Ngee Ann Polytechnic

Engineering Science

93ESFYP Final Report

Year 2018

Project number : PS09

Condition Monitoring of Railway Tracks with Soft and Stretchable Sensors

By Darren Chan Yu Hao [S10173972E] and Yeoh Guan Wei [S10160483H]

Supervisors:

Dr. Koh Soo Jin Adrian (External Supervisor)

Dr. Koh Chan Ghee (External Supervisor)

[National University of Singapore (NUS)]

and

Dr. Jaslyn Law Bee Khuan (Internal Supervisor)

Mr. Koh Fook Leong (Internal Supervisor)

[Ngee Ann Polytechnic (NP)]

Abstract

This report is a continuation of the “Condition Monitoring of Railway Tracks with Soft and Stretchable Sensors” EGDGN report, which showcases the design and working principles of an automated system that inspects the conditions of the power rails of the Sengkang-Punggol Light Rail Transit (LRT) system. The main objective is to integrate the stretchable sensors and the Radio Frequency Identification (RFID) technology together to detect faults on the LRT tracks, and also to find out if any condition, for instance, the speed, of the LRT will affect the results obtained. To obtain the said objective, studies on the LRT system and the train were made and the method to utilize and combine results from the stretchable sensor and the RFID technology was implemented.

Calculations on the mechanics of the LRT train and system were made and a MATLAB program that takes in results from the stretchable sensor and RFID technology to detect faults on the LRT track and its potential location was created. This report will touch on the calculations and considerations on the LRT mechanics, as well as the process and details of making the MATLAB program.

Table of Contents

1. Introduction

The Light Rail Transit (LRT) system has been running in Singapore for around 20 years. The average ridership for the Sengkang-Punggol LRT is around 121,000 in early 2018, which is a 5.9% increase from 2017. [1] Thus, it is vital that the LRT is able to function properly and reliably. Unfortunately, due to its long service, the LRT has been experiencing multiple breakdowns in the recent years. For instance, in 2015, the Sengkang LRT experienced 5 major breakdowns of over 30 minutes. [2] Earlier this year, there was also a breakdown on Sengkang West Loop on the first day of Chinese New Year on 16th Feb 2018. [3] With the LRT being so problematic, the public has become increasingly vocal about its unreliability. If upgrades are not performed on it, it will continue to be a faulty and unreliable transportation that will bring inconvenience to its users and problems to the transportation companies.

One way that the LRT can break down is a power fault. This happens when the spring-loaded current collector device is disconnected from the power rail due to various reasons such as a dent too deep for the current collector device to connect to or there is a misaligned rail.

A way to solve this is by enhancing the rail monitoring system. The current method employed to monitor the power rails is through manual inspection of the tracks and visually identifying any damaged areas. The problem with this is that inspection can only be done when the train is not in service. Additionally, it is not only time-consuming but also labor intensive due to the immense size of the railway track, roughly around 10.2km. Next, it is also expensive for the transportation companies. Lastly, these inspections might not be accurate or precise enough as workers might miss a fault and that might allow it to escalate into a major problem that can cause a breakdown. All these play into the narrative that the current method is unreliable and outdated. Therefore, this project will propose a new and updated process to monitor the railway, a structural health monitoring system.

The structural health monitoring system will utilize stretchable dielectric elastomer capacitive sensors to detect changes in the structure, in this case, the power rail. This will mean that the identification of faults will be left up to precise sensors that can detect faults in the millimeters range. This is superior to the visual inspection used presently by being more precise and accurate. Additionally, this method does not require as much manpower or time, thereby reducing the operating cost over time. It has also been tested extensively to ensure its reliability.

This stretchable sensor will be mounted on the spring-loaded current collector device which is connected to the power rail. This will allow it to measure any unexpected displacement through a change in capacitance which can then be categorised into different forms of faults such as a consistent wear or a misalignment of tracks.

Alongside the stretchable sensors, Radio Frequency Identification (RFID) technology is used. Although the stretchable sensor is able to identify faults, it is unable to locate the fault, thus a positioning system will need to be used. After research on multiple positioning systems, the RFID was chosen due to its high precision in a localised area. It will be used to detect where the LRT train is, at any given point in time and using that data, it will be related to the stretchable sensor signals and identify the point of the fault. The RFID consists of several components such as the reader, antenna, and tags. To control it, a Raspberry PI is used.

The RFID has been proposed to be mounted under the train and tags will be running across the track. As the LRT moves along the track, the tags will be read and recorded, sending information on the location of the train. With this, the location of the fault will be known and maintenance can be carried out to rectify the problem.

With the cost of the RFID being semi-low and requiring little modification to the LRT system, it is ideal for practical usage in the real world. For this project, the end product should be a real-world ready structural health monitoring system that can automatically detect and read faults on the LRT track even for users without any expertise or prior experience in track maintenance or fault identification.

2. Project Outline & Objective

As this is a continuation of the project “Condition Monitoring of Railway Tracks with Soft and Stretchable Sensors”, the overall objective remains to be designing and creating of an automated system that detects and records possible faults on the LRT power rails using the stretchable sensor. The project can be broken down into two parts: designing and using the sensors to obtain the objective and processing the data acquired from the sensors and making sense of it.

For this period of the project, the objective will be to focus on the signals acquired from the stretchable sensors and analysing as well as utilizing these signals. To elaborate, the authors will denoise the signals obtained and also try to find the best methods of denoising the signals. Additionally, the authors will work on combining both the stretchsense data with the RFID data to obtain an automated detection of faults using a simple UI. Lastly, some calculations on the forces on the LRT during turns and how it will affect the CCD was done. The figure below shows the Gantt chart that acts as the timeline for our project.

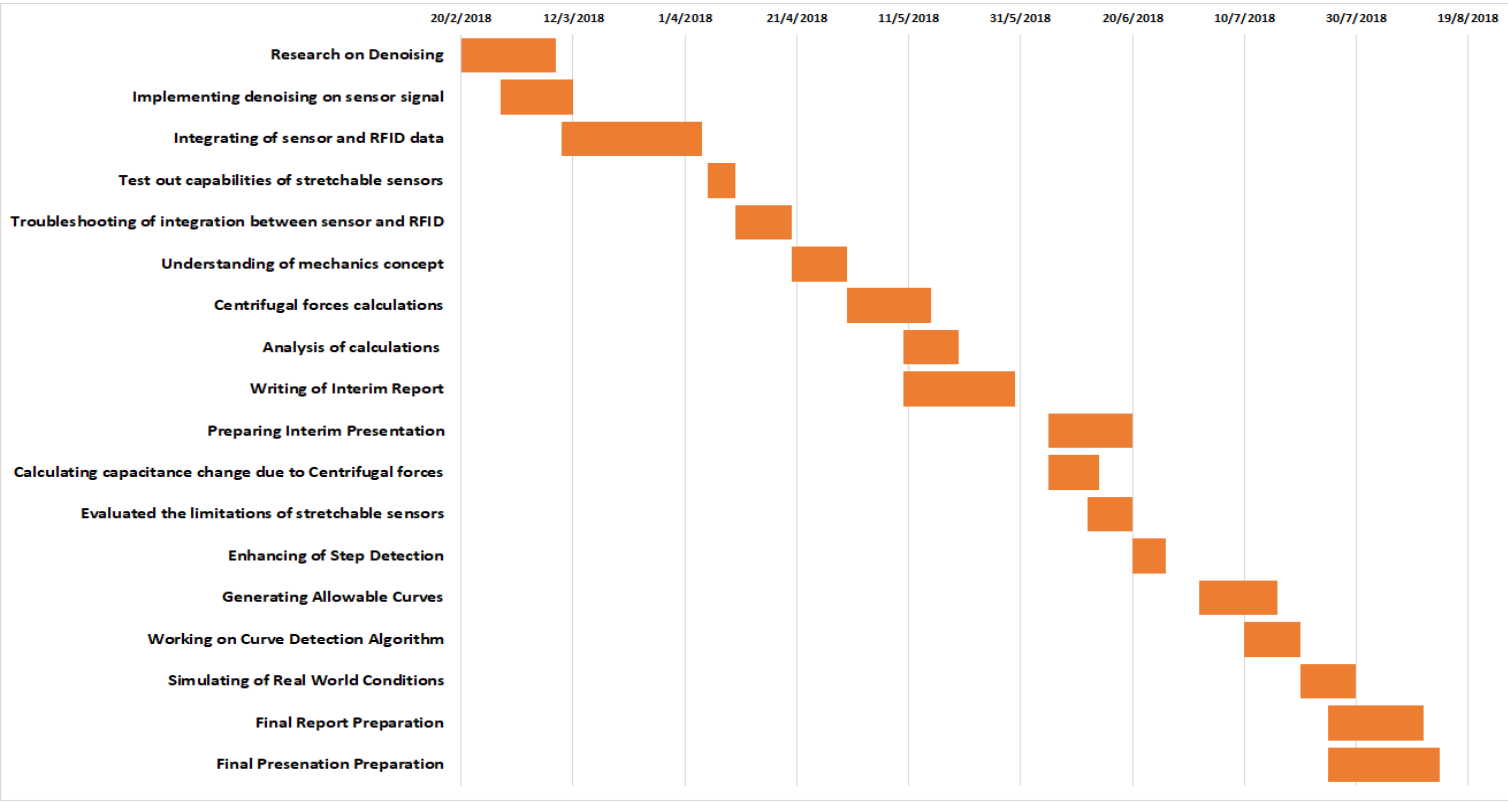


Figure 2. Gantt chart as timeline for project

3. Literature Review

3.1 Theory on Stretchable sensor & Radio Frequency Identification

A dielectric elastomer consists of a thin, incompressible elastomer, sandwiched between two compliant electrodes. It is essentially a capacitor which stores capacitance in an electric field. When the dielectric elastomer stretches, its surface area will increase while the distance between the two electrodes will decrease. When the surface area, A increases and the distance, d decreases, the capacitance will increase. Hence the change in capacitance is related to the strain of the dielectric elastomer.

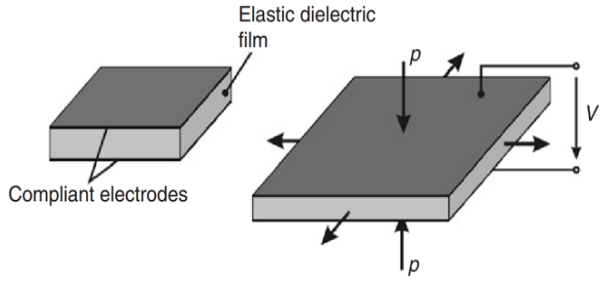


Figure 3.1. Dielectric Elastomer when stretched

Radio Frequency Identification (RFID) technology mainly consists of a reader, antenna and multiple identification tags. The reader and antenna will work together to read the tags. This is done by converting electrical current from the antenna into electromagnetic waves. These will then be received by the tags passing through the scanning field of the antenna and reflected back to signify a detection. Below is a simplified illustration.

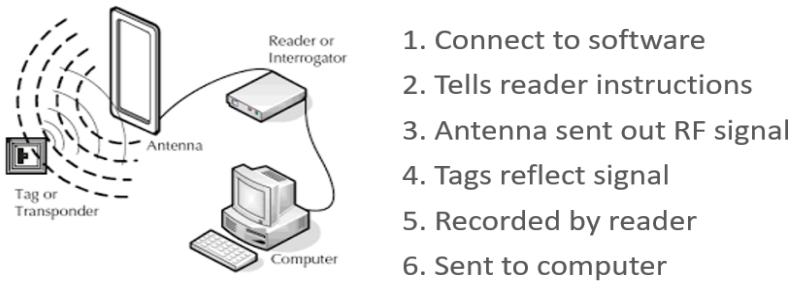


Figure 3.2. RFID step-by-step procedure

3.2 Signal Processing - Fast Fourier Transform (FFT)

Any digital waveform can be represented by a series of sinusoidal waveforms with varying frequency, amplitude, and phase. Fast Fourier transform decomposes a waveform into sinusoidal waves of different frequencies and these waves, when put together, will form back the original signal. [4] The Fast Fourier transform is more efficient than the Discrete Fourier Transform in terms of computing time.

Together with Fourier transform, filters will need to be used to filter out noises in signals. As the signal from the Stretchable sensor has noises that are low in amplitude and high in frequency, a low-pass filter can be used to filter out the noises. The choice of the type of filter is also important as it determines the resolution in time and also the shape of the signal.

The Fourier Transform has its limitations, which is related to the Heisenberg's uncertainty principle also known as the resolution of measurement. The principle states that the position and the velocity of an object cannot both be measured exactly, at the same time, even in theory. Fourier transformation uses sine and cosine waves, which are infinite in time and thus will result in a loss in resolution in time as only the range of frequencies would be found when given a range of time. [5] Hence to solve this problem, Wavelet transform can be explored and used for the signals.

3.3.1 Signal Processing - Discrete Wavelet Transform (DWT)

Discrete Wavelet Transform (DWT) is the usage of wavelet to discreetly sample a signal, possibly to remove noise or other unwanted results. To do this, it uses little "mini-wave" to sample the signal such as the one used in the testing in the experimental section, sym8.

On the other hand, as mentioned, Fast Fourier Transform (FFT) uses sine and cosine waves to transform the signal into the frequency domain. As sine and cosine waves are infinite in time, resolution is lost. DWT and other forms of wavelet transform do not have this issue due to those "mini-wave" mentioned. They are finite in time, thus allowing the sampled signal to have better resolution in frequency and time. One thing to note however is that DWT transforms the signal into the wavelet domain instead of the frequency domain. Nonetheless, because of the way the transform is done, DWT is

more advantageous than FFT because it will be able to provide a temporal resolution which can then allow both the frequency and location of the signal to be kept.

3.3.2 Theory of Discrete Wavelet Transform

To use Discrete Wavelet Transform, its working principle will have to be explored. Wavelets, like Sine and Cosine waves, have high and low frequencies and can get squished together for high frequencies and stretched out for lower frequencies as can be seen in the diagram below. [6]

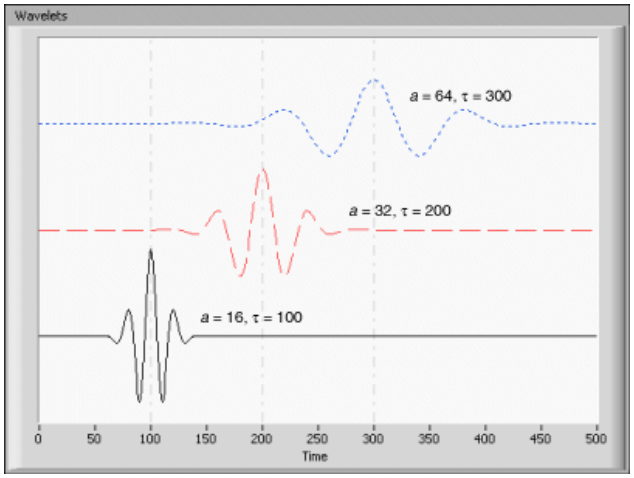


Figure 3.3. Different frequencies of wavelet

The diagram below also shows the different types of wavelet.

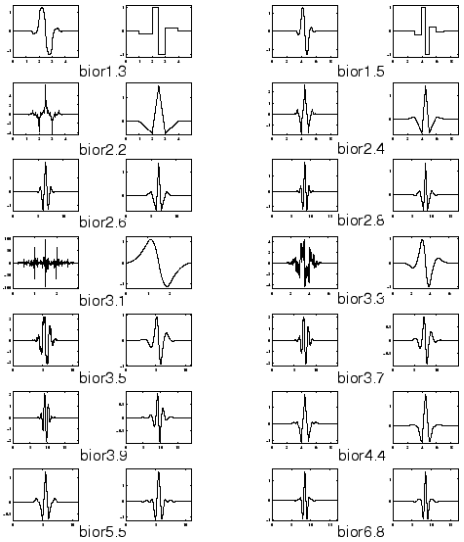


Figure 3.4 Different types of wavelet

These wavelets are used to fit the signal and compared to obtain a cleaner signal or rather denoised signal. Here are the steps to go about denoising using wavelet transform. [7]

Step 1, is to decompose the wave. As mentioned it will first choose a wavelet and the level of decomposition N. This level of decomposition can be determined by the number of samples obtained using the formula 2^N = Number of sample or rather $N = \frac{log(noOfSamples)}{log(2)}$. With the level of decomposition, it will proceed to decompose. During this decomposition, the signal will be passed through both low pass and high pass filters.

Step 2, the threshold level would have to be selected. For Matlab there are two different types of thresholding, soft thresholding and hard thresholding. For hard thresholding, the element absolute value is compared with the threshold value and if it is lower, it will be set to 0 as seen in the diagram below. The signal to be thresholded is x and thus x is x if |x| > t, and is 0 if |x| ≤ t. Next, there is the soft thresholding which actually shrinks the signal if |x| > t. It basically minus away the threshold value or (x)(|x| - t). For this project, soft thresholding is chosen as it will allow the signal to be more smooth.

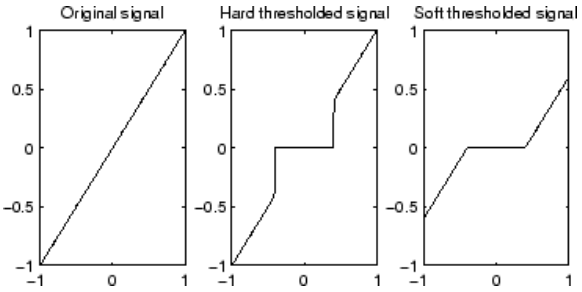


Figure 3.5. Soft and Hard Thresholding

Additionally, Matlab also consists of different type of threshold selection rules namely, ‘sqtwolog', ‘rigrsure', 'heursure' and 'minimaxi'. According to Matlab’s user manual, for ‘sqtwolog’ and ‘heursure’, they are more efficient in removing noise while ‘rigrsure’ and ‘minimaxi’ are more conservative and is more for removing detailed noise that are closer to the actual signal itself. To test the validity of this, testings which will be shown later were made.

Step 3 is to reconstruct the signal. It is to compute wavelet reconstruction using the original approximation coefficients of level N and the modified detail coefficients of levels from 1 to N. This basically means to put everything back together to obtain the original signals but with the noise and unwanted signal removed.

4. Experimental Results & Analysis

4.1 Digital Signal Processing

To get any sort of meaningful results out of the setup, the raw signal from Stretchsense sensors and UHF RFID will need to be converted to readable results that can be easily interpreted. To do this, noises from the signal data acquired by the Stretchable Sensors will firstly and most importantly have to be removed. Fourier transform and Discrete Wavelet transform will be utilised to remove the noises in MATLAB.

The noises in the signal from the Stretchable sensors comes from mainly 3 forms, vibration, electrical noises as well as roughness noise. Below shows a few different test runs done by the previous batch. As can be seen, noise is very prevalent in the data, making it almost impossible to make out any meaning from the data.

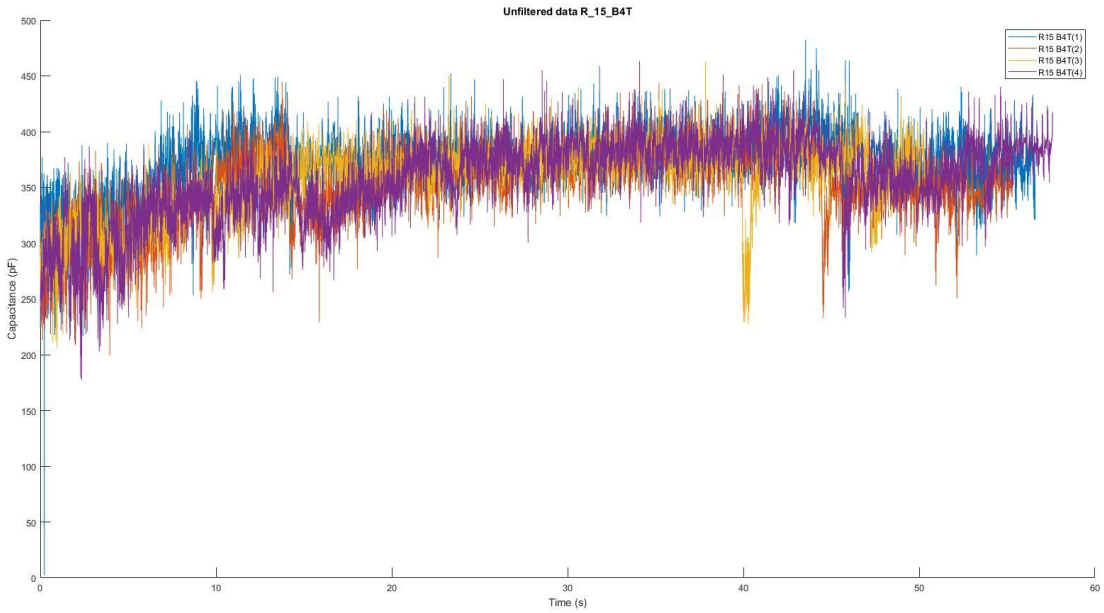


Figure 4.1. Unfiltered signal of sensor

To remove these noises, either Fast Fourier Transform (FFT) or Discrete Wavelet Transform (DWT) can be used in MATLAB. Hence, research was done online for how to code these two transformations in MATLAB. The expected result is a clean signal.

4.1.1 Fast Fourier Transform (FFT) in MATLAB

Starting with FFT as the authors want a simpler approach to understand how signal processing work and at the same time get familiar with the MATLAB program. The code and explanation for FFT are given below. [8]

`Y = fft(X)` computes the discrete Fourier transform (DFT) of `X` using a fast Fourier transform (FFT) algorithm.

- If `X` is a vector, then `fft(X)` returns the Fourier transform of the vector.
- If `X` is a matrix, then `fft(X)` treats the columns of `X` as vectors and returns the Fourier transform of each column.
- If `X` is a multidimensional array, then `fft(X)` treats the values along the first array dimension whose size does not equal 1 as vectors and returns the Fourier transform of each vector.

Figure 4.2. Fast Fourier Transform function in MATLAB

Since FFT is being used, there will be a need to choose a suitable filter to go along with it. There are four types of filter that were found, Butterworth, Chebyshev Type 1 and 2 and Elliptic Filter. Chebyshev 2 and Elliptic filter will have a higher roll-off rate than the Butterworth and Chebyshev 1 filter, meaning it will have a higher attenuation at the cut off frequency and act more like an ideal filter. However, a downside to Chebyshev 1 and the Elliptic filter is that it has ripples in the passband, and Chebyshev 2 and Elliptic filters have stopband ripples which means that the signal passed through it might have ripples too and that is not ideal for this application. Ultimately the Butterworth filter was chosen as it has no passband or stopband ripples, allowing a smoother transition. [9]

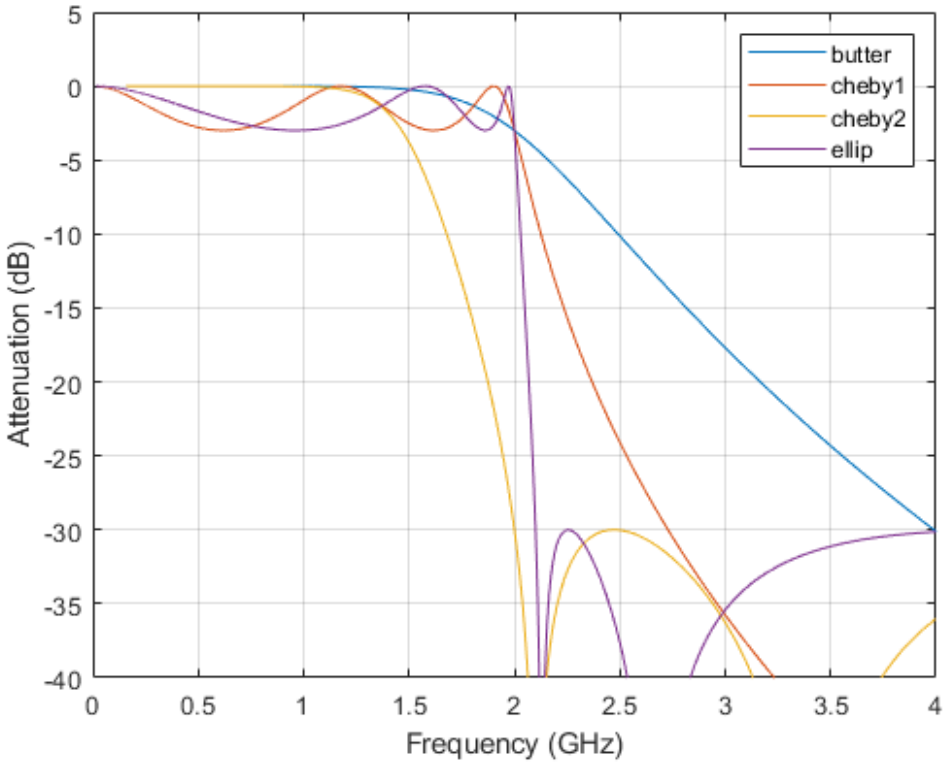


Figure 4.3. Gain magnitude vs frequency response graph

The low pass filter was selected for the signal to remove the noise as noises are in high frequency and the main signal for the sensor is low in frequency and hence a low pass filter will filter out the noises.

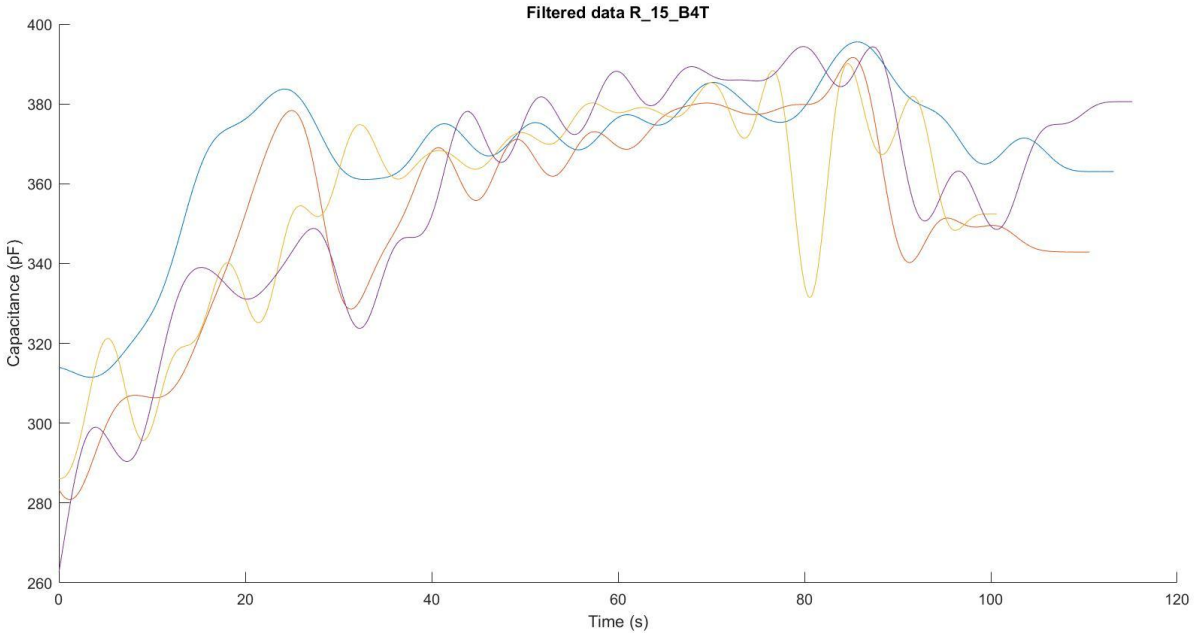
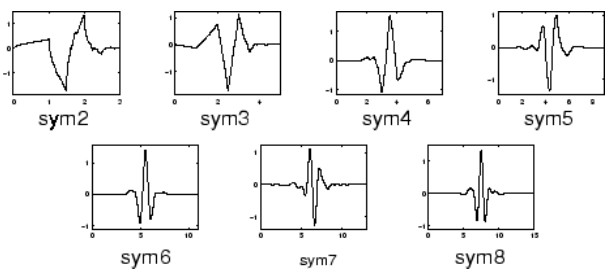


Figure 4.4. Filtered signal with FFT

As can be seen from the picture above, the noises are removed and the general shape of the signals can be seen clearly and meaningful results can now be obtained from it.

4.1.2 Discrete Wavelet Transform (DWT) in MATLAB

With the FFT denoising done, the Discrete Wavelet Transform (DWT) was explored to compare the two methods and see which is more ideal and useful for the signals and to see if the theory is correct. DWT requires a little more selection compared to the FFT. Firstly, the shape of the wavelet has to be chosen, followed by the level of decomposition, type of threshold and more. Research, as well as testings, were carried out, and a wavelet shape of ‘sym8’, ‘sqtwolog’ which is the universal threshold and ‘mln’ which is rescaling using a level dependent estimation of level noise was chosen. [6]




```
dirDec = 'c';          % Direction of decomposition
%level  = 7;           % Level of decomposition

level=log(noOfSamples)/log(2);
level=floor(level);% round N up

wname  = 'sym8';    % Near symmetric wavelet db1, sym
                    %https://www.mathworks.com/help
decCol = mdwtdec(dirDec,y_capacitance,level,wname); %
[XD,decDEN] = mswden('den',decCol,'sqtwolog','mln');%
```

Figure 4.5. Different types of sym wavelets (Left), Sample code for DWT (Right)

The same set of data was run through the DWT code and the results are shown below.

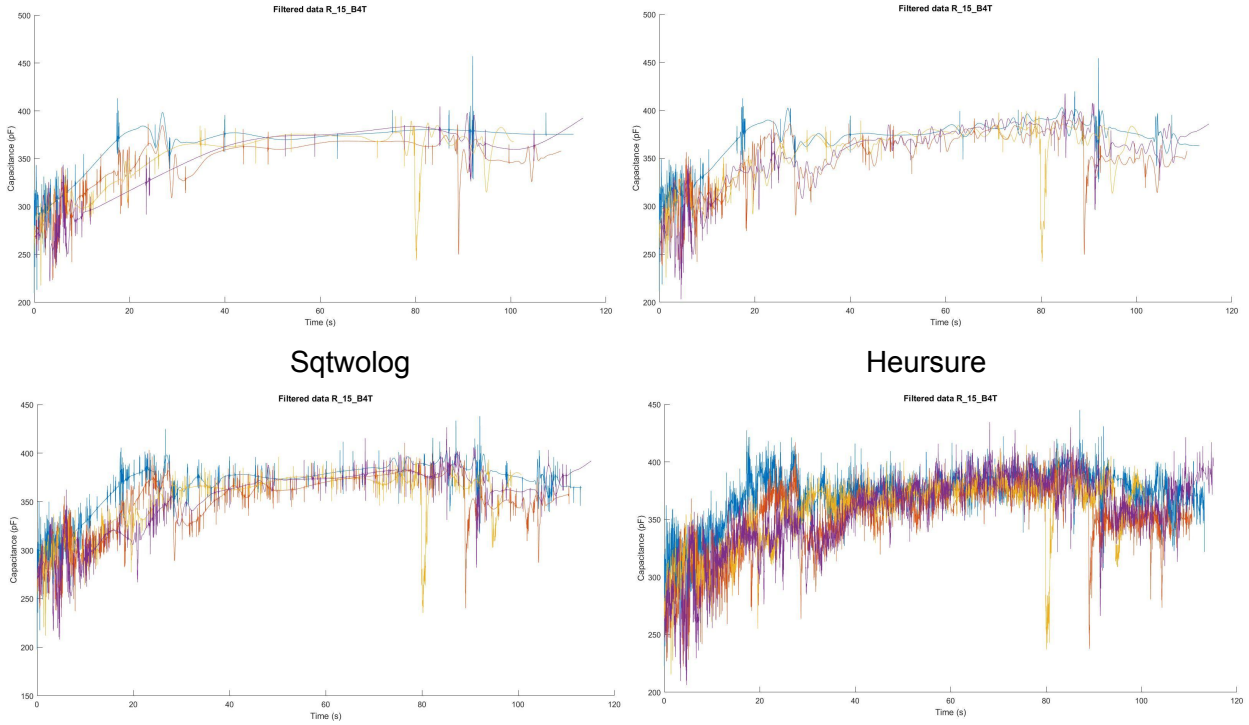


Figure 4.6. Results of different thresholding methods

As can be seen, the ‘sqtwolog’ threshold is the best at denoising out of the four and most of the noise has been removed, except for the first 10 seconds and a few spikes that still remain unremoved. From the look of the two denoising methods, it seems like FFT is a better choice than the DWT at denoising. However a single test is not sufficient to derive any conclusion, thus testings with another set of data were made to see if the two denoising methods will make any difference to the data obtained from different situations.

A set of data that simulated a ‘step’ where there is a misalignment between tracks, causing a sudden spike and elevation of capacitance value was run. The picture below is the raw signal from the simulation, and as can be seen, the noise is very prevalent.

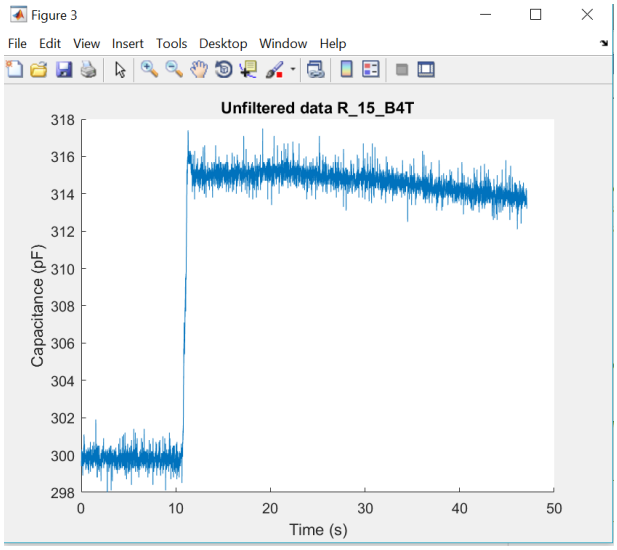


Figure 4.7. Unfiltered signal of simulated 'step'

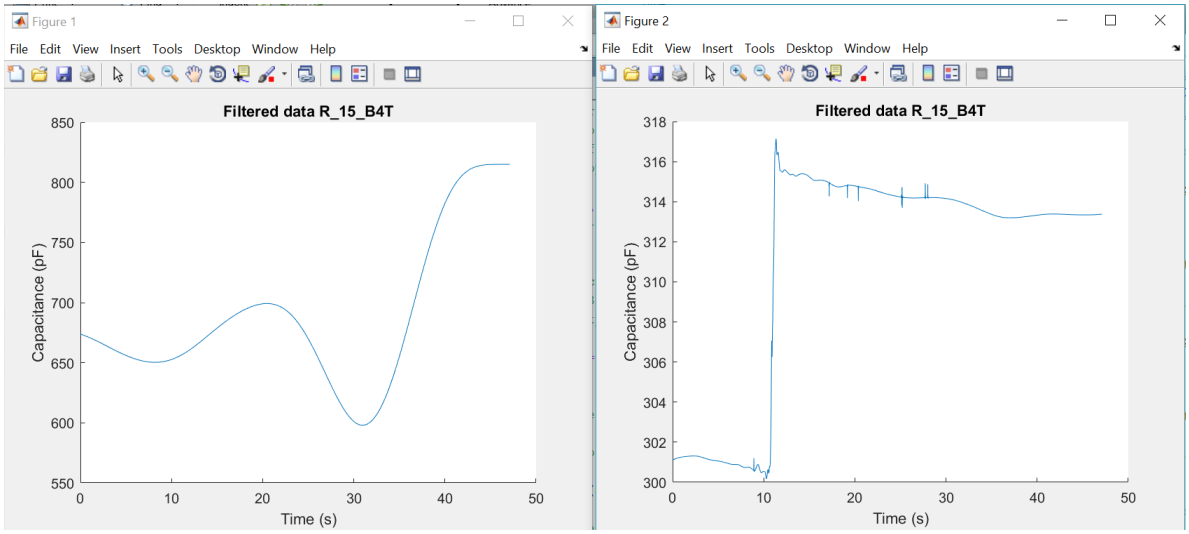


Figure 4.8. Filtered data of simulated 'step' using FFT (Left) and DWT (Right)

After running the data through both methods, it can be seen that FFT has caused the signal to lose its shape and even the overall capacitance range after denoising while DWT retained the shape of the signal. This shows that FFT might not be ideal for this application as it might cause information to be lost especially when there is a step as Butterworth filter may turn a sudden spike in value into a smooth gradual rise. In conclusion, DWT was selected for this program and signals.

4.2 Integration of Stretchable Sensors with UHF RFID

With the data refined and readable, the next step would be to combined the signal from the stretchable sensors with the UHF RFID. This will allow the user to know the location of the fault to a reasonable degree, depending on the distances between the tags.

To do that, the stretchable sensor data will first have to be converted from a capacitance vs data number scale to a capacitance vs time scale in order to be able to pair with the data obtained from the RFID. As the sampling frequency is known and can be set manually in the stretchable sensor program, an algorithm can be used to change the number of data to an actual time. By dividing the data number by the sampling frequency, the time in seconds can be obtained for each data.

$$\frac{Data\ number\ (Data)}{Sampling\ Frequency\ (Data/s)} = Time\ (s)$$

With this formula, the time of each capacitance value can be found.

With that, the initial plan was to plot the capacitance vs time graph from the sensor and the Tag number vs time graph from the RFID. However, this turns out to be not possible to be plotted as the tag number is not an actual value but just a name in character format and is not progressive thus it cannot be plotted. Therefore, to counter this, every capacitance value was assigned a tag number depending on the time. For example, capacitance values that have a time from 0 to 10 seconds will be tagged with a tag number of 2 provided that the time interval between each tag is 10 seconds.

The limitation of this method would be the time recorded for the sensor and RFID. If the time recorded from the sensor exceeds the time recorded from the RFID, there will not be enough tags to match the number of capacitance values after the last time recorded on the RFID, thus causing an error. The current method to go around this problem is to tag the extra capacitance values to the last tag number, this is due to the fact that any capacitance value recorded after the time the last tag is detected belongs to the area the LRT train goes after the last tag. A disclaimer message was included to inform the user that the data has exceeded and the data after the last tag is considered to be at last tag.

```
>> codeTest
========30-May-2018 16:25:05========
------Run 1------
wear at
    'Tag 6'

step at
    'Tag 6'

Sensor data exceeded number of RFID tags
Faults detected after last tag are considered to be at the last tag
```

Figure 4.9. Sample output when sensors data exceed number of RFID tag

4.3 Automated detection of faults

With the sensor and RFID integrated together, the next step was to develop a program in MATLAB that will detect and display any faults automatically based on the capacitance value of the sensor. Some of the common faults include normal wear, consistent wear, and 'step' or misalignment of track. A detection of centrifugal forces was also included to find out if there will be a huge force on the Current Collector Device (CCD) which may cause it to be damaged.

4.3.1 Wear

A wear can be identified as an abnormal change in capacitance value in the sensor. Building on this theory, the sample capacitance value can be compared with a set of pristine capacitance value to find out the difference and if it is more than a certain threshold, it will be considered as a fault.

```
if tagFlag == 1
    capacitanceDiff(capDiffCounter,1) = y(unhealthyCounter,1)- yhealthy(healthyCounter,1);
    capDiffCounter= capDiffCounter + 1;
    %check for wear in the same location
    if (yhealthy(healthyCounter,1)+ 2)<y(unhealthyCounter,1) %% if deviation is greater than noise, it is a wear
        capacitanceDiffUnhealthy(faultCounter,1) = y(unhealthyCounter,1)-yhealthy(healthyCounter,1);%form an array of the di
        capacitanceDiff_flag = 1;

        %{...%}

        faultCounter = faultCounter+1;
        startPosHealthySensor = startPosHealthySensor+1;

        if unhealthyCounter>1
            tagFlag2 = strcmp(tagNumberforUnhealthySensor(unhealthyCounter,1),tagNumberforUnhealthySensor((unhealthyCounter-
            if tagFlag2 == 0 % at different tag number from prev
                printFlag2 = 0; %reset the 'printed' flag
            end
            if tagFlag2 ==  0 || (printFlag2 == 0 && tagFlag2 == 1) %diff tag from prev OR same tag but no prev faults print
                 disp ('wear at');
                 disp (tagNumberforUnhealthySensor(unhealthyCounter,1));
                 printFlag2 = 1; %'printed' flag
            end %end if tagFlag2 == 0
        end %end if unhealthyCounter > 1
```

Figure 4.10. Code for detecting wear

Above is part of the code for detecting a wear. The threshold value used is 2 pF, meaning if the difference between the sample capacitance value and the pristine capacitance value is more than 2 pF, it will be deemed as a wear. The reason for choosing a value of 2 pF was that noises are around that range and hence if the difference in capacitance is more than the noise it can be considered a wear. When there is a wear at an area, the program will display a text that says 'Wear at Tag N' where N is the tag number associated with that specific capacitance value. Below is an example of the output when the code was run.

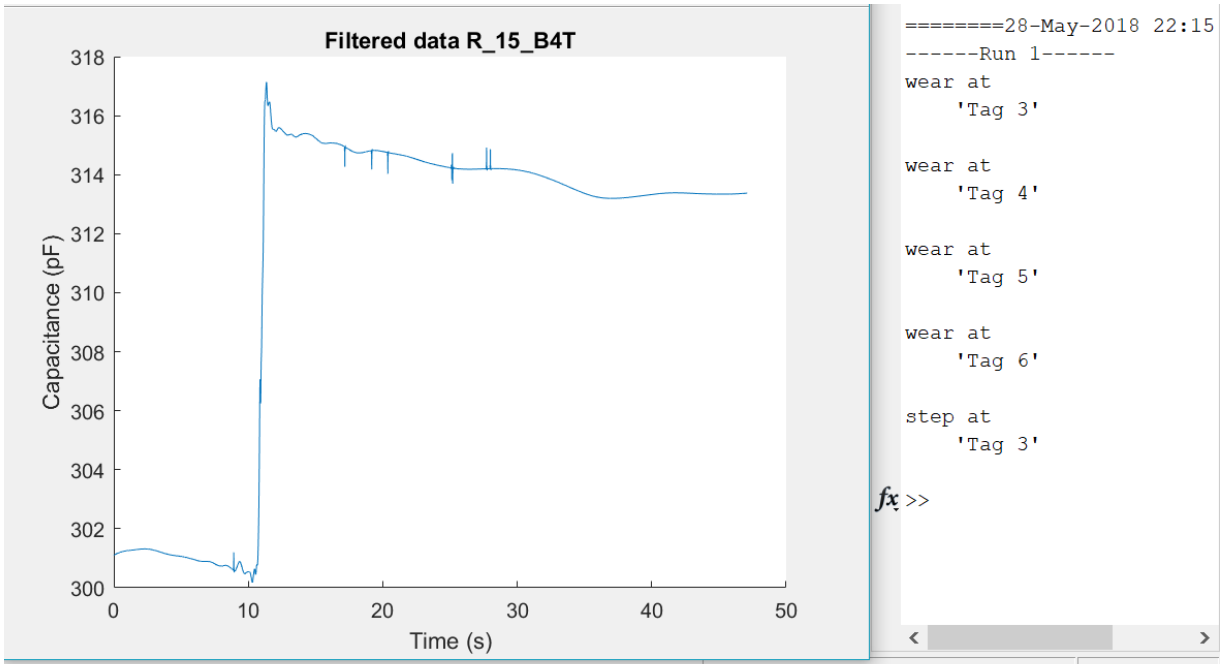


Figure 4.11. Sample output of wear

When faults are very prevalent along the track it can be classified as consistent wear, meaning that there are faults constantly along the track mainly caused by the friction between the track and the CCD over long runs. In the code when the number of faults detected adds up to 80% of the number of healthy data, and the standard deviation of the capacitance value is small, it will display 'Consistent wear of approximately x', where x is the mean value of the wear.

```
function consistentWear(obj, capacitanceDiff_flag, capacitanceDiff)
    if capacitanceDiff_flag == 1
        % check for consistent wear
        M = mean(capacitanceDiff);
        k = length(capacitanceDiff); %or k=k-1;
        counter = 0;
        for i3 = 1:k
            if (capacitanceDiff(i3,1)>(M -3)) && (capacitanceDiff(i3,1)<(M + 3))
                counter = counter+1;
            end
        end
        if counter>(obj.heightHealthy*8/10)%use heightUnhealthy as benchmark bec
            disp ('consistent wear of approximately');
            disp (M);
        end
    end
end %end function
```

```
wear at
    'Tag 2'

wear at
    'Tag 3'

wear at
    'Tag 4'

wear at
    'Tag 5'

wear at
    'Tag 6'

wear from
Tag 2 to Tag 6

consistent wear of approximately
12pF or 4mm
```

Figure 4.12. Sample output of consistent wear

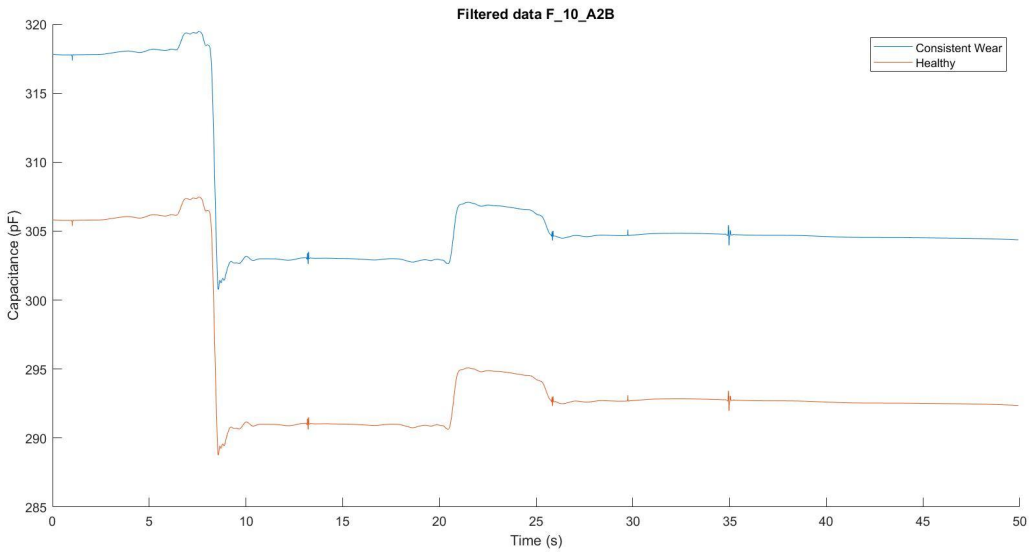


Figure 4.13. Sample graph of consistent wear

The above figures show that the shapes of the two graphs are the same, but the overall capacitance value of the sample data is higher (by 12 pF). This reflects a consistent wear as the track is worn out and the displacement of the sensor will be larger compared to a healthy track and thus the value will be higher.

4.3.2 Step

A step happens when there is a misalignment of the LRT track and can be identified by a sudden change in capacitance value. Using the same theory as detecting wear, the difference between the healthy and sample data can be found to see if it exceeds a certain value. The code is shown below.

```
if capDiffCounter>2
    if (capacitanceDiff(capDiffCounter,1)-15) > capacitanceDiff((capDiffCounter-1),1)%criteria to identify step
            disp ('step at');%butterworth filter make a smooth transition making it difficult to identify steps
            %check for steps with butterworth
            %filter if possible, if not there is
            %another check below
            disp (tagNumberforUnhealthySensor(unhealthyCounter,1));
    end %end if
end %end if
```

Figure 4.14. Sample code for detecting step

The theory behind this code is to compare the current difference in capacitance value between the healthy and unhealthy data with that from its previous set. If the two values are different by more than 15 pF, there may be a step. However, there are some flaws with this code. Firstly, the capacitance difference will only be recorded when there is a fault, meaning if there is no fault before the step occurs, it will not be able to identify it as a step as there are no previous values to compare it to. Secondly, if the fault before the step is also high in capacitance (a step), the current step will not be recognised as well due to the low difference in value between the current step and the previous fault which is also a step. Below is an illustration of the problem.

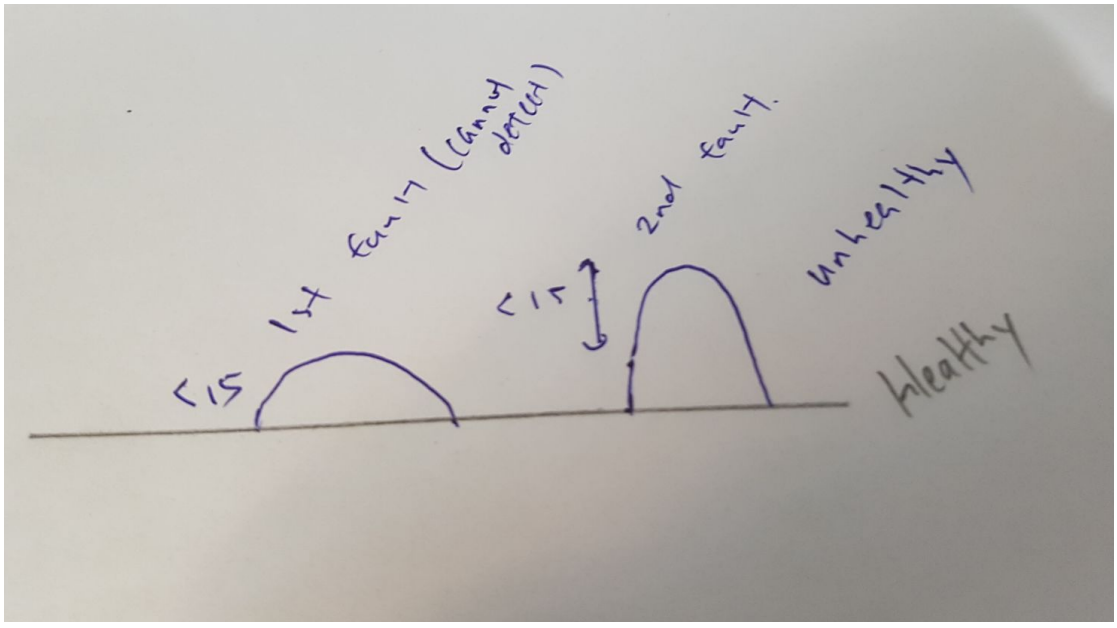


Figure 4.15. Illustration of the problem with step code

To solve this issue, the method of detecting a step has been revised.

The new method of detection uses only the sample data itself instead of comparing it to a healthy set of data. By taking the mode (most occurring/common) capacitance value

of all the data for each tag number as a standard value, every capacitance value with the same tag number will be compared against it. If there are any capacitance difference of more than 15 pF, there is a step. The mode of different tag numbers will also be compared against each other. Below are the code and sample output.

```
            tagCounter = tagCounter + 1;
        else
            capforTag(tagCounter,1) = y(unhealthyCounter,1);
            tagCounter = tagCounter + 1;
            startingPos = unhealthyCounter+1;
            break;
        end
    end
    modeOfCap = mode(capforTag);

    for tagSelector = 1:tagCounter-1
        if (capforTag(tagSelector,1) - 15) > modeOfCap && printFlag == 0 %can be less than TOO
            disp ('step at');
            disp(tagNumberforUnhealthySensor(unhealthyCounter,1));
            printFlag = 1;
        end
    end
    if modeOfCap - 15 > prevMode && tagType > 1 && printFlag == 0
        disp ('step at');
        disp (tagNumberforUnhealthySensor(unhealthyCounter,1));
        printFlag = 1;
    end
    prevMode = modeOfCap;
end
```

Figure 4.16. Code for new method for step detection

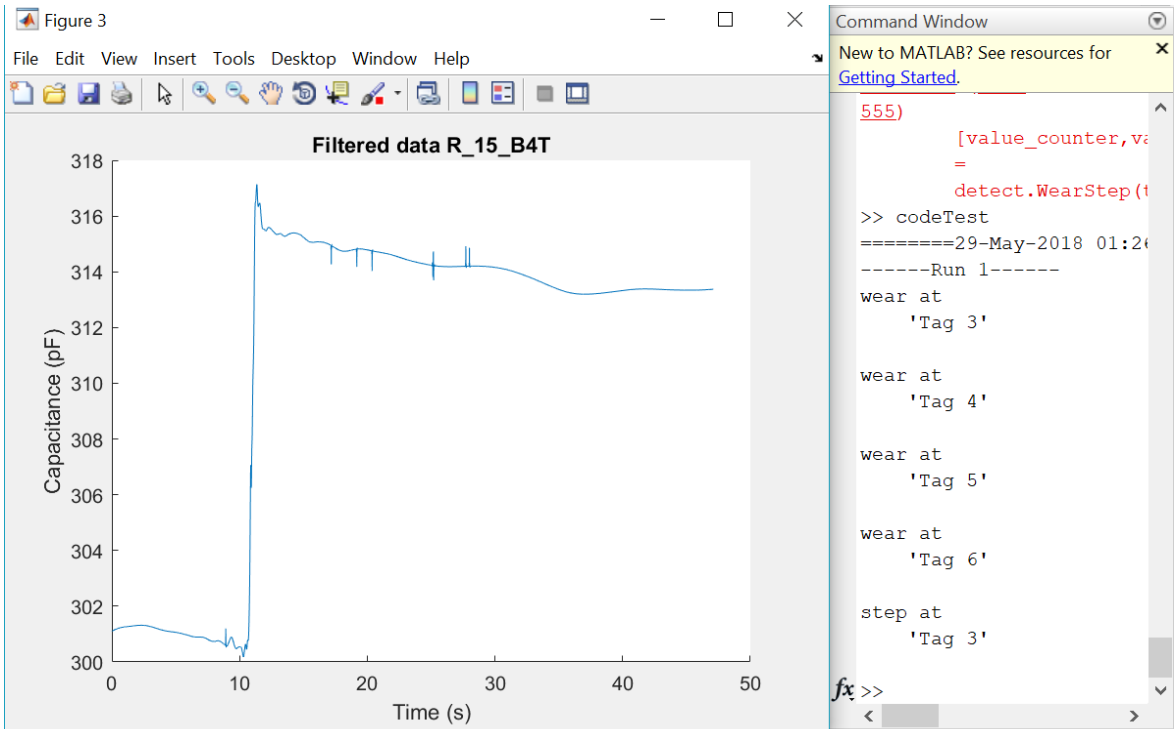


Figure 4.17. Sample output of step detection

The above output shows that the code is correctly detecting a step at Tag 3, which has a time interval from 10s to 20s. This method, however, will only work for straight tracks as the capacitance values change constantly when the train makes a turn and thus there will not be a mode value to compare with. The method of detection will need further improvements, perhaps the effect of centrifugal forces on the sensor's capacitance value can be analysed.

4.3.3 Centrifugal forces [Pre-analysis]

When the LRT train makes a turn, there will be a centrifugal force acting on the train towards the outer side of the turn. This force may result in a force acting on the CCD which will affect the capacitance value of the stretchable sensor attached to it. If the magnitude of said force is high enough, it may cause the CCD to fully compress and become a rigid body, which may lead to the damage of the CCD over a period of time.

```
if tagFlag4 == 0 %if diff means new tag number
    printFlag4 = 0; %reset 'printed' flag at every new tag number
end

if(noisy_y(j,1) <= 250)% 250 being the resting capacitance of the sensor
    if tagFlag4 == 0 || (tagFlag4 == 1 && printFlag4 == 0)
        disp ('Outer centrifugal forces acting at');
        disp (x2(j,1));
        printFlag4 = 1; %'printed' flag
    end
end

tagFlag5 = strcmp(x2(j,1),x2((j-1),1)); %check if same tag as prev
if tagFlag5 == 0 %if diff means new tag number
    printFlag5 = 0; %reset 'printed' flag at every new tag number
end

if(noisy_y(j,1) >= 390)% 390 being the maximum capacitance of the sensor when CCD fully ex
    if tagFlag5 == 0 || (tagFlag5 == 1 && printFlag5 == 0)
        disp ('inner centrifugal forces acting at');
        disp (x2(j,1));
        printFlag5 = 1; %'printed' flag
    end
end %end if
```

Figure 4.18. Code for detecting centrifugal forces

The method to detect this is to compare the capacitance value of the sensor to its capacitance values at maximum compression and extension. The maximum compression signifies that the CCD has been fully compressed and thus becoming a rigid body, while the maximum extension signifies that the CCD has been fully extended and it might detach from the power rail and lead to a breakdown.

As centrifugal forces are difficult to simulate, there are currently no methods to determine the accuracy of the code and it should be analysed again when more testings are available.

4.4 Recalculation of centrifugal forces

4.4.1 Centrifugal Forces calculation equations

Due to the turns present on the LRT tracks, there is a possibility that when turning, the centrifugal forces may have an effect on the displacement of the CCD on the train. This will result in a possible change in capacitance value recorded by the stretchable sensor. Therefore, the purpose of calculating the centrifugal forces is to allow the displacement caused by it on the CCD to be calculated, allowing the capacitance change of the sensor connected to the CCD to be subsequently calculated. Additionally, it is also to determine if the train will slip or tip.

While there is previous work done on calculating the centrifugal forces' effects on the CCD, it did not take into account the forces absorbed by the guide wheels. The centrifugal forces calculation calculated in previous work assumed all forces to be taken by the CCD, this is not true as the guide wheels are also there to share the load and perhaps even the majority of it.

To calculate the centrifugal forces, the formula $\frac{MV^2}{R}$ where M is the mass in kilograms (kg), V is the velocity in meter per second (m/s) and R for the radius of curvature in meter (m) is used. With this, recalculation of the overall forces acting on the LRT itself was done. Shown below is the Free Body Diagram that is used for 0° bank as well as the forces on the actual LRT schematics.

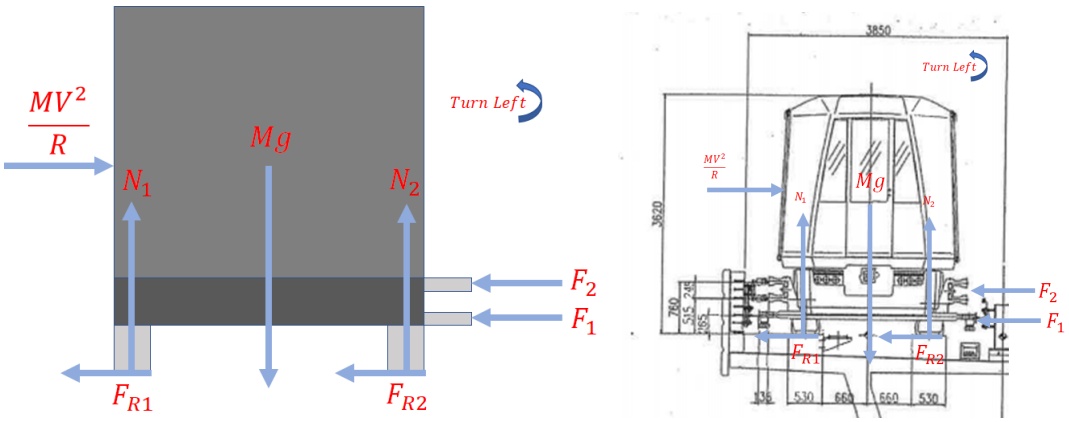


Figure 4.19. Free body diagram (Left) and Schematic drawing (Right) of LRT train

With these FBD, calculations can be formulated to find the force acting on the CCD. So the equations that are used are :

For $\Sigma F_x = 0$; **- Equation 1**

$$F_1 + F_2 + F_R = \frac{MV^2}{R}$$

Using Spring constant; **- Equation 2**

$$\frac{F_1}{K_1} = \frac{F_2}{K_2}$$

$$F_1 = \frac{F_2 K_1}{K_2}$$

Subbing in Equation 2 into Equation 1, it can be shown as:

$$\frac{F_2 K_1}{K_2} + F_2 + F_R = \frac{MV^2}{R}$$

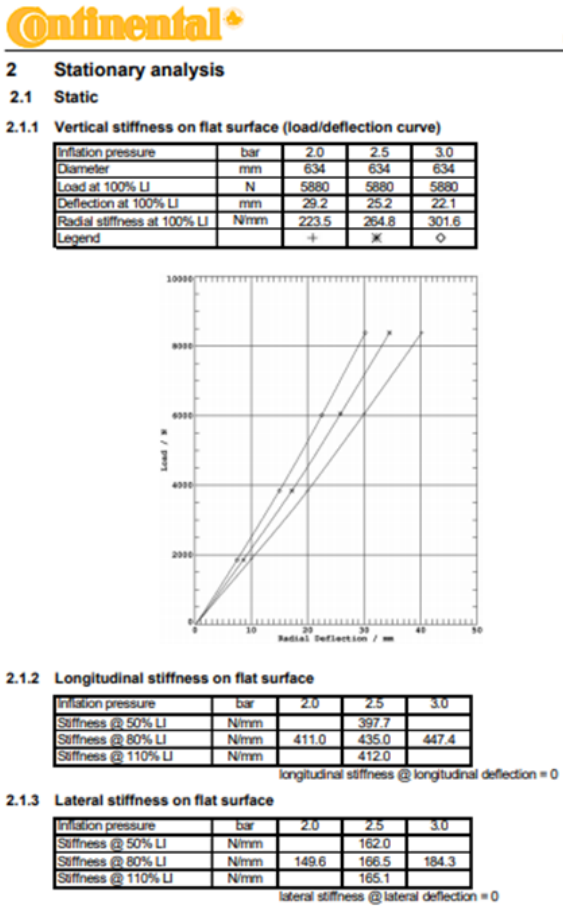
Using this, the F_2value, which is the force acting on the CCD, can be effectively calculated

However, one number that is lacking is the Keff value of the guide wheels, K_1. To get this value, 2 methods were proposed. These 2 methods will give two different results but did actually bring the author closer to a more accurate depiction of the centrifugal forces acting on the CCD.

4.4.2 Obtaining Keff value for Guide Wheels

The first method of getting the K effective value is through the use of datasets from online sources. For this dataset, the vertical stiffness of the tyres is sought. This value, however, turns out to not be a very commonly found value that can be sourced online. Based on research, this sort of value is usually confidential, thus, finding the value was difficult.

Fortunately, a single dataset from a company named Continental AG was able to be acquired. This data actually provided the vertical stiffness of their conventional car tyres as shown below. [10]



Continental

2 Stationary analysis

2.1 Static

2.1.1 Vertical stiffness on flat surface (load/deflection curve)

Inflation pressure	bar	2.0	2.5	3.0
Diameter	mm	634	634	634
Load at 100% LI	N	5880	5880	5880
Deflection at 100% LI	mm	29.2	25.2	22.1
Radial stiffness at 100% LI	N/mm	223.5	264.8	301.6
Legend		+	x	◇

2.1.2 Longitudinal stiffness on flat surface

Inflation pressure	bar	2.0	2.5	3.0
Stiffness @ 50% LI	N/mm		397.7	
Stiffness @ 80% LI	N/mm	411.0	435.0	447.4
Stiffness @ 110% LI	N/mm		412.0	

longitudinal stiffness @ longitudinal deflection = 0

2.1.3 Lateral stiffness on flat surface

Inflation pressure	bar	2.0	2.5	3.0
Stiffness @ 50% LI	N/mm		162.0	
Stiffness @ 80% LI	N/mm	149.6	166.5	184.3
Stiffness @ 110% LI	N/mm		165.1	

lateral stiffness @ lateral deflection = 0

Figure 4.20. Sample data from Continental

Using the values in this dataset, the Keff of the guide wheels was able to be obtained.

As the sizes of the guide wheels are actually different from the continental wheels, scaling law was utilized. The author derived the following relation :

$$K = AE/L \propto D$$

Where K stands for stiffness, A for cross-sectional area, E for Young's Modulus, L for length, D stands for characteristic dimension

This was derived as Area is D * D which makes it proportional to D^2, Length is proportional to D and E is assumed to be the same. With this, $D^2/D = D$ was obtained. Therefore, it was concluded that stiffness scales proportionally with size. In layman terms, if the guide wheels are 10 times bigger than the tyres from Continental, the Keff value will also be 10 times bigger.

With this, the authors are able to calculate the Keff of the guide wheel for the first method.

For calculation of the scaling value:

$$\frac{A_{GW}}{L_{GW}} = x\frac{A_{CW}}{L_{CW}}$$

Where x is the scaling ratio A_{GW} and L_{GW} are the area and length for guide wheels respectively while A_{CW} and L_{CW} are the area and length for Continental wheels

$$\therefore \frac{A_{GW}}{L_{GW}} \times \frac{L_{CW}}{A_{CW}} = x$$

$$\frac{150 \times 135}{135} \times \frac{317}{183 \times 317} = 0.8196721311$$

So with this, the Keff value for the guide wheels can be calculated

It is roughly around K_{Scal} = 217049 N/m

However, a limitation to using this method of calculating is that usually, the formula K = AE/L is only used when the cross-sectional area A is uniform, however as the subject is a tyre and the force is acting on a non-uniform cross-sectional area, the K value that is calculated will be slightly higher than it is in reality.

For the next method of calculating the Keff value, the same formula will be used

$$K = AE/L$$

However, this time it will be calculated it using Young's Modulus directly. The Young's Modulus was decided to be of natural rubber which is what is suspected to be the material used to construct the guide wheels. The E value of natural rubber is between 0.0015GPa and 0.0025GPa. This value is obtained through Cambridge University Engineering Department: Materials Data Book 2003 ed. With this, the values were input in and the Keff value was obtained.

$$\frac{150\ mm \times 135\ mm \times 0.0020\ GPa}{135\ mm} = 300\ 000\ \text{N/m}$$

It shall be named K_{Calc} = 300 000 N/m

Looking at the two Keff values, they are different, this can be because, for K_{Scal} the material of the tyre is actually pressurized air with a rubber covering but for K_{Calc} the material is actually fully solid natural rubber. The author decided to do it this way because, as can be seen in the figure below which depicts the guide wheels, there is a high chance that it is actually constructed out of fully solid rubber. However, the author does not want to rule out the possibility that it might be an air pumped tyre.

Figure 4.21. Photo of Guide wheel

4.4.3 Results analysis of Centrifugal forces

With the Keff values obtained, the values shall be input into the equation formed previously to find the force acting on the CCD.

$$\frac{F_2 K_1}{K_2} + F_2 + F_R = \frac{MV^2}{R}$$

$$F_2\left(\frac{K_1}{K_2} + 1\right) = \frac{MV^2}{R} - F_R$$

$$F_2 = \left(\frac{MV^2}{R} - F_R\right)/\left(\frac{K_1}{K_2} + 1\right)$$

E.g calculation :

$$F_2 = \left(\frac{21207 \times 15.27777778^2}{50} - (21207 \times 9.81 \times 0.45)\right)/\left(\frac{300\,000}{213.7} + 1\right)$$

$$= 3.829928\ N$$

From this, it was actually found that, with a mass of 21207 Kg, 50 meter radius of curvature,a force will start acting on the CCD from the velocity of 53.48 km/h onwards. This actually indicates that the LRT will slip slightly starting from that velocity onwards with those parameters.

From this force on CCD, the author actually managed to deduce a few things. One of them is that the guide wheel actually takes up a huge portion of the force. This is due to the fact that, in previous calculations, the force on CCD when it was taking the entire load is roughly around 32kN but now using the new data, the force at the same velocity is roughly only around 3.8N for the calculated value and 5.2N for the scaled as seen below.

Keff of Calculated Guide Wheel N/m	Keff of scaled down of air tyres N/m	Keff Value N/m of CCD	Velocity of train/kmh^-1	Velocity of train/ms^-1	F=(MV^2/R)	Friction when Static friction=0.45 /N	Force on CCD?	M1 (Moments on both Wheels) / Nmm (Calc)	M1 (Moments on both Wheels) / Nmm (Scaled)	F2 (Force on CCD exerted by power rail) /N (Calc)	F2 (Force on CCD exerted by power rail) /N (Scaled)	x/m (Calc)	x/m (Scaled)	x/mm (Calc)	x/mm (Scaled)

300000	217049.1803	213.7	55	15.27777778	98998.72685	93618.3015	force on CCD	215379.9724	212518.184	3.82992814	5.292192094	0.01792198475	0.02476458631	17.92198475	24.76458631

Figure 4.22. Table of data for centrifugal forces

This means that instead of previously the CCD compressing by 42 m straight away at that velocity, it now only compresses by roughly around 17mm using calculated and 24mm using scaled calculations. This is more reasonable as the total compression of the CCD is only roughly around 90mm.

4.4.4 Radius of curvature

Now that the basic values of the centrifugal forces and when the force will start to act on the CCD is calculated, the authors decided it is time to use a more realistic radius of curvature data. This is because the previous radius of curvature used, 50m can be considered an arbitrary number. With a more realistic radius of curvature, the calculation of force on the CCD can be closer to the real value.

In order to obtain this new radius of curvature, the authors looked at the data that SBS provided on the track path of the Sengkang LRT as seen below.

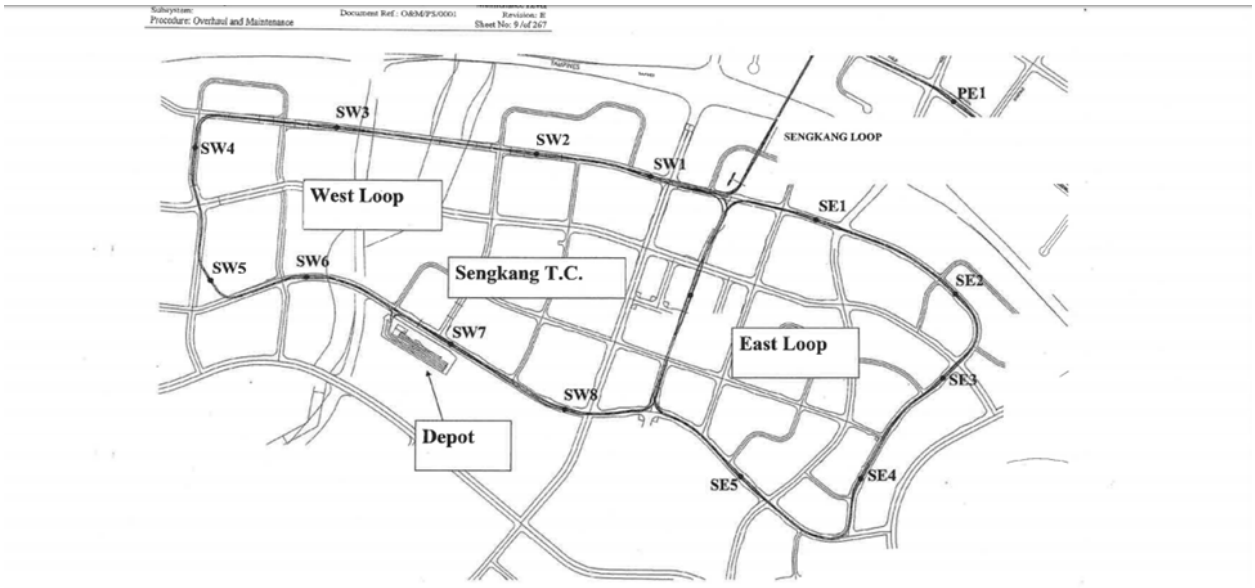


Figure 4.23. Map of Sengkang LRT

Once this image was found, the location of it was found on Google and the satellite imagery was observed. [11]

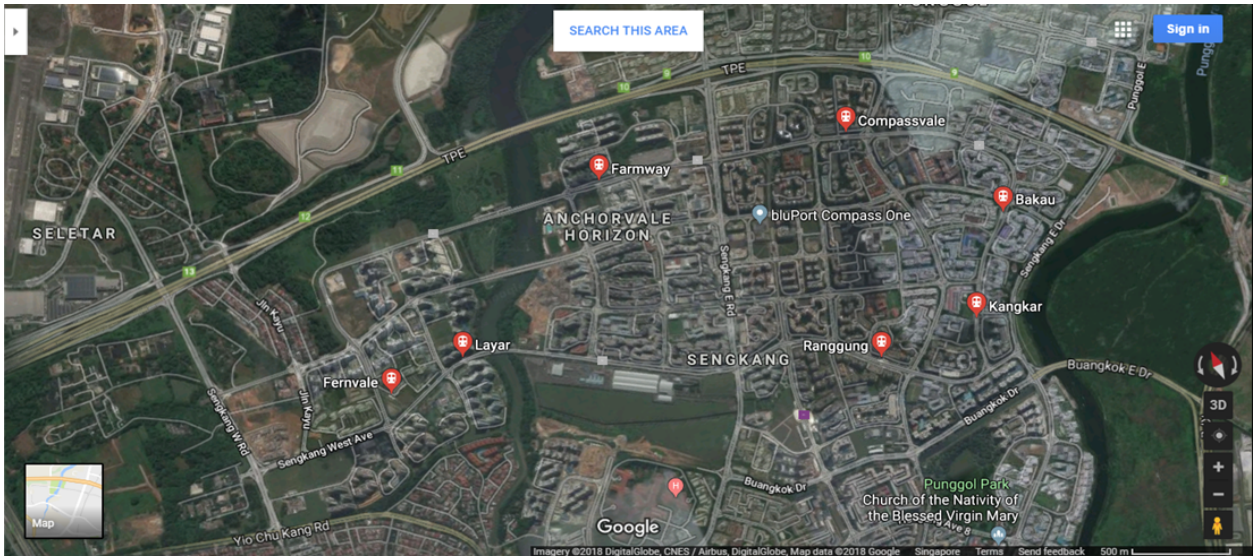
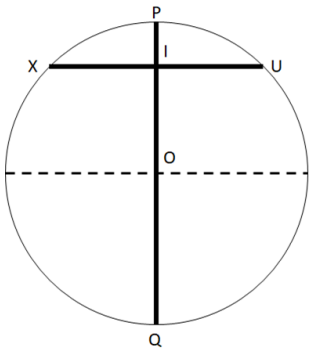


Figure 4.24. Sengkang LRT map in Google Maps

Now, to pick the bent needed to calculate the radius of curvature. For this, the bent shown below were used as it is one of the more drastic turns that the LRT has to turn at.

Figure 4.25. Top left corner of Sengkang LRT loop

With this, the intersecting chord theorem will be used to calculate the radius of curvature.



Intersecting Chord Theorem states that :
XI * IU = PI * IQ

Let XU be ℓ, PI be h

XI = IU = $\frac{\ell}{2}$; PI = h

$$\therefore \frac{\ell}{2} \times \frac{\ell}{2} = h \times IQ$$

$$IQ = \frac{\ell^2}{4h}$$

$$Diameter = IQ + PI = \frac{\ell^2}{4h} + h = \frac{\ell^2 + h^2}{4h}$$

$$Radius = \frac{\ell^2 + h^2}{8h}$$

Using the above formula the radius can be obtained with just the arc and the chord length.

To obtain those, measurements using google map tools were used. Below are the measurements taken.

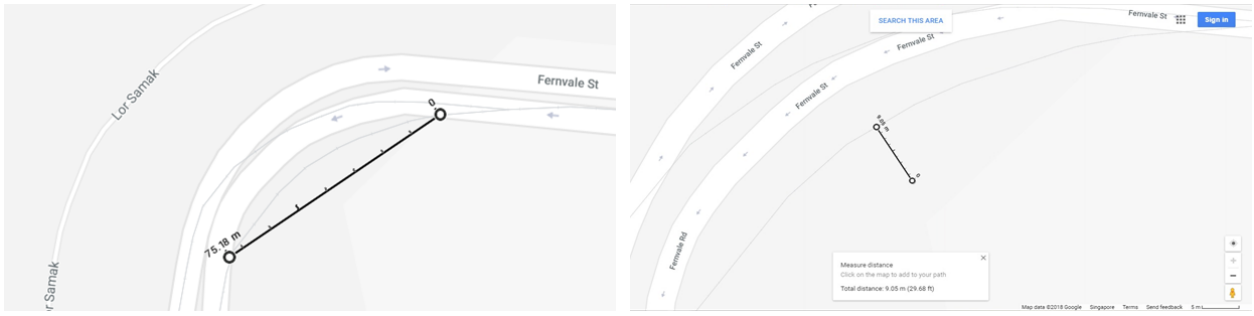


Figure 4.26. Measurement of chord length and height using Google tools

Keying in the values, the radius would be

$$Radius = \frac{75^2 + 9^2}{8(9)} = 79m \approx 80m$$

With this value, the centrifugal forces were calculated again to see when force will start to act on the CCD. It was found that the force will only start at roughly around 70 km/h and this is can be seen as more realistic as the author do not believe that the operators of the LRT would want any force on their CCD before reaching its operating speed of 70 km/h.

Keff of Calculated Guide Wheel N/m	Keff of scaled down of air tyres N/m	Keff Value N/m of CCD	Velocity of train/kmh^-1	Velocity of train/ms^-1	F=(MV^2/R)	Friction when Static friction=0.45 /N	Force on CCD?	M1 (Moments on both Wheels) / Nmm (Calc)	M1 (Moments on both Wheels) / Nmm (Scaled)	F2 (Force on CCD exerted by power rail) /N (Calc)	F2 (Force on CCD exerted by power rail) /N (Scaled)	x/m (Calc)	x/m (Scaled)	x/mm (Calc)	x/mm (Scaled)
300000	217049.1803	213.7	70	19.44444444	100225.9838	93618.3015	force on CCD	217742.1166	214227.5639	4.70352188	6.499323329	0.02200992925	0.03041330524	22.00992925	30.41330524

Figure 4.27. Table of data for centrifugal force at 80 m radius of curvature

4.4.5 Toppling

When the LRT train experiences a centrifugal force, two scenarios may happen, one being that it will slip due to the centrifugal force overcoming the frictional force, while the other is that it may topple. This analysis is important due to the fact that toppling or even the tilting of the train will result in a force on the CCD.

Using the same FBD previously shown, the moments on the LRT train can be analysed. As moments due to the centrifugal force plays a huge role in the toppling of the train, the moments will be taken about two points, the inner and outer wheel when turning.

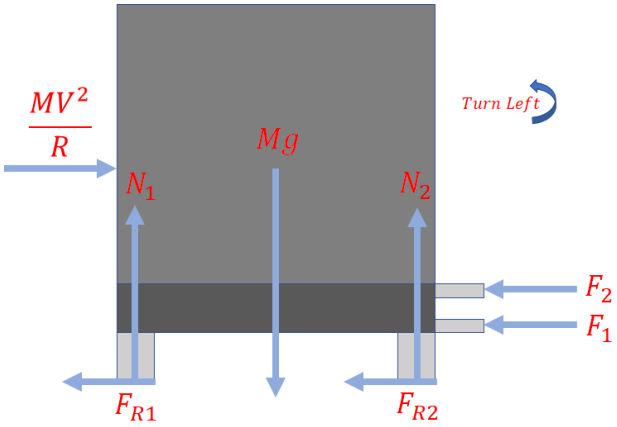


Figure 4.28. Free body diagram of LRT train

By taking moments about the inner wheel (Left), the following equation can be formed:

$$mg\frac{d}{2} + \frac{mv^2}{r}h = N_2 d$$

$$N_2 = \frac{mg\frac{d}{2} + \frac{mv^2}{r}h}{d}$$

By taking moments about the outer wheel (Right), the following equation can be formed:

$$mg\frac{d}{2} - \frac{mv^2}{r}h = N_1 d$$

$$N_1 = \frac{mg\frac{d}{2} - \frac{mv^2}{r}h}{d}$$

Where d is the distance between the two normal forces, h is the height from the ground to the center of the train, m is the mass of the train, r is the radius of curvature and v is the velocity of the train

From the previous equation, it is known that when the velocity increases, the normal force on the inner wheel N_1 will decrease. When N_1 reaches a value of 0N and below, it means that the inner wheel has lifted off the ground and the LRT train has been tilted to one side and it might have a chance to topple.

The formulas were used in the excel sheet and the results are shown below.

Radius of Curvature /m	Mass of train with passengers/kg	Taking Moments about	Velocity of train/kmh ^-1	Velocity of train/ms^-1	F=(MV^ 2/R)	Inner Reaction Force	Outer Reaction Force	Chance to topple?
80	21207	Outer	67.5	18.75	93194.82422	-6302.186643	Not Valid	Yes
80	21207	Outer	70	19.44444444	100225.9838	-14625.55933	Not Valid	Yes
80	21207	Outer	72.5	20.13888889	107512.8219	-23251.60012	Not Valid	Yes
80	21207	Outer	75	20.83333333	115055.3385	-32180.309	Not Valid	Yes
80	21207	Outer	77.5	21.52777778	122853.5337	-41411.68599	Not Valid	Yes
80	21207	Outer	80	22.22222222	130907.4074	-50945.73107	Not Valid	Yes

Figure 4.29. Table of data for toppling

From the table above, it is shown that the train will have a chance to topple at a velocity of 67.5 km/h and above.

In conclusion, the centrifugal force on the LRT train will cause it to tilt to the outer side at a velocity of 67.5 km/h and cause it to slip at 70 km/h, resulting in a force on the CCD.

4.4.6 10° bank

With the calculation of centrifugal forces on a levelled track more or less completed, the next step would be calculating the same forces on an angled track which in this case is 10°.

The FBD is slightly different from the previous FBD, with the centrifugal force and the mass affecting the friction and the also the moment. Below is the FBD.

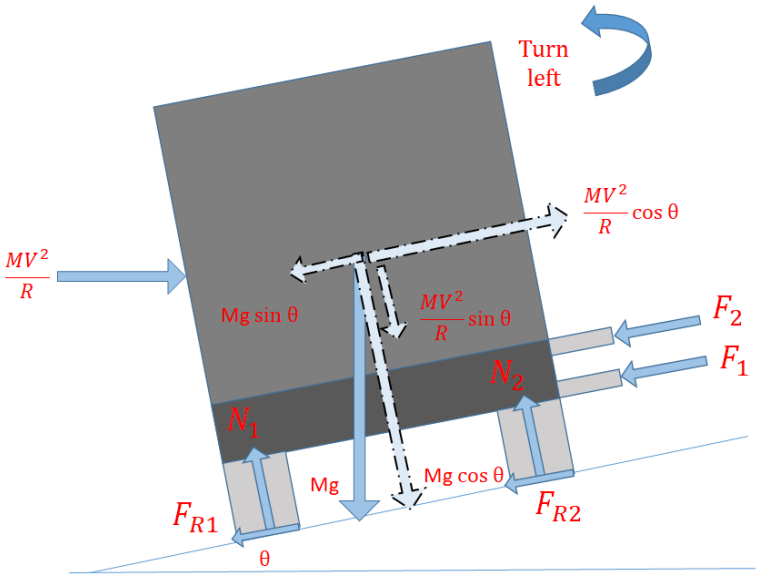


Figure 4.30 Free Body Diagram at 10° bank

Using the same method of analysis, the following equations can be derived:

For $\Sigma F_x = 0$; **- Equation 1**

$$F_1 + F_2 + F_R = \frac{MV^2}{R}cos\theta - Mgsin\theta$$

Using Spring constant; **- Equation 2**

$$\frac{F_1}{K_1} = \frac{F_2}{K_2}$$

$$F_1 = \frac{F_2 K_1}{K_2}$$

Subbing in Equation 2 into Equation 1, it can be shown as:

$$\frac{F_2 K_1}{K_2} + F_2 + F_R = \frac{MV^2}{R}cos\theta - Mgsin\theta$$

Using this, the F_2 value, which is the force acting on the CCD, can be effectively calculated

$$F_2\left(\frac{K_1}{K_2}+1\right) = \frac{MV^2}{R}cos\theta - Mgsin\theta - F_R$$

$$F_2 = \left(\frac{MV^2}{R}cos\theta - Mgsin\theta - F_R\right)/\left(\frac{K_1}{K_2}+1\right)$$

The formula is used to calculate the set of data in the excel sheet.

Keff of Calculated Guide Wheel N/m	Keff of scaled down of air tyres N/m	Keff Value N/m of CCD	Velocity of train/kmh ^-1	Velocity of train/ms ^-1	F=(MV^2/R)	Friction when Static friction=0.45 /N	MV^2/R * Cos - Mg*Sin	Frictional Force acting on?	Force on CCD?	M1 (Moments on both Wheels) / Nmm (Calc)	M1 (Moments on both Wheels) / Nmm (Scaled)	F2 (Force on CCD exerted by power rail) /N (Calc)	F2 (Force on CCD exerted by power rail) /N (Scaled)	x/m (Calc)	x/m (Scaled)	x/mm (Calc)	x/mm (Scaled)
300000	217049.1803	213.7	80	22.22222	130907.4074	102425.3539	92792.74651	Left	no force on CCD	154769.1998	154769.1998	0	0	0	0	0	0

Figure 4.31. Centrifugal Forces at 80 km/h at 10° bank

Above is the data for the train at 80 km/h and as can be seen there is no force on the CCD at 80 km/h, which is its maximum velocity.

The calculations for toppling was also done as follow:

By taking moments about the inner wheel (Left), the following equation can be formed:

$$\left(Mgcos\theta + \frac{MV^2}{R}sin\theta\right) \times \frac{d}{2} + \left(\frac{MV^2}{R}cos\theta - Mgsin\theta\right) \times h = N_2 d$$

$$N_2 = \frac{\left(Mgcos\theta+\frac{MV^2}{R}sin\theta\right)\times\frac{d}{2}+\left(\frac{MV^2}{R}cos\theta-Mgsin\theta\right)\times h}{d}$$

By taking moments about the outer wheel (Right), the following equation can be formed:

$$\left(Mgcos\theta + \frac{MV^2}{R}sin\theta\right) \times \frac{d}{2} - \left(\frac{MV^2}{R}cos\theta - Mgsin\theta\right) \times h = N_1 d$$

$$N_1 = \frac{(Mgcos\theta + \frac{MV^2}{R} sin\theta) \times \frac{d}{2} - (\frac{MV^2}{R} cos\theta - Mgsin\theta) \times h}{d}$$

Where d is the distance between the two normal forces, h is the height from the ground to the center of the train, M is the mass of the train, R is the radius of curvature and V is the velocity of the train

The formula is used in the excel sheet again and the results are shown below.

Radius of Curvature /m	Mass of train with passengers/kg	Taking Moments about	Velocity of train/kmh^-1	Velocity of train/ms^-1	F=(MV^2/R)	Inner Reaction Force	Outer Reaction Force	Chance to topple?
80	21207	Outer	67.5	18.75	93194.82422	44650.34817	Not Valid	No
80	21207	Outer	70	19.44444444	100225.9838	37063.90024	Not Valid	No
80	21207	Outer	72.5	20.13888889	107512.8219	29201.58147	Not Valid	No
80	21207	Outer	75	20.83333333	115055.3385	21063.39187	Not Valid	No
80	21207	Outer	77.5	21.52777778	122853.5337	12649.33144	Not Valid	No
80	21207	Outer	80	22.22222222	130907.4074	3959.400171	Not Valid	No

Figure 4.32. Table of data for toppling at 10° bank

Compared to the data of the 0° bank which showed that there is a chance to topple at 67.5 km/h, this data showed that the LRT train will not topple at that velocity and even up till its maximum velocity.

Hence it can be concluded that the bank aids in the reduction of the effect of the centrifugal force.

4.5 Allowable-Non Allowable Graph for Radius and Speed

Now that the centrifugal forces have been calculated, the allowable-non allowable curve can be plotted. The main information this will provide is the maximum operational speed of the LRT when it is turning which can then be used in the detection program to identify any abnormalities when turning. As mentioned in section 4.4, there are two situation that might happen during turning that can have an adverse effect on the current collector device. Firstly is the possibility of slipping and other is toppling.

4.5.1 Slipping

On Wet Concrete

The criteria chosen for it to be non-allowable when slipping are as follows : For inner radius, it is when the CCD becomes fully extended and is about to be detached from the power rail. For the outer radius, it is when the CCD is fully compressed and becomes a rigid body as any further compression will cause damage to the CCD itself. Seen below is the graph when the LRT is moving at 0°, 5° and 10° bank.

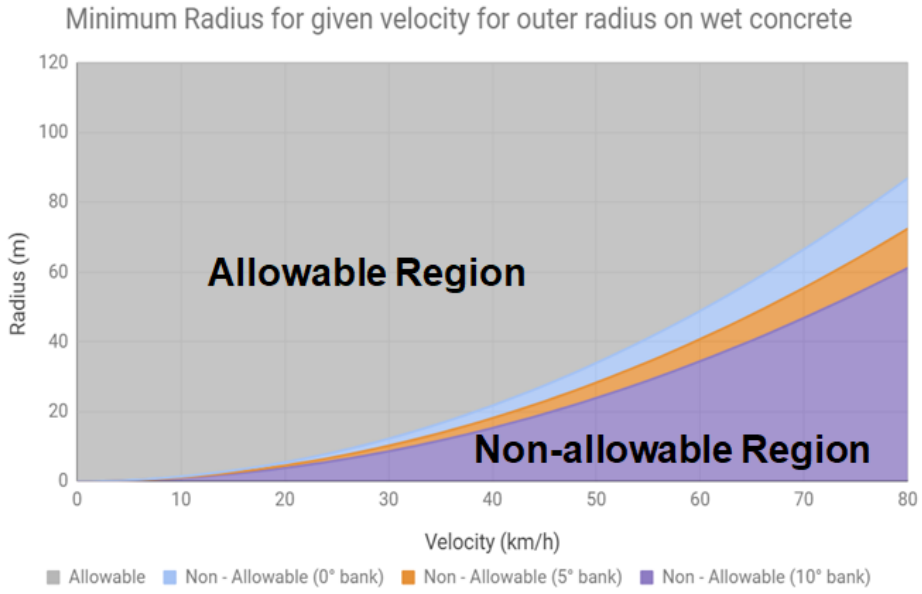


Figure 4.33. Allowable Non-Allowable Curve for Outer Radius when Slipping on Wet Concrete

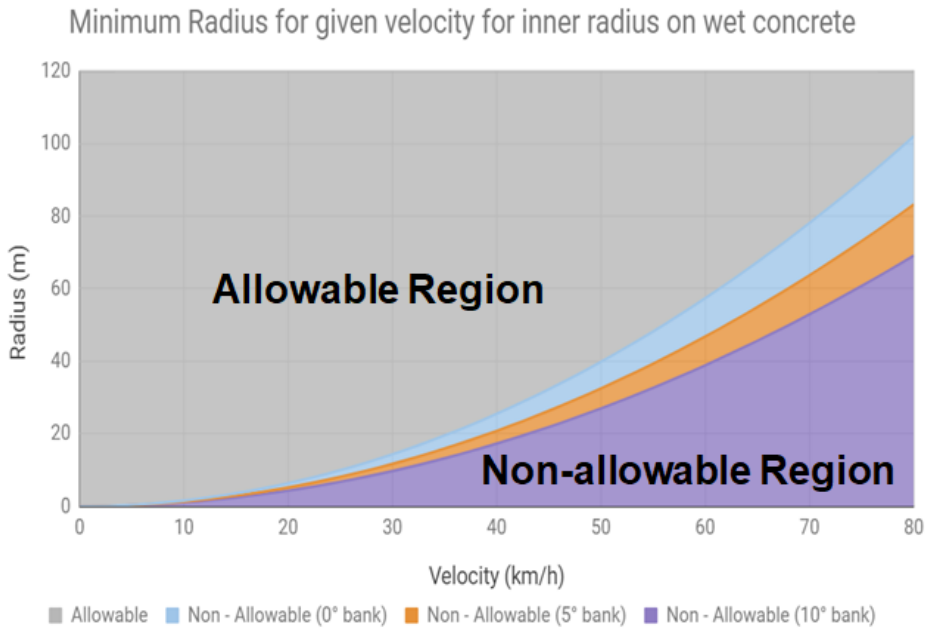


Figure 4.34. Allowable Non-Allowable Curve for Inner Radius when Slipping on Wet Concrete

Figure 4.33 and 4.34 plots the allowable non-allowable region for any given speed for the outer and inner radius respectively. A conservative approach was taken when determining the coefficient of friction, setting it as wet concrete against rubber. With this, it can be known if the train is operating above its operational limits, an example, if the train is moving around a curve of 80 meters, if it is detected to be going over the speed of 70 km/h, problems might arise. It can also be confirmed here in the graph, that the banks does affect the forces significantly as expected. When the bank increase, so does the allowable area, allowing higher speed to be obtained at a sharper turn.

On Dry Concrete

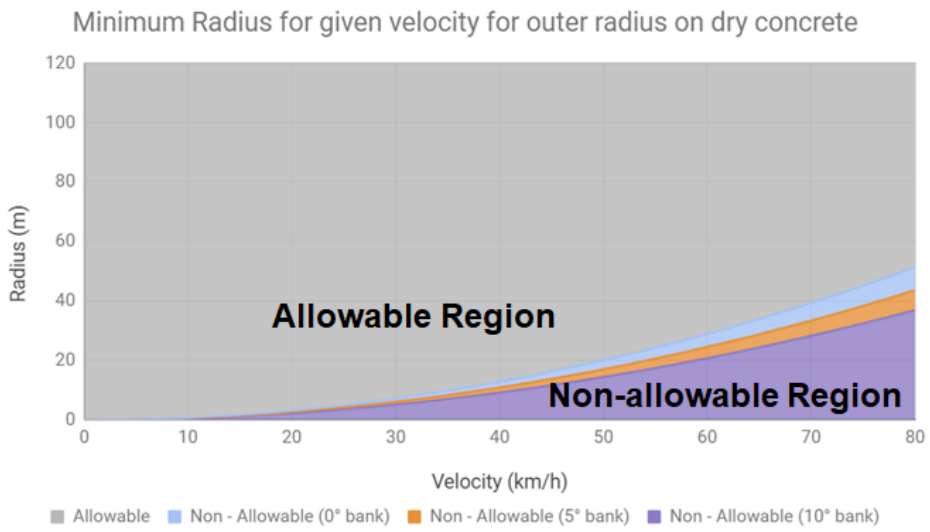


Figure 4.35. Allowable Non-Allowable Curve for Outer Radius when Slipping on Dry Concrete

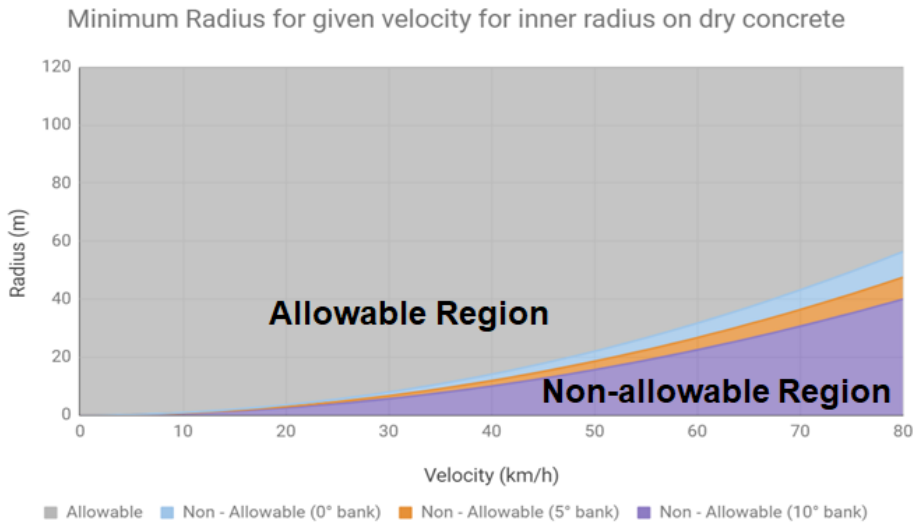


Figure 4.36. Allowable Non-Allowable Curve for Inner Radius when Slipping on Dry Concrete

As shown above in 4.35 and 4.36, in addition to calculating when it is wet, the calculation for when the contact point between the LRT and the ground is dry is also calculated. This is done to show that there is a difference if the ground is dry and as expected, a larger allowable region is obtained which means that in the future, there will be a need to find out the conditions of the tracks so that a more accurate analysis of the tracks can be obtained.

Comparison with previous work

When looking at previous work done on calculating the centrifugal forces and plotting of allowable non-allowable curve, the criteria used was conservative, in that the moment a force is exerted on the CCD, it is to be counted as non-allowable. With this new data, it is set so that it only counts as non-allowable when it reaches the limit, the point where damage will actually start to happen on the CCD. What this provides is a larger region of allowable range for the speed of the LRT, allowing it to go faster at narrower turns. Shown in figure 4.37 is a comparison of the calculations from previous work and the current calculations.

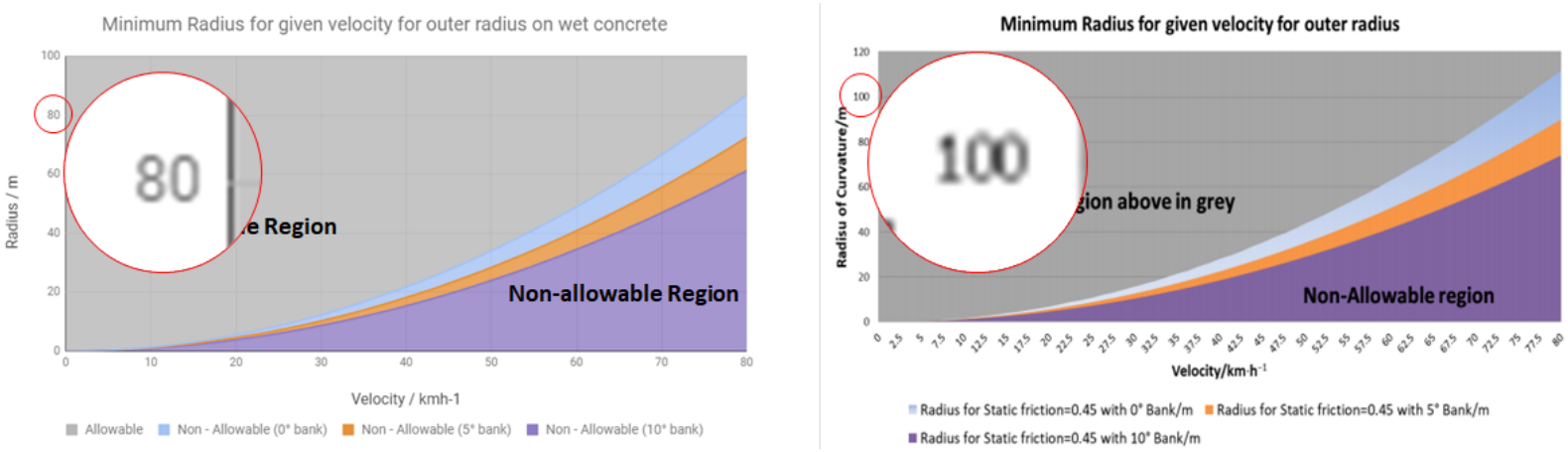


Figure 4.37. Left : New data for Allowable Non-Allowable Curve for Outer Radius when Slipping on Wet Concrete
Right : Old data for Allowable Non-Allowable Curve for Outer Radius when Slipping on Wet Concrete

As seen in the above figure, when comparing similar conditions, the shape is as expected, similar to previous data but the new data requires a lower radius of curvature to accommodate for higher speeds. The reason the new data is more desirable over the previous is due to the fact that the detection system is meant to detect any damage done to the CCD, so the system should only notify that there is problems with the CCD when it is either fully compressed or detaching, not when it is starting to be compressed. The old data however, is still useful when the calculations have to be conservative and no compression at all to the CCD is wanted.

4.5.2 Toppling

Lastly, the plotting of the allowable non-allowable curve for toppling is shown below.

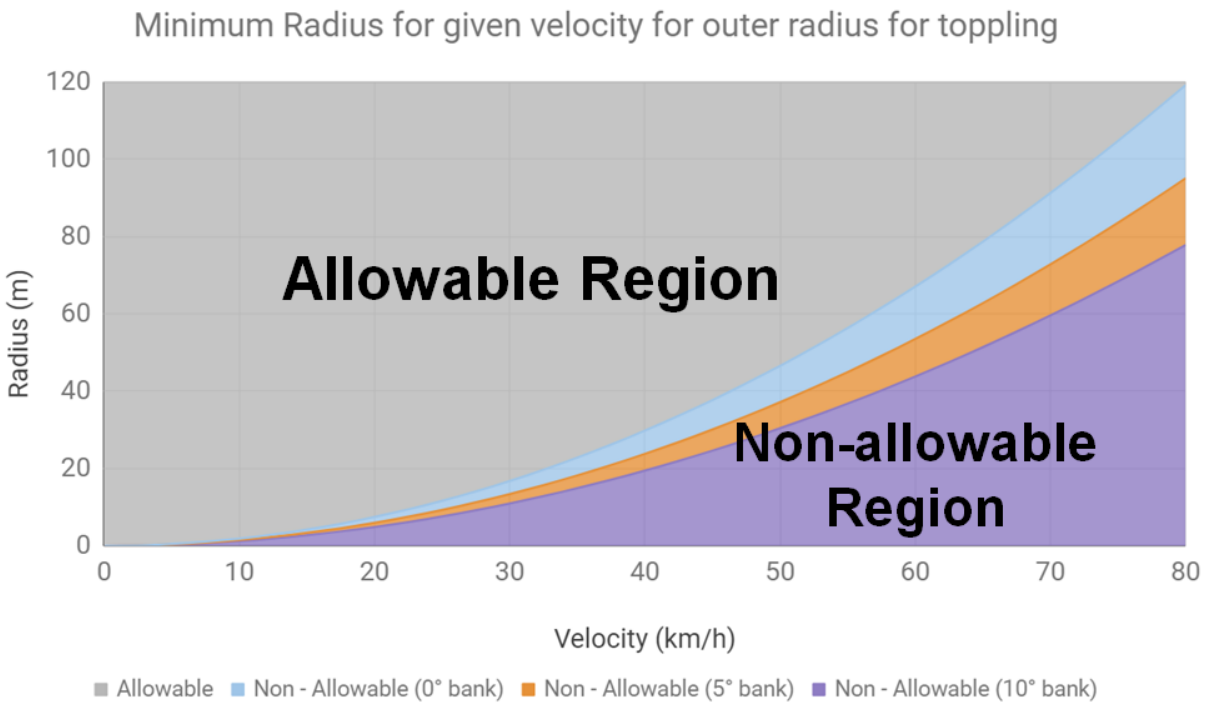


Figure 4.38. Data for Allowable Non-Allowable Curve for Outer Radius when Toppling at 0°,5° and 10° banks

For toppling, the criteria has been set that, the moment reaction forces on the inner wheel reaches 0, it is not allowed. This is conservative and is a good measure as even the possibility that it is lifted off the ground by 1° is too high of a risk for the LRT. While this curve will not be able to show how much compression or extension the CCD is undergoing when it is toppling, it is able to indicate that it is going to topple and that is enough for it to be counted as a problem. With this, it can be included as well eventually into the detection algorithm to determine if the train is going above operational limits.

4.5.3 Combination of all plots

With so many different plots, it is important and vital to know, exactly, which one is the true driving factor for it to be non-allowable. That is why time was spent to combine all the factors together in a manner that will allow all the different data to be compared and find the main factor. Show in in figure 4.37 is the combination of all data to find the overall non-allowable range.

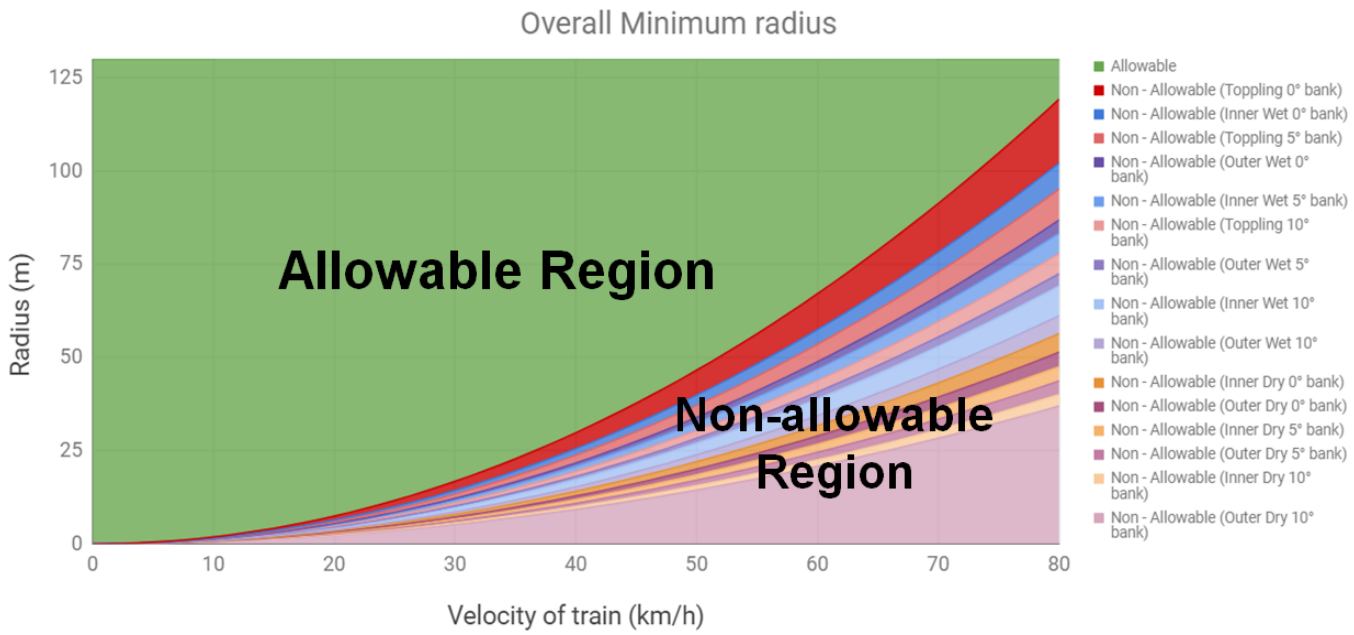


Figure 4.39. Combined Data of all the allowable non-allowable curve

As can be seen, the main driving factor that will determine the allowable region is actually the toppling situation at 0° bank. Interestingly, toppling is the factor that will happen the fastest even with 10° bank it still happens before the a 5° bank of wet slipping. This shows that the toppling will need extra caution in the future when being implemented into the detection system. Currently, it has not been implemented due to a constraint in time to achieve it.

With this all these information, it is now possible to relate the analysis to the detection program. For the analysis of centrifugal forces and its related measurements, it should be noted that calculating all these has allowed the building of a framework that allows these factors to be easily calculated in the future. Thus, if new information that will allow the author to improve the accuracy of these calculation are present, they can just be as easily added.

4.6 Relating analysis to detection program

With the concept of turning and centrifugal forces understood, the analysis would need to be related back to the detection program for it to be able detect undesirable situations when the LRT train is making a turn in addition to the usual fault detection.

The first challenge that was faced was the lack of data. To properly develop an algorithm for detecting the effects of centrifugal forces, the following information will need to be determined: speed of the train, radius of curvature, coefficient of friction, and

mass of the train. The speed of the train may vary over time, the mass will also vary depending on the number of passengers, the radius of curvature of the track may not be the same throughout the entire track and the coefficient of friction might change due to the weather. Another potential factor that may affect the results is the banks. As there are no banks on the tracks that are straight, the banks will be need to appear gradually during a turn. This may affect the results as the analysis that were made are based on constant banks. In order to build a basic framework for the detection algorithm, many of the mentioned factors were predetermined and kept constant.

The values are as shown:

Speed of train: 70 km/h
Mass of train: 21207 kg
Radius of curvature: 80 m
Coefficient of friction: 0.45 (Rubber-Wet concrete)

With these information, a new algorithm to detect centrifugal forces was made. This algorithm is able to detect the presence as well as the severity of the centrifugal force on the sensor.

4.6.1 Classifying the detected results for new algorithm

The algorithm was expected to detect mainly 3 types of scenarios:

1. Centrifugal forces detected but it is of reasonable values
2. Centrifugal forces detected but its value exceeds the expected range
3. No centrifugal forces detected.

The first scenario is what is expected for a healthy track as there should be and only be normal centrifugal forces acting on the train and the sensor when the train is making a turn under normal circumstances. Hence by running a healthy data through this algorithm, scenario one should be obtained and if not, it signifies that the algorithm is incorrect or has flaws.

4.6.2 Simulating Healthy data for a curved track

To test the algorithm, a set of healthy data with centrifugal forces is needed. The capacitance change for a normal centrifugal force at the predetermined conditions was obtained from the analysis. A normal centrifugal force would cause a capacitance change of around 66 - 91pF where 66pF is the capacitance change for the calculated K value and 91pF for the scaled K value for the Guide wheels. The algorithm is then written to detect such changes in capacitance within the given tag range. If it is

detected, the program will print out a message indicating that the turn is normal. The algorithm would also detect for any value that exceeds the range of capacitance and would mark and print it as abnormal forces. Lastly, if no centrifugal forces is detected, the algorithm will also print a suitable message to alert the user. A simulation was carried out to produce data with centrifugal forces. The sample outputs are shown below.

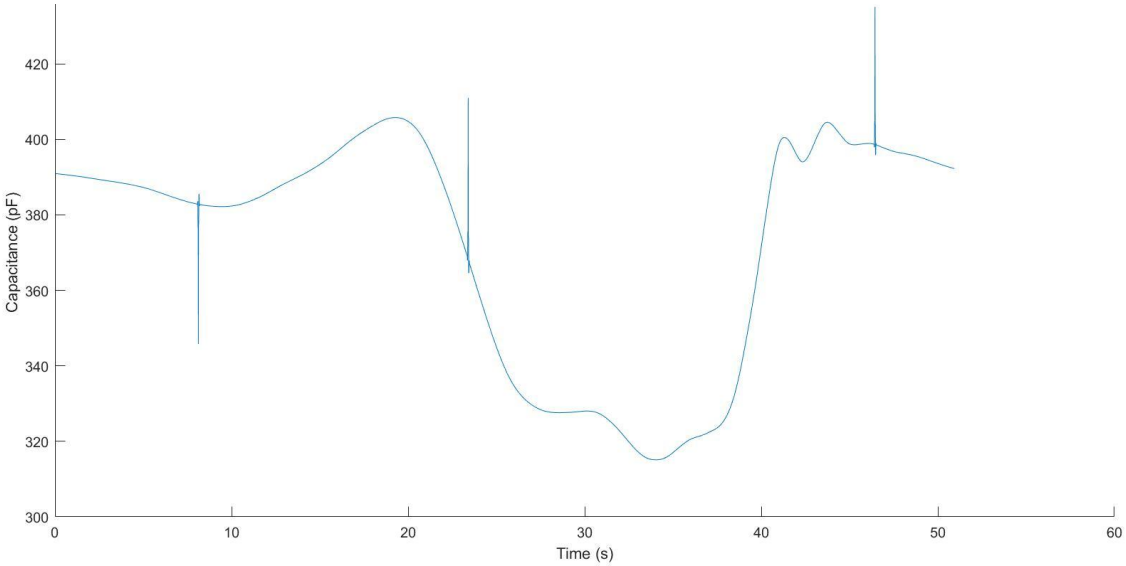


Figure 4.40 Graph for simulated centrifugal forces

```
Centrifugal forces detected from Tag 4 to Tag 5 as expected
```

Figure 4.41 Output for running simulated data

As can be seen from the figures above, the program is able to detect the centrifugal forces between the given tag range (Tag 4 - Tag 5 which is 20 - 40 sec).

Another simulation of centrifugal forces with deliberate abnormal forces added to it was produced and ran through the program and the results are shown below.

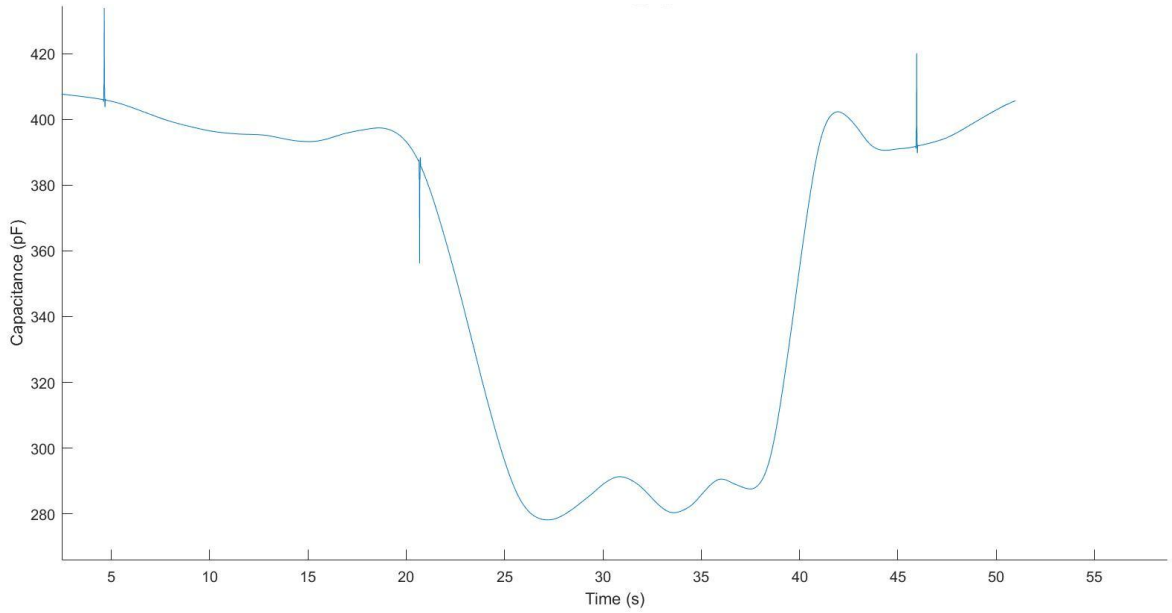


Figure 4.42 Graph of simulated centrifugal forces with abnormal forces

```
Abnormal Centrifugal forces from Tag 481 to Tag 778
```

Figure 4.43 Output for running simulated data

To further test the accuracy of the algorithm, a straight healthy graph was passed through the program.

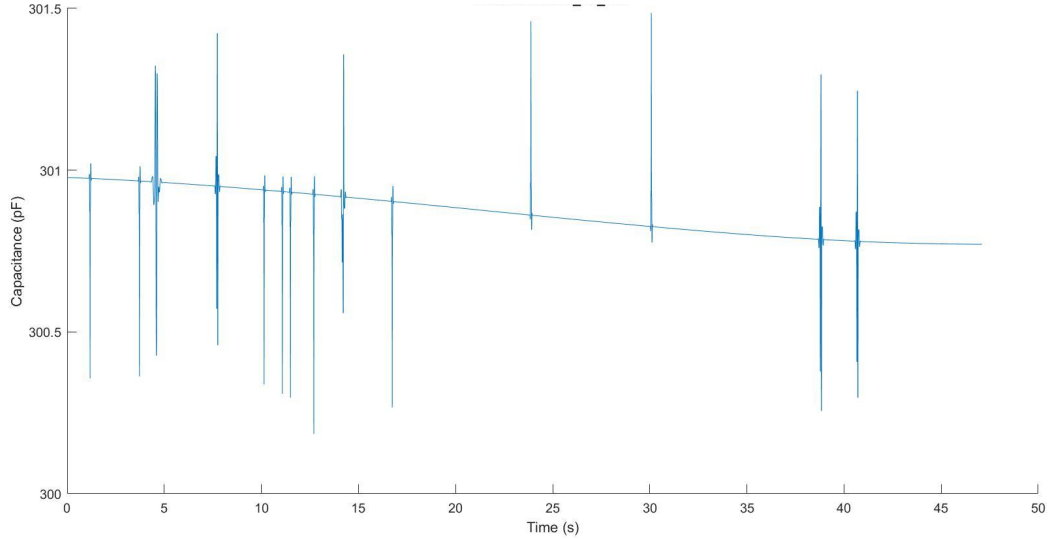


Figure 4.44 Straight healthy graph without centrifugal force

```
No centrifugal forces detected although expected
```

Figure 4.45 Output of running healthy data

The figures above shows that the algorithm is able to detect scenario three. As the current sensor is unable to detect any change in capacitance due to it slacking at around 280 pF, simulation of full compression of CCD is not achievable and it can either be done through manual alteration of data or simulation of new sensors that can accommodate the length of the CCD.

With the current algorithm tested to be working, it can be now used on the sample data. However, one limitation is that all the predetermined conditions must remain constant throughout the turn as any changes will affect the accuracy of the detection algorithm.

4.6.3 Combining curved detection with main program

As the locations of the RFID tags are known, the start and end of a curve on the track can be determined using tag numbers. These tag numbers determine the range at which the algorithm to detect faults for curved track is used.

```
%Normal fault detection
detect = Detect(y_capacitance,yhealthy); %Enter unhealthy capacitance and healthy capacitance as arguments
[value_counter,value] = detect.Wear(tagNumberforUnhealthySensor,tagNumberforHealthySensor,y_capacitance,yheal
detect.Step(tagNumberforUnhealthySensor, y_capacitance,tagNumberforHealthySensor,yhealthy); %Enter tag number
detect.consistentWear(value_counter, value); %Enter value_counter and value as arguments
detect.centriForces(tagNumberforUnhealthySensor,y_capacitance); %Enter tag number for Unhealthy Sensor and ur

%For curves only
turnTagObject = turnTag('Tag 389', 'Tag 778', tagNumberforUnhealthySensor, tagNumberforHealthySensor);%Input
curveHealthy = detectCurve(turnTagObject,y_capacitance,yhealthy);
curveHealthy.Wear(tagNumberforUnhealthySensor,tagNumberforHealthySensor,y_capacitance,yhealthy);
curveHealthy.centriForces(tagNumberforUnhealthySensor,tagNumberforHealthySensor,y_capacitance,yhealthy);
```

Figure 4.46 Code in main program for detecting faults

The current program can only work for one curve, multiple curves would require multiple lines of the same code, which is not very practical. Hence, another improved algorithm can perhaps be developed in the future to accommodate multiple curves.

4.7 Further testing of RFID capabilities

The RFID was tested to see if any condition will affect the reading or detection range. The tests include metal interference and rain interference. This is necessary as the RFID will be attached to the LRT train which has a lot of metal in the area of attachment and on rainy weather, the rain might affect the reading range or speed of the RFID.

The first test was metal interference test. Before the test starts the range of detection was tested and it was found to be about 1 meter. A metal tray was used to cover the face of the antenna entirely to intentionally block the pathway of the RF signals sent out by the antenna. The tags were brought close to it and the program was monitored to

see if any tags were detected. The result showed that when covered by metal, the detection becomes unreliable; the tags will sometimes be detected but sometimes not.

Figure 4.47. Metal tray covering RFID antenna behind

Next was the rain test which was conducted using two methods. The first method was to use a shower head to simulate rain. A shower head was turned on and the RFID tag was placed behind the stream of water. The antenna was placed more than 1 meter away, then brought closer to the tag until it reaches 1 meter. The result is that the detection range was not affected and it was not unstable too.

The second method is to use the actual rain. The tag was secured onto a rod and was brought out of the window, 1 meter into the rain. The antenna was directed at the tag and it was able to detect the tag as well.

Figure 4.48. RFID tags placed out in the rain 1 meter from the antenna

These two tests are not very accurate and real test with the whole setup in the actual rain should be carried out in near future.

4.8 Stretchable Sensors Sampling Experiment

When working on the integration of the RFID and sensor data and also developing the program, there were a few errors and after a few analysis and testings, it was found that the number of data for the sensor was incorrect. For instance, a recording of 60 seconds at a sampling rate of 100 Hz should result in a file with approximately 6000 data. However, the number of data obtained after the 60-second test was around 120000. More tests were conducted to find out the reason for this huge difference in data.

The tests were conducted such that one parameter which may affect the reading of the sensor is varying while the other parameters were kept constant. The parameters include the distance between the sensor and the device (Phone) and the type of phone used. The results from multiple tests are more or less the same, with a slight difference between different phones. It was noticed that all the number of data recorded were about two times larger than the expected value.

	Distance (cm)	Time (s)	Sampling rate (Hz)	Expected Samples	Actual Samples	Increased Samples*
S7 Edge Test 1	50	60	100	6000	11 763	12 174

Figure 4.49. Initial test result of sensor

	Distance (cm)	Time (s)	Sampling rate (Hz)	Expected Samples	Actual Samples	Increased Samples*
S7 Edge Test 1	100	60	100	6000	11 879	12 088

Figure 4.50. Test result of sensor after changing distance

	Distance (cm)	Time (s)	Sampling rate (Hz)	Expected Samples	Actual Samples	Increased Samples*
Oppo R9 Test 1	50	60	100	6000	11 821	12 171

Figure 4.51. Test result of sensor after changing phone

The results raised suspicion on the sampling rate and thus the sampling rate was inspected. By default, the sampling rate should be 100 Hz, but the results lean towards

200 Hz. Hence the sampling rate was manually set to 100 Hz and tests were conducted again. The outcome is that the number of data is close to the expected value and it has proven that there is a bug in the Stretchsense app.

	Distance (cm)	Time (s)	Sampling rate (Hz)	Expected Samples	Actual Samples	Increased Samples*
Test 1	50	60	100	6000	5718	6083

Figure 4.52. Test result of sensor after manually setting sampling frequency

One thing that was discovered during the tests was that the data will be automatically appended onto the previous excel file when the user is prompted to choose between recording the data in a new file or append it onto the previous file that was last used for recording. This only happens after at least one recording has been done. Therefore, it is worth noting that the data recorded should be copied or backed up after every run.

4.9 Simulation under realistic conditions

As most of the framework has been established, it is necessary to test the system in a simulation that simulates the conditions that the system will go through. For example, an important thing to note is the train's operational speed, to match previous estimates 70 km/h is used. The plan is to place the tags 1 meter from each other, to do this the total length of the track will need to be known. With the LRT tracks being 10.7 km long, if there were to be a tag every meter, it will be roughly around 10 700 tags placed.

For the simulation, 1000 tags are generated to simulated 1 kilometer of track. Next, to allow for an easier time doing the test, current sensor data will be used. This is possible as, regardless of the speed and other factors, the stretchable sensors will still only be sampling at 100 Hz, the highest the current sensors can go. With this, it is calculated that for 50 seconds, the train goes ~972 meters and the time between each tag is roughly ~0.05 seconds. With the sensor sampling every 0.01 second, it would mean that each tag will have roughly 5 capacitance data.

This creates a problem, previously, the RFID is configured to only detect time up to an accuracy of seconds not milliseconds, for now with the generated data, the milliseconds will be included but further testing of the RFID capabilities will need to be done in the future to incorporate milliseconds into its data output.

Tag 969	Date :	20/7/2018	14:15:36.794
Reader_name:			
Tag 970	Date :	20/7/2018	14:15:36.846
Reader_name:			
Tag 971	Date :	20/7/2018	14:15:36.897
Reader_name:			
Tag 972	Date :	20/7/2018	14:15:36.949
Reader_name:			

Figure 4.53. RFID output generated to Tag 972 with milliseconds incorporated

With that, it is time to test the program to see if it runs and some expected problems arised. Firstly, the program was set to seconds but now it will need to run in milliseconds format so a slight change to the formatting of the time was done. The bigger problem was the step algorithm mentioned previously in section 4.3.2. As the step was related to the tags, with only 5 capacitance data set, it was close to impossible to have a mode that can be reflected in properly a step, additionally, because each tag only contains 5 capacitance data, the change in capacitance is actually seen as gradual. Shown below is an example.

1074	4627306	300.6		1109	4627656	309.4
1075	4627316	301.3		1110	4627666	310.5
1076	4627326	300.8		1111	4627676	310.9
1077	4627336	301.1		1112	4627686	311.8
1078	4627346	301.6		1113	4627696	312.4
1079	4627356	301.2		1114	4627706	312.5
1080	4627366	302.6		1115	4627716	313.9
1081	4627376	303.3		1116	4627726	313.9
1082	4627386	304.2	...	1117	4627736	315.2

Figure 4.54. Capacitance Data Output from sensor data 12 from data no. 1074 to 1117

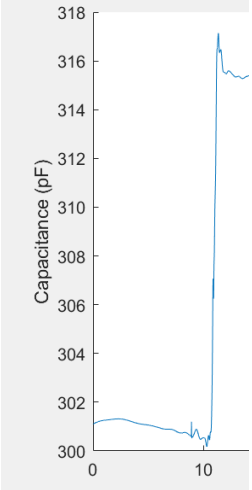


Figure 4.55. Corresponding step in capacitance to the data shown in Fig. 4.54.

As can be seen, while it is actually a pretty straight sharp step the data, if they are only confined to the 5 capacitance that the tags have, can actually be quite gradual and result in step not being detected. To counter this problem, the step algorithm had to be rewritten with new conditions. One such condition is changing it from being tag reliant to being time reliant, so it will actually compare the mode of the capacitance, instead of it being to the tag, but to a certain time interval. Through experimenting with the time interval window, it was discovered that the best time interval, without compromising any risk of missing an important step, is 3 seconds. With that, the program runs properly.

There are things that have yet to be tested and will need to be incorporated and those shall be talked about in the future works section.

4.10 Graphical User Interface

Development on a Graphical User Interface (GUI), is also currently underway. With this set up, one will easily be able to use the denoising and fault detection program. Just key in the file location (sensor and RFID datas), the speed the train is travelling at and also the location of the sensors on the train. If the requested data is available it will denoise the signal, detect the faults and display the wears etc.

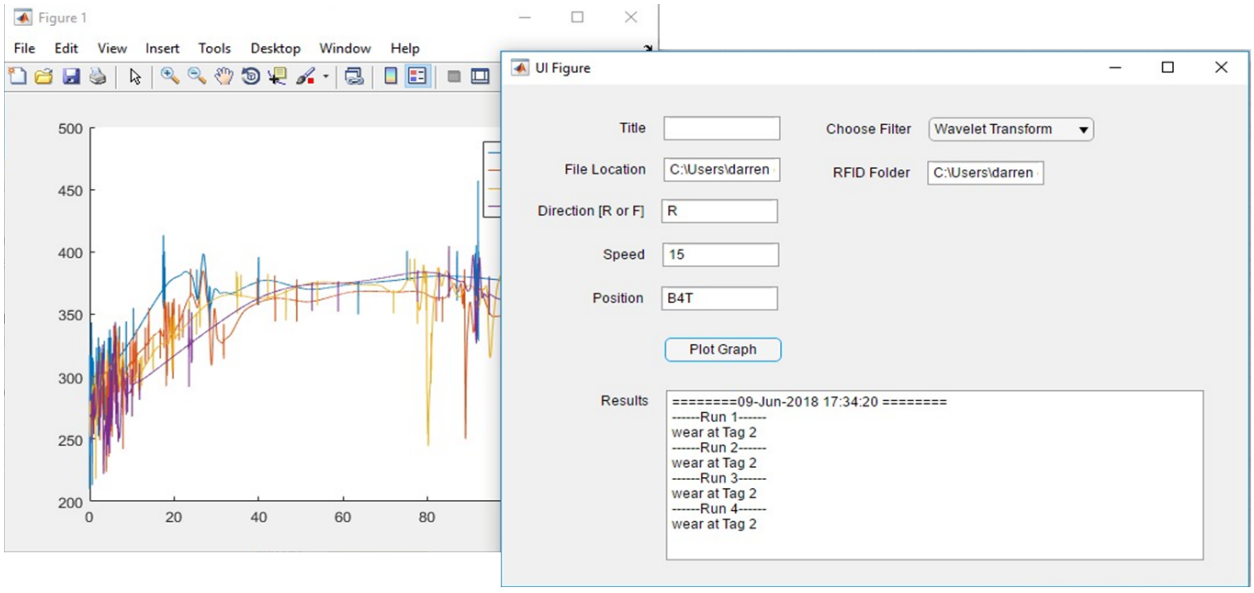


Figure 4.56 Showing the GUI and graph denoised at the back

Currently though, it is still extremely new in development, a prototype. With the main functions like checking for wear on the unhealthy track. The healthy and RFID datas are also prefixed, but this can be implemented relatively simply. The code behind is also

rather messy and thus, unsustainable. But once it is complete, it will ease usage of the program a lot.

4.11 Databases

Another work in progress is a database of healthy tracks data for when the train is turning. As different speeds when turning will cause a different centrifugal force acting on the LRT, varying the value of the capacitance, the data for unhealthy track and healthy track will not be able to be compared directly if they are of different speed. Hence another method for comparing data for turns is needed.

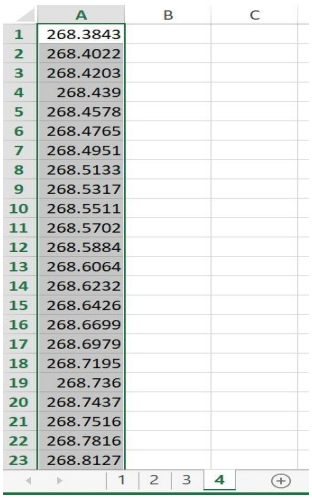
To do this, a database for the healthy track curve capacitance data for speed at every 10 km/h is created. This will be recorded in an excel file. The speed of the LRT will be derived from the time between each tag and the distance between them. The capacitance data from the LRT will then be compared to the capacitance data in the database that has the same speed as the sampled data. For example, if the speed of the sampled data is 50 km/h, the dataset for 50 km/h in the database will be retrieved and used as the healthy data. This will ensure that the number of tags per second and also the base capacitance of the curve are the same between the sampled track and the healthy track for fair and accurate comparison. That is the basic idea behind the database. It is currently in early stages of development and below are some things that have been done.

Firstly, extraction of the denoised sensor datas is mostly possible. The authors experimented with the data set, R_15_B4T for the extraction of denoised data. With the B4T being a multi signal data, 4 signal data in 1 file, there will be a need to store all 4 datas. With this 2 methods were devised and currently the authors are still deciding which one to go with.

1. To separate the 4 into different files like shown below and name them 1,2,3 and 4.

Figure 4.57 Storing and separating data into 4 different folders

2. Is to have them in the same file but different workbook.



	A	B	C
1	268.3843		
2	268.4022		
3	268.4203		
4	268.439		
5	268.4578		
6	268.4765		
7	268.4951		
8	268.5133		
9	268.5317		
10	268.5511		
11	268.5702		
12	268.5884		
13	268.6064		
14	268.6232		
15	268.6426		
16	268.6699		
17	268.6979		
18	268.7195		
19	268.736		
20	268.7437		
21	268.7516		
22	268.7816		
23	268.8127		

1 2 3 4

Figure 4.58 Storing data into different workbooks, seen at the bottom of the picture, 4 different workbook

For storing into different folders, the advantage would be the ability to have smaller file sizes as they are separated and allow the graph to be used easily, choosing the most clean graph out of all the signals, however, it will be difficult to put them all back together into one file again.

For storing into the same folder but different workbooks might possibly allow an easier time for the program to find, call and use the data it requires.

The pros and cons of each method is to be further evaluated or perhaps a better way can be devised in the future.

5. Conclusion

In conclusion, a program that analyses and automatically detects various faults on the LRT track was developed and the mechanics and effects of forces on the LRT train's Current Collector Device and the stretchable sensor were analysed.

The program utilized discrete wavelet transform to denoise raw data from the stretchable sensor. The stretchable sensor and RFID were integrated together through the MATLAB program by combining and relating their time values together and tagging each capacitance value with a tag number. An autonomous fault detector was made by comparing the capacitance value from the sensor with a healthy set of data and anomalies were identified as faults. The mechanics of the LRT train were studied and the effects of centrifugal forces were found and listed according to the velocity of the train. Conditions for toppling and slipping of the train were considered. A set of graphs was made to determine the various other limits of the LRT train at various conditions to build a framework for any future analysis being made on the LRT train. Lastly, the effects of centrifugal forces were related back to the detection program to allow it to detect any abnormal impact on the train's Current Collector Device.

With this program, it is easier for the untrained personnel to identify the location of the faults and what type of fault it belongs to so that they can make necessary changes and plans to fix the faulty track. It will also make inspection easier compared to the current method of inspection.

In summary, all the calculations and development of programs have made progress to the realisation of the proposed condition monitoring using soft and stretchable sensors. This project has shown that it is practical and possible to be developed for the LRT system.

6. Future Plans

While the program to detect and denoise the signal is mostly complete, there are still a few critical points to tackle. 1. The ability to handle varying speed will need to be implemented and 2. A Graphical User Interface will need to be implemented.

To tackle point 1 we suggest a database, work on this has already been underway. This database will contain all data for healthy track at varying speeds to allow detection to be done at various speeds. This database is important for turning as centrifugal force is different when the train is travelling at different speeds.

For point 2, a GUI will allow users to be able to easily use the program and it will also be cleaner. Similar to point 1, some work on this has been done but only a simple prototype has been fabricated. This port from the current command line program to GUI will greatly enhance the usability of this system.

Additionally, the mount that has been sent to SBS Transit for inspection and approval has not been approved due to a lack of communication from SBS Transit. There is a high possibility that a revision of the mount will be needed. Therefore, a revised and improved version of the mount will have to be designed and manufactured. This will result in various testing to be done on the revised mount. An example will be the amount of force that will be experienced by the mount or the size of it. All these will need to be done when redesigning of the mount.

Furthermore, once funding from SBS Transit has been approved, purchase of the new sensors and RFID will be acquired. This is important as there have been major changes made to the old design that was used on the test track.

Moreover, there will be a need to do testing on the RFID. As the RFID will need to be run continuously for an extended period of time, perhaps even an entire day, there might be issues such as overheating that might occur. Therefore, testing the capabilities of the RFID and how it works under long period of hours will need to be conducted. Perhaps a monitoring of the conditions of the RFID sensors can be proposed to ensure that it is running at its peak performance.

Last but not least, the RFID will need to be programmed to reach up to the milliseconds as seconds is too slow to detect accurately the location of the train with the train travelling at 70 km/h

7. Acknowledgement

To end it off, the authors would like to say that it has been an extremely fruitful journey this past year working on this project. We learned so much and were given a chance to tackle a real life issue through helping to create a structural health monitoring system for the LRT in Sengkang which will ease the workload of so many hardworking engineers and staff members working to detect faults on the LRT. Being attached to a project in National University of Singapore, working with students there, truly opened our eyes to how research is done in a top university in the world.

We would like to give our deepest appreciation to our supervisor, Dr Law Bee Khuan from Ngee Ann Polytechnic for the constant support, guidance and advice on this project as well as pointing us in the correct direction when needed. Dr Law was always willing to provide us with assistance even outside of working hours and the constant encouragement given really aided in helping us get through the tough and difficult challenges we faced during this project period. The project would not have gone as smoothly without her constant support.

We are also extremely appreciative and grateful to Prof. Adrian Koh from National University of Singapore for providing us with the opportunity to do our Final Year Project under him. Prof Adrian Koh was constantly giving guidance and assisting us on the project whenever possible. We would also like to thank Prof Adrian Koh for his patience when imparting his knowledge to us and assisting us with topics that we are not familiar with. Prof Adrian Koh would always accept our request to meet and discuss our project to provide any assistance he can and we are extremely grateful.

Additionally, we will also like to thank SBS Transit for their support on this project, Stretchsense for their assistance in the project and their stretchable sensors and Vanskee for loaning us the Ultra-High Frequency Radio Frequency-Identification set.

We would also like to thank Jie Jian from the National University of Singapore for his constant support and assistance when working together. He was always quick to respond to any difficulties we had and was always willing to correct and assist us. It was always a pleasure to be able to work with him ever since the start of the project and it was thanks to him that we are able to learn the required quickly during our Research and Development Module period. We would also like to thank Dhanish Musharraf Ubaidali and Lu Ziyou from NUS high school for their contribution and working with us on this project.

Finally, we would like to wish Rui Hang from National University of Singapore all the best on the project and the best of luck. Thank you very much for your hard work!

We wish everyone the best in their future endeavours.

8. References

[1] "SBS Transit profit grows 50% to $47m," The Straits Times, 12 February 2018. [Online]. Available: http://www.businesstimes.com.sg/companies-markets/sbs-transit-profit-grows-50-to-47m. [Accessed May 2018].

[2] "Timeline: Recent major breakdowns on Bukit Panjang LRT," The Straits Times, 28 September 2016. [Online]. Available: https://www.straitstimes.com/singapore/transport/timeline-recent-major-breakdowns-on-bukit-panjang-lrt. [Accessed May 2018].

[3] "Power fault causes 2-hour LRT disruption in Sengkang: SBS Transit," Channel News Asia, 16 February 2018. [Online]. Available: https://www.channelnewsasia.com/news/singapore/power-fault-causes-2-hour-lrt-disruption-in-sengkang-sbs-transit-9966340. [Accessed May 2018].

[4] "Fourier Transform," TheFourierTransform.com, 2010. [Online]. Available: http://www.thefouriertransform.com/. [Accessed March 2018].

[5] G. Dallas, "Wavelets 4 Dummies: Signal Processing, Fourier Transforms and Heisenberg," 14 May 2014. [Online]. Available: https://georgemdallas.wordpress.com/2014/05/14/wavelets-4-dummies-signal-processing-fourier-transforms-and-heisenberg/. [Accessed March 2018].

[6] "Introduction to Wavelet Families," Mathworks, [Online]. Available: https://www.mathworks.com/help/wavelet/gs/introduction-to-the-wavelet-families.html#f3-1009152. [Accessed April 2018].

[7] "Denoising Signals and Images," Mathworks, [Online]. Available: https://www.mathworks.com/help/wavelet/examples/denoising-signals-and-images.html. [Accessed March 2018].

[8] "Fast Fourier transform," Mathworks, [Online]. Available: https://www.mathworks.com/help/matlab/ref/fft.html?searchHighlight=fft&s_tid=doc_srchtitle. [Accessed April 2018].

[9] "Butterworth filter design," Mathworks, [Online]. Available: https://www.mathworks.com/help/signal/ref/butter.html. [Accessed April 2018].

[10] "Tyre Model Performance Test," 2003. [Online]. Available: http://tmpt.tuwien.ac.at/m-docu-1.pdf. [Accessed 18 May 2018].

[11] "Sengkang LRT," Google, [Online]. Available: https://www.google.com.sg/maps/search/sengkang+lrt/@1.3918995,103.8819064,15z?hl=en&authuser=0. [Accessed 26 May 2018].

9. Appendix

9.1 Matlab code

9.1.1 Main code (codeTest.m)

```
%{
Name : Lau Jie Jian, Yeoh Guan Wei, Darren Chan Yu Hao
National University of Singapore - Ngee Ann Polytechnic

Description : Main code for denoising as well as detecting faults

Last updated: 8/8/18
%}

samplingFreq = 100;  % for Butterworth Filter, can be changed depending on the sampling rate
User = 'G'; %GW/Darren [G/D]
%'folder' contains the address of the main folder that is used to keep all the relevant files

if User == 'G'
        folder = 'C:\Users\User\Desktop\Poly stuff\3.1\FYP';
elseif User == 'D'
        folder = 'C:\Users\darren chan\Desktop\School\Engineering Science Sem 3.1\Final Year Project';
end

%-------------------------START: Locating data files and plotting figures-------------

CombinedfindAverage = 0;  %set to 1 to find Average Data for combined data or 0 if not needed
findAverage = 0;  %set to 1 to find Average Data or 0 if not needed

%Locate file to be processed
%-------------------------For Test Track data----------------------------------------
fileloc = [folder, '\All Graphs-unfiltered\']; %Enter main folder address
file = File('F','10','A2B', fileloc); %Enter Direction[F/R], Speed, Position and main folder Directory as arguments
%data_UnhealthySensor = csvread(file.Locate); %read from the directory, 'Locate' returns full directory of file (user-defined class function)

%-------------------------For Mock track simulated data-----------------------------
data_UnhealthySensor = xlsread('C:\Users\User\Desktop\Poly stuff\3.1\FYP\Signal processing\stepsimulation2'); %consistentwear
%stepsimulation
%noisedetection2

%-------------------------For CCD simulated data-------------------------------------
%data_UnhealthySensor = xlsread([folder, '\SensorData\Sensor Data 12']); %8/12 - Step, 37 - Centrifugal Force
%-------------------------Choose Denoising method-------------------------------------
denoising = 2; % 0 - No filter, 1 - FFT, 2 - DWT, 3 - DWT w Threshold
curvedTrack = 'N';
%-------------------------Record output to a CSV file (For future references)---------
```

```
%outputFilename = [folder, '\History_', char(file.fileName), char( string(denoising) ), '.csv']; %name of file
%outputFilename = 'C:\Users\User\Desktop\Poly stuff\3.1\FYP\Bogus Data History\Latest output2.csv';
%diary(outputFilename); %diary function to record output (command window) into a CSV file

disp( '========' + string( datetime('now') ) + '========' ); %Header for separation between data

%Plots a single graph
file.Plot(denoising); %'Plot' generates figure, title and axes (user-defined class function)
hold on;

%-------------------------END: Locating files and plotting figures--------------------

[heightUnhealthySensor, widthUnhealthySensor] = size(data_UnhealthySensor);  %to find width to set for loop limits

%above portion take in raw data and time

%-----------------------------------START: RFID-------------------------------------

%below take in RFID tag number and time-------------------------------------------------

%Healthy RFID Data-----------------------------------------------------------------

%address_HealthyRFID =[folder,'\All Graphs-unfiltered\RF 2.csv'];
address_HealthyRFID =[folder, '\RFData\RF2.csv'];

object_rfidHealthy = RFID(address_HealthyRFID); %file directory as argument
tagNumberforHealthyRFID = object_rfidHealthy.Tag; %get the tag numbers
timeHealthyRFID = object_rfidHealthy.Time; %get the time

%Healthy Sensor Signal-------------------------------------------------------------
address_HealthySensor = [folder,'\Signal processing\healthy track']; %mock track
%address_HealthySensor = [folder,'\All Graphs-unfiltered\R_15_\R_15_B4T.csv']; %Test track
%address_HealthySensor = [folder, '\SensorData\Sensor Data 11']; %CCD

object_SensorHealthy = Sensor(address_HealthySensor,samplingFreq); %file directory and sampling rate as arguments
timeSensorHealthy = object_SensorHealthy.time; %get the time of the sensor data
heightSensorHealthy = object_SensorHealthy.height2; %get height of sensor data
%Assign RFID tag number to capacitance value in sensor signal
[tagNumberforHealthySensor, exceedDataHealthy] = StoR(object_rfidHealthy, heightSensorHealthy,timeSensorHealthy,
timeHealthyRFID,tagNumberforHealthyRFID);

%RFID data for Unhealthy Signal----------------------------------------------------
%address_UnhealthyRFID = [folder,'\All Graphs-unfiltered\RF 2.csv'];
address_UnhealthyRFID = [folder, '\RFData\RF2.csv'];

object_rfidUnhealthy = RFID(address_UnhealthyRFID); %file directory as argument
tagNumberforUnhealthyRFID = object_rfidUnhealthy.Tag; %get the tag number
timeUnhealthyRFID = object_rfidUnhealthy.Time; %get the time

%-----------------------------------END: RFID-----------------------------------------

%================================================================================
if denoising == 0 %No filter
```

```
        for i = 1:widthUnhealthySensor-1  %for loop to allow consecutive plotting of data with multiple columns

        column = data_UnhealthySensor(:,i+1);  %starts with the left most column of data after the time column
        column(column < 250) = mean(column);  %substitutes all values lower than expected value to mean value
        noOfSamples = length(data_UnhealthySensor);  %to find the length of the time column
        time = (1:noOfSamples)/samplingFreq;  %converts number of samples into time in seconds
        x = time;
    y_capacitance=column;

%       FindNaN=find(isnan(column));  %Find all elements that are NaN
%      AddNaN=length(FindNaN);  %Find the total number of NaN elements
%       y = column(~isnan(column));
%       for j=1:AddNaN
%       y(end + 1)=NaN;
%       end

        plot (x,y_capacitance);
        hold on

        end %end for
end %end if
%================================================================================

if denoising == 2 %Discrete Wavelet Transform [In use]*
        for i = 1:widthUnhealthySensor-1  %for loop to allow consecutive plotting of data with multiple columns
%------------------------START: DWT(Sample)-----------------------------------------

        disp('------Run ' + string(i) + '------'); %Sub-header to separate Runs
        column = data_UnhealthySensor(:,i+1);  %starts with the left most column of data after the time column
        column(column == 0) = [];  %substitutes zeroes in the column to Blanks
        noOfSamples = length(column);  %to find the length of the time column
        time = (1:noOfSamples)/samplingFreq;  %converts number of samples into time in seconds
        x_time = time;
        y_capacitance = column;
        noisy_y = column;
        dirDec = 'c';           % Direction of decomposition
        %level  = 7;            % Level of decomposition

        level=log(noOfSamples)/log(2);
        level=floor(level);% round N up

        wname  = 'sym8';   % Near symmetric wavelet db1, sym4 'bior2.4'
                %https://www.mathworks.com/help/wavelet/gs/introduction-to-the-wavelet-families.html#f3-1009152
        decCol = mdwtdec(dirDec,y_capacitance,level,wname); %performs wavelet decomposition at level LEV of each row
        [XDU,decDEN] = mswden('den',decCol,'sqtwolog','mln');%computes thresholds and performs denoising of 1-D signals using wavelets.
        y_capacitance = medfilt1(XDU);

        plot (x_time,y_capacitance);
        legendTitles{i} = ['Run ', int2str(i)];
        %legend(legendTitles);

%----------------------END: DWT(Sample)-----------------------------------------------
```

```
            %combine signal to be processed with RFID location for unhealthy signal
        [tagNumberforUnhealthySensor, exceedDataUnhealthy] =
StoR(object_rfidUnhealthy,noOfSamples,x_time,timeUnhealthyRFID,tagNumberforUnhealthyRFID);

%-------------------------START: DWT(Healthy)----------------------------------------

            column = object_SensorHealthy.healthydata(:,i+1);
            column(column == 0) = []; %replace any zeroes with blanks
            noOfSamples = length(column);
            time = (1:noOfSamples)/samplingFreq;  %converts number of samples into time in seconds
            x_time2 = time;
            yhealthy = column;


            dirDec = 'c';            % Direction of decomposition
            level=log(noOfSamples)/log(2); % Level of decomposition
            level=floor(level);% round N down

            wname  = 'sym8';   % Near symmetric wavelet db1, sym4 'bior2.4'
                    %https://www.mathworks.com/help/wavelet/gs/introduction-to-the-wavelet-families.html#f3-1009152
            decCol = mdwtdec(dirDec,yhealthy,level,wname); %performs wavelet decomposition at level LEV of each row

            [XDH,decDEN] = mswden('den',decCol,'sqtwolog','mln');%computes thresholds and performs denoising of 1-D signals using wavelets.

            yhealthy = medfilt1(XDH);

%----------For recording denoised healthy data to keep as database (In-progress so not in use), comment if not
needed-------------------------------------------

            %outputFilename = [folder,'\HealthyDataBase\SensorData\HealthyforSpeed',file.speed,'.csv']; %name of file
            %xlswrite(outputFilename,yhealthy,int2str(i),'A1');

%----------For plotting healthy graph, comment if not needed-------------------------

            %figure;
            %plot (x_time2,yhealthy);
            %legend('Step','Healthy');
            %title('Healthy data');
            %ylabel('Capacitance (pF)');
            %xlabel('Time (s)');
            %}
%-------------------------END: DWT(Healthy)----------------------------------------

%-------------------------START: Detecting Wear-------------------------------------

            %Normal fault detection
            detect = Detect(y_capacitance,yhealthy); %Enter unhealthy capacitance and healthy capacitance as arguments

           [value_counter,value] = detect.Wear(tagNumberforUnhealthySensor,tagNumberforHealthySensor,y_capacitance,yhealthy);
    %Enter tag number for Unhealthy Sensor, tag number for Healthy Sensor,unhealthy capacitance and healthy capacitance as arguments

            detect.Step(tagNumberforUnhealthySensor, y_capacitance,tagNumberforHealthySensor,yhealthy);
```

```
    %Enter tag number for Unhealthy Sensor, unhealthy capacitance, tag number for Healthy Sensor and healthy capacitance as arguments

            detect.consistentWear(value_counter, value);  %Enter value_counter and value as arguments

            detect.centriForces(tagNumberforUnhealthySensor,y_capacitance);
            %Enter tag number for Unhealthy Sensor and unhealthy capacitance as arguments

             %For curves only
             if curvedTrack == 'Y'
             turnTagObject = turnTag('Tag 389', 'Tag 778', tagNumberforUnhealthySensor, tagNumberforHealthySensor);
            %Input Start tag,End tag,tag number for Unhealthy Sensor and tag number for Healthy Sensor as arguments

            curveHealthy = detectCurve(turnTagObject,y_capacitance,yhealthy);

            curveHealthy.Wear(tagNumberforUnhealthySensor,tagNumberforHealthySensor,y_capacitance,yhealthy);

            curveHealthy.centriForces(tagNumberforUnhealthySensor,tagNumberforHealthySensor,y_capacitance,yhealthy);
             end

             %Exceeded Data Warning (When the time of the sensor exceeds the time of the RFID)
             if exceedDataHealthy == 1 || exceedDataUnhealthy == 1 %IF Either set of sensor data exceeds its RFID tags
             disp('Sensor data exceeded number of RFID tags');
             disp('Faults detected after last tag are considered to be at the last tag which is ' + string(tagNumberforHealthySensor(end)));
             end
%------------------------END: Detecting Wear----------------------------------------

            hold on

            end %end for
end %end if
%================================================================================

            hold off;
            diary off;
            clear;
```

9.1.2 Straight Track Detection code (Detect.m)

```
 %{
 Name : Lau Jie Jian, Yeoh Guan Wei, Darren Chan Yu Hao
 National University of Singapore - Ngee Ann Polytechnic


 Description : Detect Summary of this class goes here
               Detect various wear and forces-

 Last updated: 24/7/18
 %}
 classdef Detect
```

```
        properties
        capacitanceDiff_flag
        heightUnhealthy
        heightHealthy
        end

        methods
        function obj = Detect(y, yhealthy)

        obj.heightUnhealthy = length(y); %number of capacitance capacitanceDiff for sample
        obj.heightHealthy = length(yhealthy); %number of capacitance value for healthy
        end %end function

        function [capacitanceDiff_flag, capacitanceDiffUnhealthy] =
Wear(obj,tagNumberforUnhealthySensor,tagNumberforHealthySensor,y,yhealthy)
        faultCounter = 1; %counter for capacitanceDiff to be stored
        startPosHealthySensor = 1; %counter for healthy signal to prevent reading from the same capcitance for the same RFID tag
        % assumption of same sampling rate and speed for healthy and new
        % signal so that intermediate capacitance between each rfid tags
        % can be compared
        capacitanceDiff_flag = 0; %flag to indicate if the are any capacitance difference
            capacitanceDiffUnhealthy = 0; %array to store the capacitance diff for faults (if any)
            printFlagWear = 0; %flag to prevent printing multiple faults in the same tag
            faultTagStart = 0; %to indicate where the first fault is detected
            faultTagEnd = 0; %to indicate where the last fault is detected after detecting multiple faults
            prevTagIsFault = 0; %to indicate that there is a fault before the current tag
            maxHeightofSensor = 0;

            %Find lower limit
            if(obj.heightUnhealthy >= obj.heightHealthy)
            maxHeightofSensor = obj.heightHealthy;
            else
            maxHeightofSensor = obj.heightUnhealthy;
            end

            for unhealthyCounter = 1:obj.heightUnhealthy
            for healthyCounter = startPosHealthySensor:obj.heightHealthy
            tagFlag = strcmp(tagNumberforUnhealthySensor(unhealthyCounter,1),tagNumberforHealthySensor(healthyCounter,1));
            %compare the RFID tag number on signal vs healthy signal

            if tagFlag == 1
            %check for wear in the same location
            if (yhealthy(healthyCounter,1)+ 2)<y(unhealthyCounter,1) || (yhealthy(healthyCounter,1)- 2)>y(unhealthyCounter,1)
            % if deviation is greater than noise, it is a wear

                        capacitanceDiffUnhealthy(faultCounter,1) = y(unhealthyCounter,1)-yhealthy(healthyCounter,1);
                        %form an array of the difference in value
                        capacitanceDiff_flag = 1; %indicate that there is a fault

                        if (prevTagIsFault == 0)
                        if(faultTagStart == 0) %no fault yet
                        faultTagStart = unhealthyCounter; %record current index
                        prevTagIsFault = 1; %indicate that there is a fault previously
                        end
                        else
                        faultTagEnd = unhealthyCounter; %store current index as end of fault
```

```
                    end



                    if(faultTagStart ~= 0 && unhealthyCounter == maxHeightofSensor)
                    %disp ('wear from');
    %disp (string(tagNumberforUnhealthySensor(faultTagStart,1)) + ' to ' + string(tagNumberforUnhealthySensor(faultTagEnd,1)) + newline);
                    end

                    faultCounter = faultCounter+1;
                    startPosHealthySensor = startPosHealthySensor+1;
                    %shift the starting position by 1 to prevent it from comparing from the same capacitance value

                    if unhealthyCounter>1
      tagFlagWear = strcmp(tagNumberforUnhealthySensor(unhealthyCounter,1),tagNumberforUnhealthySensor((unhealthyCounter-1),1));
                   % prevent reptition of tag if wear is at the same tag

                    if tagFlagWear == 0 % at different tag number from prev
                      printFlagWear = 0; %reset the 'printed' flag
                    end
                    if tagFlagWear ==  0 || (printFlagWear == 0 && tagFlagWear == 1)
                    %diff tag from prev OR same tag but no prev faults printed
                    disp ('wear at');
                    disp (tagNumberforUnhealthySensor(unhealthyCounter,1));
                    printFlagWear = 1; %'printed' flag
                    end %end if tagFlag2 == 0
                    end %end if unhealthyCounter > 1
          else
                    prevTagIsFault = 0;
                    if(faultTagStart ~= 0 && faultTagEnd ~= 0)
                    %disp ('wear from');
    %disp (string(tagNumberforUnhealthySensor(faultTagStart,1)) + ' to ' + string(tagNumberforUnhealthySensor(faultTagEnd,1)) + newline);
                    faultTagStart = 0; %reset
                    end
          end %end if yhealthy < y
          break %break out of healthy for loop
          else
          startPosHealthySensor = healthyCounter;
          end %end if tagFlag == 1
          end %end for healthyCounter
          end %end for unhealthyCounter
end %end function

function Step(obj,tagNumberforUnhealthySensor, y, tagNumberforHealthySensor, yHealthy)
          timeInterval = 300; %time interval for mode (most common value) to be calculated and compared to
          numOfInterval = obj.heightUnhealthy/timeInterval; %find number of such interval to loop through
          startingPosUnhealthy = 1;
          startingPosHealthy = 1;

          for timeRange = 1:numOfInterval
          %-----------START: Getting Mode Unhealthy----------------------------------------------
          capforTagUnhealthy = 0;
          timeCounterUnhealthy = 1;

          for unhealthyCounter = startingPosUnhealthy:obj.heightUnhealthy-1
```

```
currentTimeIndex = unhealthyCounter/timeInterval;
        if(floor(currentTimeIndex) ~= currentTimeIndex) %Not end of time interval yet
capforTagUnhealthy(timeCounterUnhealthy,1) = y(unhealthyCounter,1); %store capacitance values that are within the time interval
      timeCounterUnhealthy = timeCounterUnhealthy + 1;
else %Reached end of time interval
capforTagUnhealthy(timeCounterUnhealthy,1) = y(unhealthyCounter,1);
timeCounterUnhealthy = timeCounterUnhealthy + 1;
startingPosUnhealthy = unhealthyCounter+1; %set new starting position for next loop
break; %break once end of 1 interval
end
end
modeOfCapUnhealthy = mode(round(capforTagUnhealthy)); %find mode of capacitance within the time interval,
%capacitance is rounded off to nearest integer to neglect changes smaller than 1pF to obtain more constant values

%------------END: Getting Mode Unhealthy-----------------------------------------------

%------------START: Getting Mode Healthy-----------------------------------------------
capforTagHealthy = 0;
timeCounterHealthy = 1;

for healthyCounter = startingPosHealthy:obj.heightHealthy-1
currentTimeIndex = healthyCounter/timeInterval;
if(floor(currentTimeIndex) ~= currentTimeIndex)
capforTagHealthy(timeCounterHealthy,1) = yHealthy(healthyCounter,1);
timeCounterHealthy = timeCounterHealthy + 1;
else
capforTagHealthy(timeCounterHealthy,1) = yHealthy(healthyCounter,1);
        timeCounterHealthy = timeCounterHealthy + 1;
startingPosHealthy = healthyCounter+1;
break;
end
end
modeOfCapHealthy = mode(round(capforTagHealthy));
%------------END: Getting Mode Healthy-----------------------------------------------

%------------START: Detecting Step-----------------------------------------------------
%---------------Detect step: Unhealthy-----------------------------------------------
i = 0;
outlierCounter = 0;
for tagSelector = 1:timeCounterUnhealthy-1
 if ((capforTagUnhealthy(tagSelector,1) - 14) > modeOfCapUnhealthy || (capforTagUnhealthy(tagSelector,1) + 14) <
modeOfCapUnhealthy) %can be less than TOO
outlierCounter = outlierCounter + 1; %count number of values that deviate from mode
if outlierCounter == 1 %the instance an outlier is detected
        stepLoc = tagSelector; %record the index
end
if tagSelector == timeCounterUnhealthy-1 %when the time range ends
    i = i + 1; %counts number of steps when data go from normal to abnormal but dun come back within time range
        stepLocationArrUnhealthy(i,1) = (timeRange-1)*timeInterval + stepLoc; %mark start of outlier as location of step
end
else
if outlierCounter > 0 %got step
    i = i + 1; %counts number of steps when data go from abnormal to normal
        outlierCounter = 0; %reset to detect next step if any
        stepLocationArrUnhealthy(i,1) = (timeRange-1)*timeInterval + tagSelector; %mark current tag as step location
end
```

```
                end

            end
            numOfFaultsUnhealthy = i;
            %---------------Detect step: Healthy-------------------------------------------------------------------
        j=0;
            outlierCounter = 0 ;
            for tagSelector = 1:timeCounterHealthy-1
            if ((capforTagHealthy(tagSelector,1) - 15) > modeOfCapHealthy || (capforTagHealthy(tagSelector,1) + 15) < modeOfCapHealthy)
%can be less than TOO
            outlierCounter = outlierCounter + 1;
            if outlierCounter == 1
                    stepLoc = tagSelector;
            end
            if tagSelector == timeCounterHealthy-1
                j = j + 1; %counts number of steps when data go from normal to abnormal but dun come back within time range
                    stepLocationArrHealthy(j,1) = (timeRange-1)*timeInterval + stepLoc;
            end
            else
                  if outlierCounter > 0
                j = j + 1; %counts number of steps when data go from abnormal to normal
                    outlierCounter = 0;
                    stepLocationArrHealthy(j,1) = (timeRange-1)*timeInterval + tagSelector;
            end
            end
            end
            numOfFaultsHealthy = j;
            %---------------Compare Number of steps between Unhealthy and Healthy-----------------------------------------------
            if numOfFaultsUnhealthy > numOfFaultsHealthy

            if numOfFaultsHealthy ~= 0 %There are steps in Healthy, Not all steps detected in unhealthy are steps
            for tagCheckerHealthy = 1:numOfFaultsHealthy
                    for tagCheckerUnhealthy = 1:numOfFaultsUnhealthy
                    if stepLocationArrHealthy(tagCheckerHealthy,1) ~= stepLocationArrUnhealthy(tagCheckerUnhealthy,1)
                      disp('Step roughly at');
                    disp(tagNumberforUnhealthySensor(stepLocationArrUnhealthy(tagCheckerUnhealthy,1),1));
                    end
                    end
            end
                  else %No steps in healthy, all steps detected at unhealthy are indeed steps
            for printTagWithFault = 1:numOfFaultsUnhealthy
                    disp('Step roughly at');
                    disp(tagNumberforUnhealthySensor(stepLocationArrUnhealthy(printTagWithFault,1),1));
            end
            end
            end
            end %end timeRange
    end %end function

    function consistentWear(obj, capacitanceDiff_flag, capacitanceDiff)
            if capacitanceDiff_flag == 1
            % check for consistent wear
        M = mean(capacitanceDiff);
        k = length(capacitanceDiff); %or k=k-1;
            counter = 0;
            for i3 = 1:k
```

```
        if (capacitanceDiff(i3,1)>(M -3)) && (capacitanceDiff(i3,1)<(M + 3)) %check if value difference is within the mean of the value +-3
        counter = counter+1;
        end
        end
          if counter>(obj.heightHealthy*8/10)%use heightUnhealthy as benchmark because heightUnhealthy indicate the entire signal
        disp ('consistent wear of approximately');
        disp (string(M) + 'pF or ' + string(M/3) + 'mm' + newline);
        end
        end
end %end function

%only used for FFT
function butterStep(obj, x2, noisy_y)
        printFlag3 = 0;
        %check for step when butterworth filter is applied
        for j = 2:obj.heightUnhealthy

        tagFlag3 = strcmp(x2(j,1),x2((j-1),1)); %compare tag numbers
        if tagFlag3 == 0 % at different tag number
        printFlag3 = 0; %reset the 'printed' flag
        end

          if ((noisy_y(j,1)-15) > noisy_y((j-1),1))

        if tagFlag3 == 0 || (printFlag3 == 0 && tagFlag3 == 1) %if different from prev tag then print, if same but not printed,print too
        disp ('step at');%use noisy_y to check on itself to see if there is a step
        disp (x2(j,1));
        printFlag3 = 1; %'printed' flag
        end
        end %end if
        end %end for j loop
end %end function

function centriForces(obj,tagNumberforUnhealthySensor,noisy_y)
        printFlagOuterCentriForce = 0; %flags to signify printing of error output where 1 means printed
        printFlagInnerCentriForce = 0;
        %use unfiltered capactiance so that a sharp spike can be detected
        for j = 2:obj.heightUnhealthy

        tagFlag4 = strcmp(tagNumberforUnhealthySensor(j,1),tagNumberforUnhealthySensor((j-1),1)); %check if same tag as prev
        if tagFlag4 == 0 %if diff means new tag number
        printFlagOuterCentriForce = 0; %reset 'printed' flag at every new tag number
        end

        if(noisy_y(j,1) <= 250)% 250 being the resting capacitance of the sensor
        if tagFlag4 == 0 || (tagFlag4 == 1 && printFlagOuterCentriForce == 0)
        disp ('Outer centrifugal forces acting at');
        disp (tagNumberforUnhealthySensor(j,1));
        printFlagOuterCentriForce = 1; %'printed' flag
        end
        end

        tagFlag5 = strcmp(tagNumberforUnhealthySensor(j,1),tagNumberforUnhealthySensor((j-1),1)); %check if same tag as prev
        if tagFlag5 == 0 %if diff means new tag number
                printFlagInnerCentriForce = 0; %reset 'printed' flag at every new tag number
        end
```

```
            if(noisy_y(j,1) >= 390)% 390 being the maximum capacitance of the sensor when CCD fully extends
            if tagFlag5 == 0 || (tagFlag5 == 1 && printFlagInnerCentriForce == 0)
            disp ('inner centrifugal forces acting at');
            disp (tagNumberforUnhealthySensor(j,1));
            printFlagInnerCentriForce = 1; %'printed' flag
            end
            end %end if
            end %end for j
    end %end function

        end %end methods

end %end class
```

9.1.3 Curve Track Detection code (detectCurve.m)

```
%{
Name : Lau Jie Jian, Yeoh Guan Wei, Darren Chan Yu Hao
National University of Singapore - Ngee Ann Polytechnic


Description : Detect various wear and forces when turning-

Last updated: 24/7/18
%}

classdef detectCurve

        properties
        heightUnhealthy
        heightHealthy
        curveStartHealthy
        curveEndHealthy
        curveStartUnhealthy
        curveEndUnhealthy
        end

        methods

        function obj = detectCurve(tagObject,y, yhealthy)
        obj.heightUnhealthy = length(y); %number of capacitance capacitanceDiff for sample
        obj.heightHealthy = length(yhealthy); %number of capacitance value for healthy
        obj.curveStartHealthy = tagObject.curveStartHealthy; %index for start of a curve for healthy track
        obj.curveEndHealthy = tagObject.curveEndHealthy; %index for end of a curve for healthy track
        obj.curveStartUnhealthy = tagObject.curveStartUnhealthy; %index for start of a curve for unhealthy track
        obj.curveEndUnhealthy = tagObject.curveEndUnhealthy; %index for end of a curve for unhealthy track
    end %end constructor

    function [capacitanceDiff_flag, capacitanceDiffUnhealthy] = Wear(obj,tagNumberforUnhealthySensor,tagNumberforHealthySensor,y,yhealthy)
            %semi-completed (copy-pasted from straights detection, only changed the range of detection to the turn tags)
            faultCounter = 1; %counter for capacitanceDiff to be stored
            startPosHealthySensor = obj.curveStartHealthy;
```

```
%counter for healthy signal to prevent reading from the same capcitance for the same RFID tag

% assumption of same sampling rate and speed for healthy and new
% signal so that intermediate capacitance between each rfid tags can be compared
capacitanceDiff_flag = 0; %flag to indicate if the are any capacitance difference
capacitanceDiffUnhealthy = 0; %array to store the capacitance diff for faults (if any)
printFlagWear = 0; %flag to prevent printing multiple faults in the same tag
faultTagStart = 0; %to indicate where the first fault is detected
faultTagEnd = 0; %to indicate where the last fault is detected after detecting multiple faults
prevTagIsFault = 0; %to indicate that there is a fault before the current tag
maxHeightofSensor = 0;

if(obj.curveEndUnhealthy >= obj.curveEndHealthy)
maxHeightofSensor = obj.curveEndHealthy;
else
maxHeightofSensor = obj.curveEndUnhealthy;
end

for unhealthyCounter = obj.curveStartUnhealthy:obj.curveEndUnhealthy
for healthyCounter = startPosHealthySensor:obj.curveEndHealthy
tagFlag = strcmp(tagNumberforUnhealthySensor(unhealthyCounter,1),tagNumberforHealthySensor(healthyCounter,1));
%compare the RFID tag number on signal vs healthy signal

if tagFlag == 1
%check for wear in the same location
  if (yhealthy(healthyCounter,1)+ 2)<y(unhealthyCounter,1) || (yhealthy(healthyCounter,1)- 2)>y(unhealthyCounter,1)
  % if deviation is greater than noise, it is a wear

  capacitanceDiffUnhealthy(faultCounter,1) = y(unhealthyCounter,1)-yhealthy(healthyCounter,1);
  %form an array of the difference in value

        capacitanceDiff_flag = 1; %indicate that there is a fault

        if (prevTagIsFault == 0)
        if(faultTagStart == 0) %no fault yet
        faultTagStart = unhealthyCounter; %record current index
        prevTagIsFault = 1; %indicate that there is a fault previously
        end
          else
        faultTagEnd = unhealthyCounter; %store current index as end of fault
        end

        if(faultTagStart ~= 0 && unhealthyCounter == maxHeightofSensor)
        disp ('wear from');
        disp (string(tagNumberforUnhealthySensor(faultTagStart,1)) + ' to ' + string(tagNumberforUnhealthySensor(faultTagEnd,1)) +
newline);
        end

        faultCounter = faultCounter+1;
        startPosHealthySensor = startPosHealthySensor+1;
        %shift the starting position by 1 to prevent it from comparing from the same capacitance value

        if unhealthyCounter>1
        tagFlagWear =
strcmp(tagNumberforUnhealthySensor(unhealthyCounter,1),tagNumberforUnhealthySensor((unhealthyCounter-1),1)); % prevent reptition of tag if
wear is at the same tag
```

```
                        if tagFlagWear == 0 % at different tag number from prev
                        printFlagWear = 0; %reset the 'printed' flag
                        end
                        if tagFlagWear ==  0 || (printFlagWear == 0 && tagFlagWear == 1) %diff tag from prev OR same tag but no prev faults printed
                        disp ('wear at');
                        disp (tagNumberforUnhealthySensor(unhealthyCounter,1));
                         printFlagWear = 1; %'printed' flag
                        end %end if tagFlag2 == 0
                        end %end if unhealthyCounter > 1
                else
                        prevTagIsFault = 0;
                    if(faultTagStart ~= 0 && faultTagEnd ~= 0)
                        disp ('wear from');
                        disp (string(tagNumberforUnhealthySensor(faultTagStart,1)) + ' to ' + string(tagNumberforUnhealthySensor(faultTagEnd,1)) +
newline);
                        faultTagStart = 0; %reset
                        end
                end %end if yhealthy < y
                break %break out of healthy for loop
                else
                  startPosHealthySensor = healthyCounter;
                end %end if tagFlag == 1
                end %end for healthyCounter
                end %end for unhealthyCounter
        end %end function

        function Step(obj,tagNumberforUnhealthySensor,tagNumberforHealthySensor,y,yhealthy)
                %Not started
        end

        function centriForces(obj, tagNumberforUnhealthySensor,tagNumberforHealthySensor, y, yhealthy)
                centriCounter = 0; %count number of instances that indicates normal centri forces
                abnormalCentriCounter = 0; %count number of instances that indicates abnormal centri forces
                flagAbnormal = 0; %count number of abnormal forces in each tag type
                flagNormal = 0; %count number of normal forces in each tag type
                flagNoCentri = 0; %count number of missing forces in each tag type
                tagCounter = 0; %count number of tag types
                prevAbnormal = 0;% store previous value of flagAbnormal
                startAbnormal = 0;
                for tagSelector = obj.curveStartHealthy:obj.curveEndHealthy %from start to end of curve

                if (yhealthy(tagSelector,1)+91) < yhealthy(obj.curveStartHealthy-1,1) %91pF is the max change in cap for centri at 70km/h (scaled)
                      abnormalCentriCounter = abnormalCentriCounter + 1;
                elseif (yhealthy(tagSelector,1)+66) < yhealthy(obj.curveStartHealthy-1,1) %66pF is the min change in cap for centri at 70km/h (calc.)
                centriCounter = centriCounter + 1;
                end
                if ~strcmp(tagNumberforHealthySensor(tagSelector,1), tagNumberforHealthySensor(tagSelector+1,1)) %at the end of each tag type
                tagCounter = tagCounter + 1; %count types of tags
                  if abnormalCentriCounter >= 3 %3 is the majority in each tag type to be considered abnormal centrifugal
                flagAbnormal = flagAbnormal + 1; %increment number of abnormal forces
                abnormalCentriCounter = 0; %reset
                if flagAbnormal == 1
                        startAbnormal = tagSelector;
                end
                prevAbnormal = flagAbnormal;
```

```
                elseif centriCounter >= 3 %3 is the majority in each tag type to be considered normal centrifugal
                flagNormal = flagNormal + 1;
                centriCounter = 0;
                else %none of the forces above are detected
                flagNoCentri = flagNoCentri + 1;
                if tagSelector == obj.curveEndHealthy %End of turn
                        %TBD
                end
                end %end if-else type of forces
                if prevAbnormal == flagAbnormal || tagSelector == obj.curveEndHealthy
                endAbnormal = tagSelector;
                end
                end %end if diff tag
                end %end for loop

                if flagNormal == tagCounter %all forces recorded are normal
                   disp('Centrifugal forces detected from ' + string(tagNumberforHealthySensor(obj.curveStart)) + ' to ' +
    string(tagNumberforHealthySensor(obj.curveEnd)) + ' as expected');
                elseif flagNoCentri == tagCounter %all forces recorded are missing
                disp('No centrifugal forces detected although expected');
                elseif flagAbnormal == tagCounter %all forces recorded are abnormal
                disp('Abnormal Centrifugal forces detected from ' + string(tagNumberforHealthySensor(obj.curveStart)) + ' to ' +
    string(tagNumberforHealthySensor(obj.curveEnd)));
                elseif flagAbnormal > 0 %only some part has abnormal forces
                disp('Abnormal Centrifugal forces from ' + string(tagNumberforHealthySensor(startAbnormal))+ ' to ' +
    string(tagNumberforHealthySensor(endAbnormal)));
                else %some part has normal forces while some are missing
                disp('Normal centrifugal forces detected on ' + string(round(flagNormal/tagCounter*100,1)) + '% of the curved track');
                end %end if-else msg to print

                disp('For Healthy Track');
                disp(' ');
                %disp('Abnormal Centrifugal forces detected at ' + string(tagNumberforHealthySensor(tagSelector)));
                %disp('No centrifugal forces detected' + newline);
        end %end function

            end %end methods

    end %end class
```

9.1.4 RFID Data Processing code (RFID.m)

```
%{
Name : Lau Jie Jian, Yeoh Guan Wei, Darren Chan Yu Hao
National University of Singapore - Ngee Ann Polytechnic

Description : -Interpret and store RFID tag number and time-

Last updated: 20/7/18
%}

classdef RFID
```

```
        properties
          %fileloc
        num
        txt
        raw
        lenTag
        lenTime
        timeHMS
        end

        methods
        function obj = RFID(v)
        if nargin > 0 %check if there are arguments
        %obj.fileloc = v;
        [obj.num,obj.txt,obj.raw]= xlsread(v); %raw is all data
        obj.lenTag = length(obj.raw);
        % For time
        [SigTemp, TStr, Raw] = xlsread(v,'C1:D99999');%C1 to D999999 as a large number to signify end of cell for time collection
        %%Sig is the time
        SigTemp = SigTemp(~isnan(SigTemp));%z remove NaN in Sig (all the empty rows)
        SigTemp = datestr( (SigTemp), 'yyyy-mm-dd HH:MM:SS.FFF' );%convert time from characters to HH:MM:SS
                [obj.lenTime, col] = size(SigTemp); %find length of time to loop through
            obj.timeHMS = SigTemp;
            end %end if
end %end constructor

function tagNum = Tag(obj)
        j = 1; %counter for tagNum within the array below
            for i = 1:1:obj.lenTag
            tf = strcmp(obj.raw(i,1),'Connecting to speedwayr-10-BF-4E');%compare and ignore all these phrases in the csv file
            tf2 = strcmp(obj.raw(i,1),'Press Enter to exit.');
                tf3 = strcmp(obj.raw(i,1),' Reader_name: ');
            if (tf == 0 && tf2 == 0 && tf3 == 0 )%if no 3 phrases then store tag number
            tagNumRFID (j,1) = obj.raw(i,1);%data 2 contain tag number in characters while corresponding to time in Sig2
        j = j+1;
            end %end if
            end %end for loop
            tagNum = tagNumRFID;
end %end function

function time = Time(rfid)
            %take in time which RFID is recorded and turn into seconds

        [Y, M, D, H, MN, S] = datevec(rfid.timeHMS);%extract values of Hours minutes and seconds
            timeSec = H*3600+MN*60+S; %convert hours,minutes and seconds to seconds
            startingtime = timeSec(1,1); %startingtime is actually the time of first tag detected?
            for i = 1:1:rfid.lenTime
            timeSec(i,1) = timeSec(i,1) - startingtime; %make each time value in seconds relative to starting time
            end %end for loop
            time = timeSec;

end %end function

function [tagNumforSensor, exceedData] = StoR(rfid,heightOfSensor,timeOfSensor,timeOfRFID,tagNumRFID)
            %printFlag = 0;
```

```
            exceedData = 0;
            for j = 1:1:heightOfSensor %j is counter for sensor data
            for i = 1:1:rfid.lenTime %Convert time to the RFID location
            breakFlag = 0;
            if ( timeOfSensor(1,j) <= timeOfRFID(i,1) ) %for this to work, RFID and sensor has to start and stop at the same time for collection of
healthy data
            tagNumforSensor(j,1) = tagNumRFID(i,1); %x3 contain tag num at each row of capacitance value for a healthy signal
            breakFlag = 1;
            break;
              end %end if
            end %end for loop - no more RFID tags to assign
            if breakFlag == 0
            tagNumforSensor(j,1) = tagNumRFID(rfid.lenTime,1); %assign the extra sensor data to last tag
            exceedData = 1; %To signify Sensor data exceeding RFID tags
            end
            end %end for loop
      end %end funciton

          end %end methods

end %end class
```

9.1.5 Sensor Data Processing code (Sensor.m)

```
            %{
            Name : Lau Jie Jian, Yeoh Guan Wei, Darren Chan Yu Hao
            National University of Singapore - Ngee Ann Polytechnic

            Description : -convert sensor signal to rfid location-

            Last updated: 7/6/18
            %}

            classdef Sensor

                        properties
                        time
                        height2
                        healthydata
                        end

                        methods
                        function obj = Sensor(sDirectory, fs)
                        obj.healthydata = xlsread(sDirectory);
                        %read healthy signal
                        [obj.height2, w2] = size(obj.healthydata);
                         obj.time = (1:obj.height2)/fs;
                        end %end constructor

                        end %end methods

            end %end class
```

9.1.6 File Locator code (File.m)

```
%{
Name : Lau Jie Jian, Yeoh Guan Wei, Darren Chan Yu Hao
National University of Singapore - Ngee Ann Polytechnic

Description : -Load file from specified locations-

Last updated: 7/6/18
%}

classdef File

        properties
        fileloc
        fileTitle
        direction
        speed
        position
        fileName
        end

        methods
        function obj = File(dir,spd,pos, fileloc)
        if nargin > 0 %check if there are arguments
        obj.direction = dir;
              obj.speed = spd;
        obj.position = pos;
        obj.fileloc = fileloc;
        obj.fileTitle = strcat(dir, '\_', spd, '\_', pos); %strcat required for graph title
        %obj.fileTitle = strcat('NoiseDetection2');
        obj.fileName = [dir, '_', spd, '_', pos];
        end %end if
        end %end constructor

        function filedir = Locate(obj)
            if nargin > 0
            folder = [obj.direction, '_', obj.speed, '_\'];
            file = [obj.fileName, '.csv'];

            filedir = [obj.fileloc, folder, file]; %allocate file directory to this variable
            end %end if
    end %end function

    function Plot(obj, denoising)
            figure;
            ylabel('Capacitance (pF)');
            xlabel('Time (s)');
            if denoising == 0
            title(['Unfiltered data ' obj.fileTitle]);
            else
```

```
            title(['Filtered data ' obj.fileTitle]);
            end %end if-else
        end %end function

    end %end methods

end %end class
```

9.1.7 Turning Tags Convertor code (turnTag.m)

```
%{
Name : Lau Jie Jian, Yeoh Guan Wei, Darren Chan Yu Hao
National University of Singapore - Ngee Ann Polytechnic

Description : -convert sensor signal to rfid location-

Last updated: 7/6/18
%}

classdef Sensor

        properties
        time
        height2
        healthydata
        end

        methods
        function obj = Sensor(sDirectory, fs)
        obj.healthydata = xlsread(sDirectory);
        %read healthy signal
        [obj.height2, w2] = size(obj.healthydata);
         obj.time = (1:obj.height2)/fs;
        end %end constructor

        end %end methods

end %end class
```

9.2 RFID code on Raspberry Pi and Java

9.2.1 Main Code [Edited from OctaneSdk example] (HelloOctaneSdk.java)

```
package com.example.sdksamples;

import com.example.sdksamples.Developer;


import com.impinj.octane.ImpinjReader;
import com.impinj.octane.OctaneSdkException;
import com.impinj.octane.SearchMode;
import com.impinj.octane.Settings;
import com.impinj.octane.Tag;
import com.impinj.octane.TagData;
import com.impinj.octane.TagReport;
import com.impinj.octane.TagReportListener;
import com.impinj.octane.ReportMode;
import com.impinj.octane.ReportConfig;
import com.example.sdksamples.TagReportListenerImplementation;
import java.io.FileWriter;
import java.util.Arrays;
import java.util.ArrayList;
import java.util.List;
import java.util.Scanner;
import javax.swing.JFrame;
import javax.swing.JTextField;

public class HelloOctaneSdk {
 public static String No, Yes;
 public static int Num;
 public static int i = 0;

}


public static void main(String[] args) throws Exception {
 String csvFile = "/Users/User/Desktop/RFID2.csv"; //Set directory for csv file to be created in
 FileWriter writer = new FileWriter(csvFile); //Create a new writer

 try {
  String hostname = System.getProperty(SampleProperties.hostname);

  if (hostname == null) {
   throw new Exception("Must specify the '" +
    SampleProperties.hostname + "' property");
  }

  ImpinjReader reader = new ImpinjReader();

  // Connect
  System.out.println("Connecting to " + hostname);
  reader.connect("192.168.1.253");

  // Get the default settings
  Settings settings = reader.queryDefaultSettings();
```

```
settings.setSearchMode(SearchMode.TagFocus);

// Set session to session 1
settings.setSession(1);

// Apply the new settings
reader.applySettings(settings);

// connect a listener
TagReportListenerImplementation tagDetectedObj = new TagReportListenerImplementation(); //RUNS WHEN TAG
DETECTED
reader.setTagReportListener(tagDetectedObj);

// Start the reader
reader.start();

System.out.println("Press Enter to exit.");


//------------------------------Stored data----------------------------------------------------------------------------
//For every tag detection, data collected will be stored inside a list to be extracted later
//-------------------------------------------------------------------------------------------------------------------
while (i < 10000000) { //Looping for 10000000 times to test if the body code runs
 //if (IsKeyPressed.isWPressed()) { //Trying to detect a key pressed but doesn't work [Need to be fixed]

 Num = tagDetectedObj.count; //Data can be passed

 System.out.println(Yes + No + Num + " " + i); //Somehow the print code must be present for the code to work[Need to be fixed]

 //Num will be set to 1 when a tag is detected as 'count' in the 'TagReportListenerImplementation' class will be incremented
 if (Num == 1) { //If Num equals to 1, run the code, else ignore the whole 'if' block
  Yes = tagDetectedObj.tagEpcObj; //Pass EPC value from TagReportListenerImplementation class to variable 'Yes'
  System.out.println("HI"); //To test if the code runs, but it is too fast to be printed for now[Need to be fixed]
  No = tagDetectedObj.Date; //Pass Date value from TagReportListenerImplementation class to variable 'No'

  List < Developer > developers = Arrays.asList( //Put 'Yes' and 'No' into Developer for extraction
   new Developer(No, Yes));

  for (Developer d: developers) {

   List < String > list = new ArrayList < > (); //Create a new list
   list.add(d.getEPC()); //Adding the EPC value into the list
   list.add(d.getTime()); //Adding the Date/Time value into the list

   ExtractData.writeLine(writer, list);

  } //End for

  Num = 0; //Reset Num to 0
  tagDetectedObj.count = 0; //Reset count to 0

 } //End if

 i++; //Increment i
 //}
} //End while
//-------------------------------------------------------------------------------------------------------------------
Scanner s = new Scanner(System.in); //WAITS FOR 'ENTER' HERE
s.nextLine();
```

```
  System.out.println("Stopping  " + hostname);
  reader.stop();

  System.out.println("Disconnecting from " + hostname);
  reader.disconnect();

  System.out.println("Done");
 } catch (OctaneSdkException ex) {
  System.out.println(ex.getMessage());
 } catch (Exception ex) {
  System.out.println(ex.getMessage());
  ex.printStackTrace(System.out);
 }

 writer.flush();
 writer.close();
}
}
```

9.2.2 Tag Report Listener Implementation [Edited from OctaneSdk example] (TagReportListenerImplementation.java)

```
package com.example.sdksamples;

import com.impinj.octane.ImpinjReader;
import com.impinj.octane.Tag;
import com.impinj.octane.TagReport;
import com.impinj.octane.TagReportListener;

import java.time.LocalDateTime;
import java.time.format.DateTimeFormatter;
import java.util.List;
import java.util.ArrayList;
import java.util.Arrays;
import java.io.FileWriter;
import java.io.IOException;


public class TagReportListenerImplementation implements TagReportListener{
            public String tagEpc; //Used to store the tag EPC data here to pass to the main later
            public String Date; //Used to store the Date/Time data here to pass to the main later
            public int count; //A counter to signal that a tag is detected as it will increase every time this whole class is called
(Meaning tag detected)



   @Override
   public void onTagReported(ImpinjReader reader, TagReport report) {
      List<Tag> tags = report.getTags();

      for (Tag t : tags) {
         System.out.print(" EPC: " + t.getEpc().toString()); //Display EPC value of the tags
            tagEpc =t.getEpc().toString(); // Store Data into 'tagEpc'

         LocalDateTime now = LocalDateTime.now(); //Get Date & Time

                              DateTimeFormatter formatter = DateTimeFormatter.ofPattern("yyyy-MM-dd HH:mm:ss"); //Set Date &
```

Time format

```
                        String formatDateTime = now.format(formatter); //Format

                        System.out.println(" Date : " + formatDateTime); //Display Date & Time
                        Date= formatDateTime; //Store Data into 'Date'
                        count++; //Increase count to change 'Num' in Main

        if (reader.getName() != null) {
            System.out.print(" Reader_name: " + reader.getName());
        } else {
            System.out.print(" Reader_ip: " + reader.getAddress());
        }

        System.out.println("");


    } //end for loop
  } //end TagReported method
} //end class
```

9.2.3 Extract Data (ExtractData.java)

```
package com.example.sdksamples;

import java.io.Writer;
import java.io.IOException;
import java.util.List;

public class ExtractData {

        private static final char DEFAULT_SEPARATOR = ',';

        public static void writeLine(Writer w, List<String> values) throws IOException {
    writeLine(w, values, DEFAULT_SEPARATOR, ' ');
  }

  public static void writeLine(Writer w, List<String> values, char separators) throws IOException {
    writeLine(w, values, separators, ' ');
  }

  //https://tools.ietf.org/html/rfc4180
  private static String followCVSformat(String value) {

    String result = value;
    if (result.contains("\"")) {
      result = result.replace("\"", "\"\"");
    }
    return result;

  }

  public static void writeLine(Writer w, List<String> values, char separators, char customQuote) throws IOException {

    boolean first = true;

    //default customQuote is empty
```

```
        if (separators == ' ') {
            separators = DEFAULT_SEPARATOR;
        }

        StringBuilder sb = new StringBuilder();
        for (String value : values) {
            if (!first) {
                sb.append(separators);
            }
            if (customQuote == ' ') {
                sb.append(followCVSformat(value));
            } else {
                sb.append(customQuote).append(followCVSformat(value)).append(customQuote);
            }

            first = false;
        }
        sb.append('\n');
        w.append(sb.toString());


    }

}
```

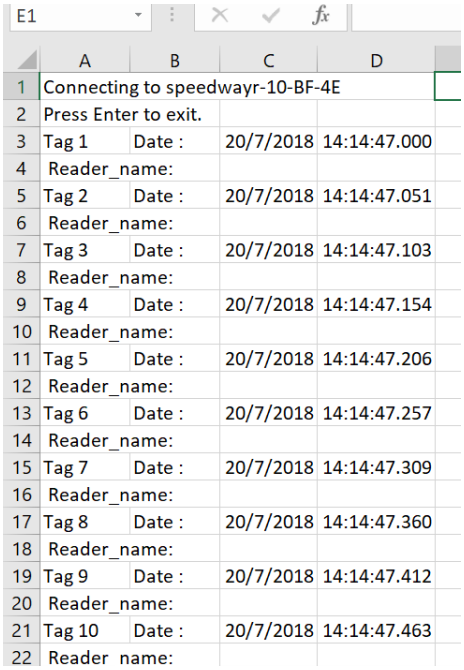
9.2.3 Get Date (GetDate.java)

```
package com.example.sdksamples;

import java.time.LocalDateTime;
import java.time.format.DateTimeFormatter;

public class GetDate {


		public void getDate(String[] args) {
			LocalDateTime now = LocalDateTime.now();

			DateTimeFormatter formatter = DateTimeFormatter.ofPattern("yyyy-MM-dd HH:mm:ss");

			String formatDateTime = now.format(formatter);

			System.out.println("Date : " + formatDateTime);
		}


}
```

9.3 Data used

9.3.1 RFID Data

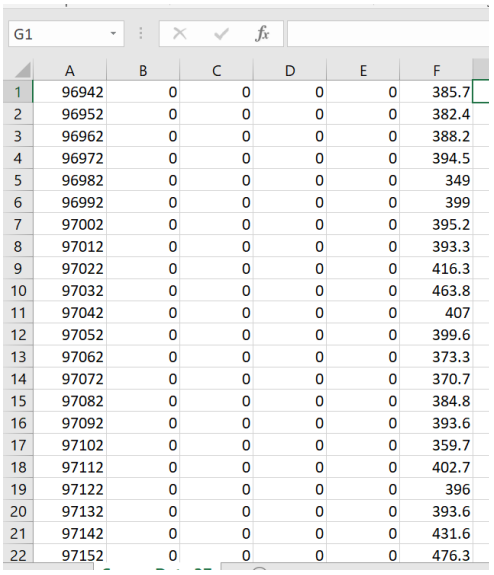
Format of RFID data, tag name is manually changed for ease of reading.



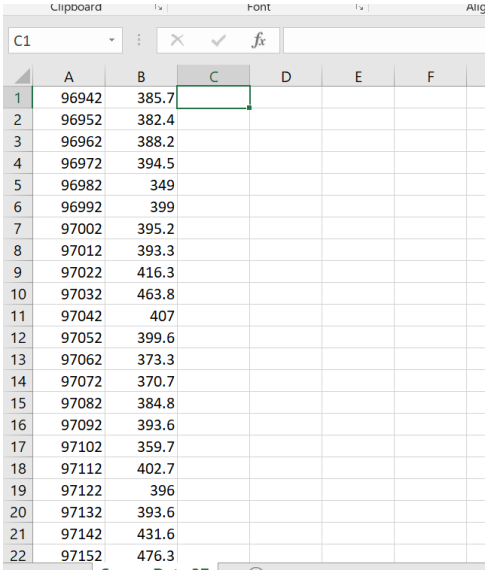
	A	B	C	D
1	Connecting to speedwayr-10-BF-4E			
2	Press Enter to exit.			
3	Tag 1	Date :	20/7/2018	14:14:47.000
4	Reader_name:			
5	Tag 2	Date :	20/7/2018	14:14:47.051
6	Reader_name:			
7	Tag 3	Date :	20/7/2018	14:14:47.103
8	Reader_name:			
9	Tag 4	Date :	20/7/2018	14:14:47.154
10	Reader_name:			
11	Tag 5	Date :	20/7/2018	14:14:47.206
12	Reader_name:			
13	Tag 6	Date :	20/7/2018	14:14:47.257
14	Reader_name:			
15	Tag 7	Date :	20/7/2018	14:14:47.309
16	Reader_name:			
17	Tag 8	Date :	20/7/2018	14:14:47.360
18	Reader_name:			
19	Tag 9	Date :	20/7/2018	14:14:47.412
20	Reader_name:			
21	Tag 10	Date :	20/7/2018	14:14:47.463
22	Reader_name:			

9.3.2 Sensor Data

Format of Stretchable sensor data, columns with values of 0 (unused channels) were removed for ease of reading and arrangement.



	A	B	C	D	E	F
1	96942	0	0	0	0	385.7
2	96952	0	0	0	0	382.4
3	96962	0	0	0	0	388.2
4	96972	0	0	0	0	394.5
5	96982	0	0	0	0	349
6	96992	0	0	0	0	399
7	97002	0	0	0	0	395.2
8	97012	0	0	0	0	393.3
9	97022	0	0	0	0	416.3
10	97032	0	0	0	0	463.8
11	97042	0	0	0	0	407
12	97052	0	0	0	0	399.6
13	97062	0	0	0	0	373.3
14	97072	0	0	0	0	370.7
15	97082	0	0	0	0	384.8
16	97092	0	0	0	0	393.6
17	97102	0	0	0	0	359.7
18	97112	0	0	0	0	402.7
19	97122	0	0	0	0	396
20	97132	0	0	0	0	393.6
21	97142	0	0	0	0	431.6
22	97152	0	0	0	0	476.3



	A	B	C	D	E	F
1	96942	385.7				
2	96952	382.4				
3	96962	388.2				
4	96972	394.5				
5	96982	349				
6	96992	399				
7	97002	395.2				
8	97012	393.3				
9	97022	416.3				
10	97032	463.8				
11	97042	407				
12	97052	399.6				
13	97062	373.3				
14	97072	370.7				
15	97082	384.8				
16	97092	393.6				
17	97102	359.7				
18	97112	402.7				
19	97122	396				
20	97132	393.6				
21	97142	431.6				
22	97152	476.3				

Raw Processed