 3.4.3 is a zoomed in extract of the same image, illustrating the over-segmentation. This "raw" watershed transformation (Eddins, 2013) [3], with its tendency to over-segment, can be made suitable, by filtering all but certain minima, for the subsequent watershed transform:

- **imextendmin** - The first operation used, this modifies the original distance transform to suppress all minima below a certain threshold (the higher, the more suppression), effectively "extending" flat regions around prominent local minima. This operation ensures that only prominent valleys, corresponding to meaningful features, such as the end of the screw, remain as markers for the watershed algorithm, while small, noisy minima are removed (commonly being the threads inside the screw). It refines the topographical map by flattening insignificant areas, preventing over-segmentation.
- **imimposemin** - The second operation used, this applies the result of the previous process as a mask on the distance transform, forcing specific pixel locations to act as local minima. By imposing minima at selected points (those derived from markers), the watershed algorithm "anchors" its flooding process to these regions, effectively controlling the segmentation outcome. This prevents the algorithm from creating unwanted regions around spurious or noisy minima and ensures segmentation aligns with the desired markers. Passing this through a watershed transform again gives us a binary image with distinct outlines around each screw, which can be individually analysed for classification purposes.

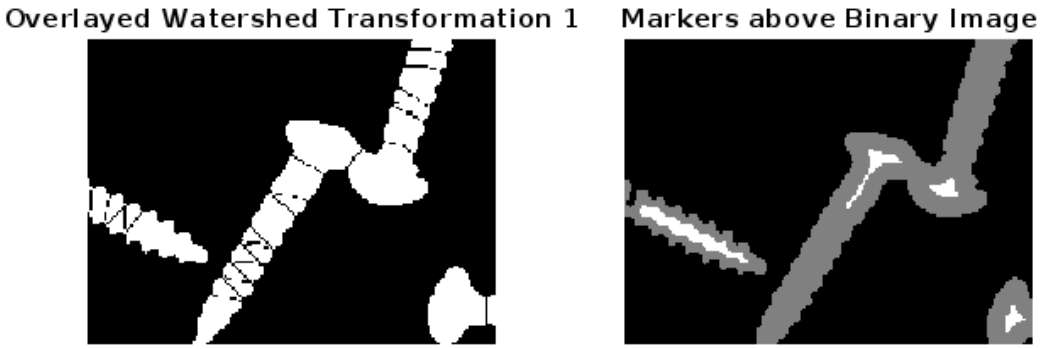


Figure 3.4.2: Hard Image, Overlayed Watershed Transformation and Overlayed Markers

3.5 Classification

The classification of the objects came in two parts, mirroring the segmentation that came before it. As previously stated, the washers has been stored elsewhere, since their identification. On the final overlay of the objects atop the original (greyscale) image, these can simply be iterated through, using the **viscircles** function and the circle attributes to display and border each (the number of washers can also be derived from the size of the array containing them). In regards to the screws, this process also proved simpler than anticipated; as there were no washers to consider, object attributes such as perimeter (to determine circularity), or axis length (to calculate the aspect ratio of each object) were not required. Instead, determining whether the object was a small screw or a large screw was merely a question of area. For the context of the practical these ranges have been hardcoded into the program, with no necessity to change, however they can also be considered as a fraction of the background size. Alternatively, taking the mean of all the screws can be used to determine where the cut-off point is between the small and large screws.

3.6 Evaluation

In the same regard to the audio, when considering how generalisable this is, there are certain pitfalls. Beginning with the pre-processing, contrast stretching proved an effective method for isolating the objects from the background; all six images are thresholded exactly as intended. However, where this could fail is in instances containing a much brighter background, paired with highly reflective objects (such as the metal screws and washers). Due to the camera flash, this pairing of bright objects and bright background would greatly reduce object-background contrast, thus requiring more context-specific tuning of the parameters to perform well.

Another pitfall of this model falls under the washer segmentation, specifically for object placements which reduce the likelihood of identifying a circle, such as a screw on top of the washer. Here the model may be able to isolate the washer, however leaving behind two ends of the screw divided, and depending on the thresholding for what is considered a small screw (by area) this will either count the separated screw as two screws, one, or none, depending on how the screw was oriented.

The final struggle of the model is illustrated as a clear cut-off point in the solutions (being the final two hardest difficulties), a flaw in the watershed segmentation method which proves difficult, if not impossible, of overcoming without rigorous over-fitting for each provided image. Tracing back to as early as pre-processing steps, in order to ready the image to be processed (for the distance transform and subsequent steps) operations such as smoothing, binarising, and contrast stretching all increase the likelihood of close objects (especially those in *Very_Hard* and *Extreme* being considered as one). This is partially circumvented, with the isolation of washers, preventing objects from using the washers as "connection points" to reach other touching objects, similar to a chain reaction. However it does not prevent bunched screwed from being considered as one. The trade-off here would be to minimise these pre-processing steps, however this leads to easier difficulties also failing, due to over-segmentation, and too much detail within the objects themselves.

Some alternative approaches have been thoroughly considered to image processing as a whole, each with their own downsides:

- **Edge Detection** - This would work as a preliminary step to watershed, providing the necessary boundaries for the basins to work appropriately. However, a large pitfall in this model, and my initial implementation (following the steps of Anju 2012 [2]), proved that there was no way to consistently separate the objects as intended. Depending on the method of segmentation, either this would pick up on all noise and consider that an edge (Using *Canny* detection), or pick up on too little, and not identify the boundaries between objects (Using *Sobel* or *Prewitt* detection). Ultimately, no amount of tuning made these work as intended.
- **Machine Learning** - A supervised classification model was considered as the primary contender to watershed-marker segmentation [5]. While this, equip with the correct dataset would surely outperform the current model (given enough hyper-parameter tuning), the pitfall lies within the dataset itself, specifically finding an appropriate one. This dataset would need to conform exactly to our objectives here, that being to identify screws of two different sizes, in relation to one another. This brings the evident question of where a dataset like this could be found, but it also introduces another area of complication, regarding the screw sizes; the model would need to be trained specifically on an image-set which contained both sized screws together, as to differentiate between, as large screws and small screws, are only those sizes in reference to each to other
- **Object Skeletons** - Inspired by Wang et al. (2023) [6], this approach would also be one to replace the watershed transforms. MATLAB offers functionality for extracting the "skeleton" of an object, in the context of screws it would be a line, which when adequately thickened could be recognised as a screw of one of the two sizes. The implementation of this method simply came down to a matter of time, and whether or not further refining watershed would be able to overcome its own challenges.

3.7 Results

The full code for parts 3 and 4, image processing, have been merged into one script - *image_analysis.m*

Figure 3.7.1 illustrates each of the five segmented and classified images overlayed above their original respective greyscale image. These images illustrate both segmentation (part 3) and classification (part 4) in one, as the former is a pre-requisite to the latter, and so only one intermediate step of the second watershed transform will be shown alongside them, for Hard (Figure 3.7.1.d). Though the watershed appears similar to the original binary image (Figure 3.3.1), one significant distinction to be made is the separations between objects, now only placed when necessary.

Using the same model, with no context-specific tuning, the first three images (Easy - Hard) are able to be segmented and classified in full, identifying each individual object correctly as well as producing the valid counts for each of the three objects. Where this model fails to succeed are in the final two images, containing multiple instances of grouped screws being considered as one, or alternatively, one screw being considered as multiple (this latter problem can be solved through tuning values of **imextendmin** however I wished to have one unchanged model for all the images). The following table shows the counts the three objects in each image:

Image Difficulty	Washers	Small Screws	Large Screws	Total Objects
Easy	3	5	4	12
Medium	3	14	4	21
Hard	6	15	8	29
Very Hard	8	19	8	35
Extreme	3	27	16	46

Table 1: Final Counts of Image Classification

It should be noted that though the final two counts (Very Hard and Extreme) appear close to the intended, these do not accurately reflect the segmentation and classification taken place, much of it being due to over-segmentation. One observation made here for the over-segmentation in Extreme is due to the image being slightly more zoomed-in than the rest, impacting the hard-encoded values set for all images to be classified by (based on area).

4 Conclusion

To conclude, this was an interesting investigation into all the various ways auditory and visual signals can be broken down into their necessary components. While there is still room for improvement in the latter half of the practical for harder instances, as a whole, both models perform well, and can do so in a fairly generalised context as well. Reaching these solutions involved much trial and error; MATLAB is built almost entirely through the use of libraries, offering functions tailored to every specific scenario, while this can be seen as a good thing, it means no knowledge in MATLAB can be transferable, it also means that there is a high likelihood some crucial library will be missed based on pure chance. Given more time I would have liked to explore the use of object skeletons to segment, as I believe it would have performed better than watershed, without the need for a dataset, like machine learning.

References

[1] R. J. Baken. *Clinical Measurement of Speech and Voice*. 2nd ed. Taylor and Francis Ltd., 2000, p. 177. ISBN: 1-5659-3869-0.

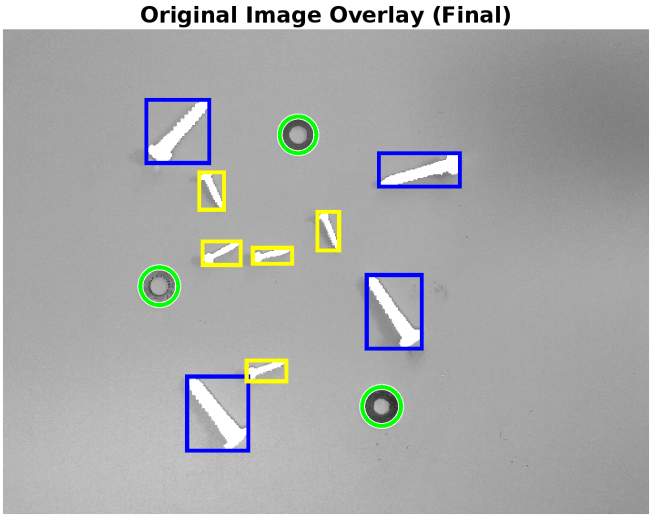
[2] Anju Bala. "An Improved Watershed Image Segmentation Technique Using MATLAB". In: *International Journal of Scientific & Engineering Research* 3.6 (2012), pp. 1–5. URL: https://www.ijser.org/researchpaper/An-Improved-Watershed-Image-Segmentation-Technique-using-MATLAB.pdf.

[3] Steve Eddins. *Image processing concepts, algorithms, and MATLAB*. 2013. URL: https://blogs.mathworks.com/steve/2013/11/19/watershed-transform-question-from-tech-support/.

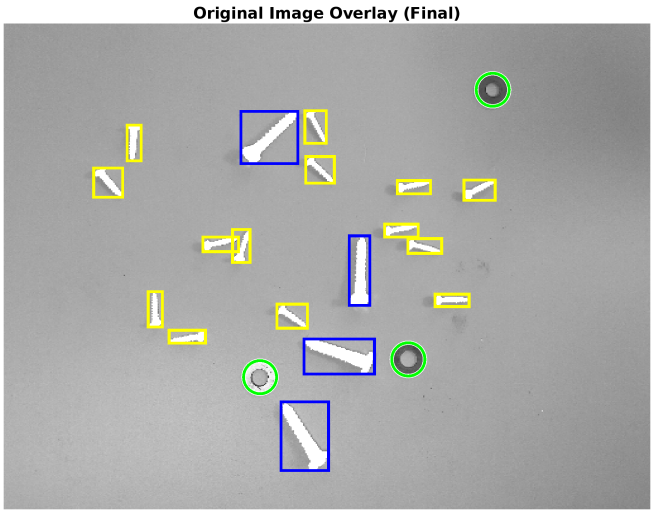
[4] Ranjan Mondal, Moni Shankar Dey, and Bhabatosh Chanda. "Image Restoration by Learning Morphological Opening-Closing Network". In: *Mathematical Morphology - Theory and Applications* 4.1 (2020), pp. 87–107. DOI: 10.1515/mathm-2020-0103. URL: https://www.degruyter.com/document/doi/10.1515/mathm-2020-0103/html.

[5] Johanna Pingel. *Semantic Segmentation for Medical Imaging*. Accessed: 2024-11-29. 2021. URL: https://blogs.mathworks.com/deep-learning/2021/05/10/semantic-segmentation-for-medical-imaging/.

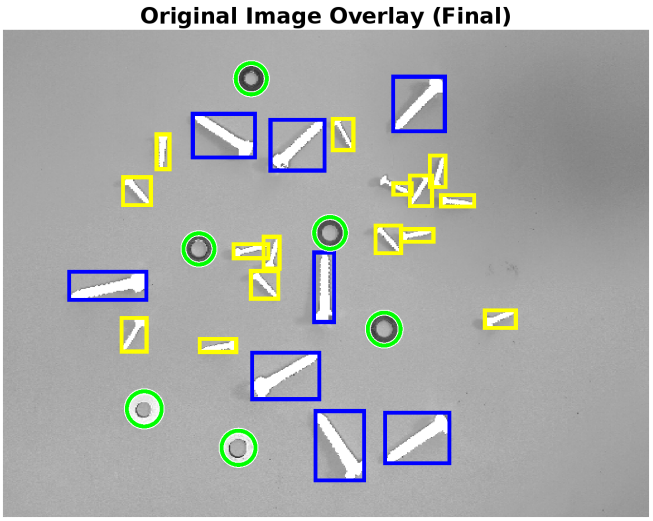
[6] T. Wang et al. "Structure-Preserving Instance Segmentation via Skeleton-Aware Distance Transform". In: *arXiv preprint arXiv:2310.05262* (2023). URL: https://arxiv.org/abs/2310.05262.



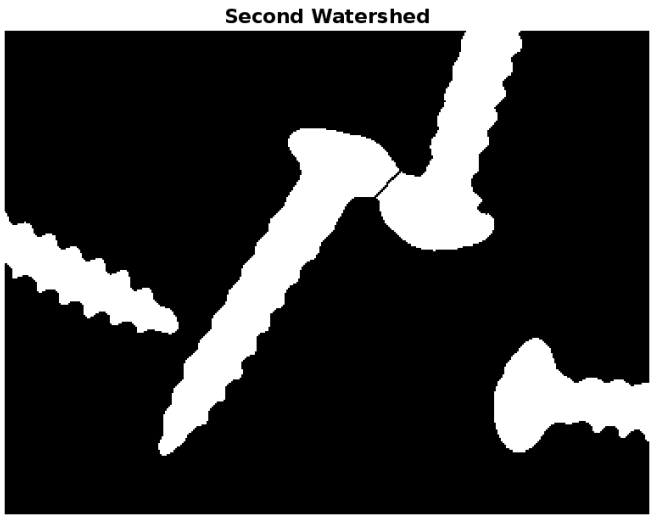
(a) Easy



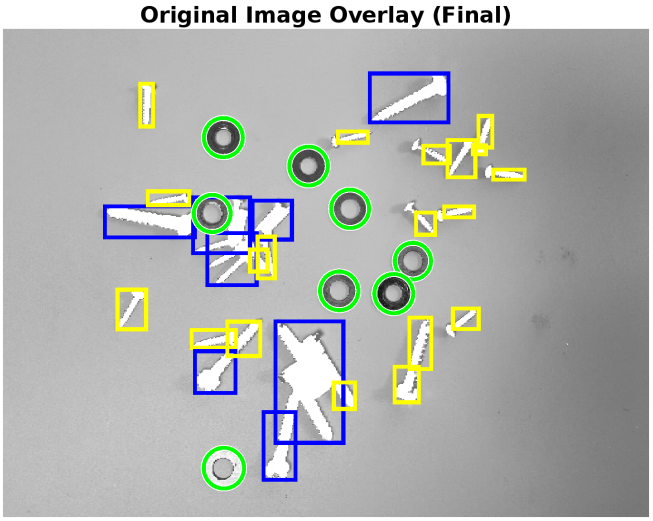
(b) Medium



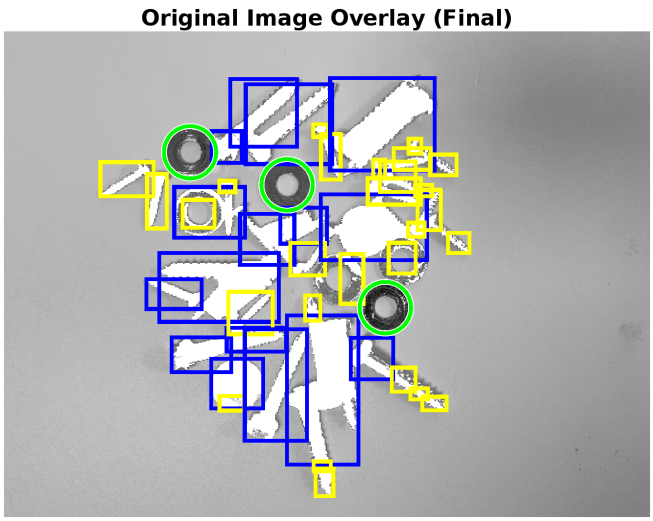
(c) Hard



(d) Hard, Second Watershed (Zoomed)



(e) Very Hard



(f) Extreme

Figure 3.7.1: All Classified Objects Overlayed on the Original Images