PROJECT REPORT

EE 4136

Knocking detection of IC engine using digital signal processing

Submitted by:

1.Thasin Mohammad Zaman(**1903069**)

2.Mridul Mondal(**1903070**)

Project Name: Knocking detection of IC engine using digital signal processing

➢ Motivation For This Work:

Engine knocking, also known as detonation or pinging, is a critical phenomenon in internal combustion (IC) engines that can significantly impact engine performance, efficiency, and longevity. Understanding and detecting engine knocking is essential for maintaining engine health and optimizing its operation.

What is Engine Knocking?

Engine knocking occurs when the air-fuel mixture in the combustion chamber of an IC engine does not burn smoothly and evenly. Instead of the controlled, progressive burn initiated by the spark plug, the mixture ignites prematurely or unevenly in multiple locations. This uncontrolled combustion creates high-pressure shock waves that collide within the cylinder, producing a characteristic "knock" or "ping" sound.

Causes of Engine Knocking

Several factors can contribute to engine knocking:

- ❖ **High Compression Ratios:** Engines with high compression ratios are more prone to knocking due to the increased likelihood of the air-fuel mixture reaching auto-ignition temperatures before the spark plug fires.
- ❖ **Low-Octane Fuel:** Using fuel with an octane rating lower than what the engine is designed for can lead to premature ignition and knocking. Higher octane fuels resist knocking better by withstanding higher compression without auto-ignition.
- ❖ **Engine Temperature:** Elevated engine temperatures can increase the likelihood of knocking as higher temperatures promote premature ignition of the air-fuel mixture.
- ❖ **Lean Air-Fuel Mixture:** A mixture with too much air and not enough fuel (lean mixture) can burn too quickly and cause knocking.
- ❖ **Advanced Ignition Timing**: If the spark plug fires too early, the pressure from the initial combustion can cause the remaining air-fuel mixture to ignite prematurely, leading to knocking.

<u>Consequences of Engine Knocking:</u>

The consequences of unchecked engine knocking can be severe:

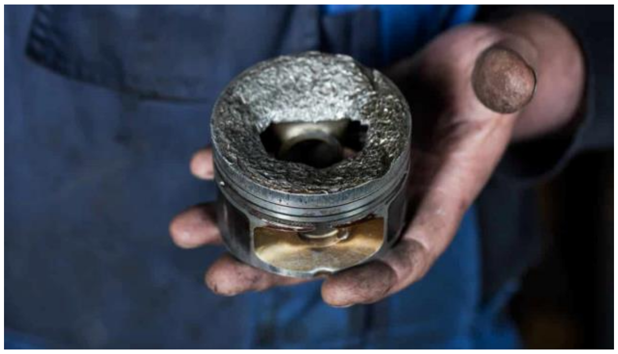
Engine Damage: Persistent knocking can lead to physical damage to engine components, such as pistons, cylinder walls, and bearings. The repeated high-pressure impacts can cause cracks, pits, and other forms of mechanical wear.

Reduced Efficiency: Knocking disrupts the smooth combustion process, leading to a loss of engine efficiency. This results in lower fuel economy and reduced power output.

Increased Emissions: Inefficient combustion caused by knocking can lead to higher emissions of pollutants, as the air-fuel mixture does not burn completely.

Engine Overheating: The additional heat generated by knocking can exacerbate engine overheating issues, further stressing the engine and its cooling system.

Some pictorial representations of consequences



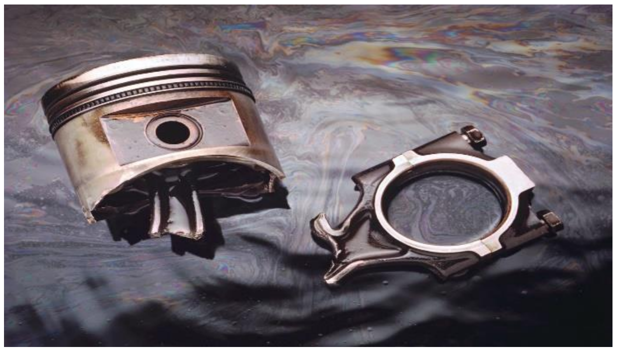


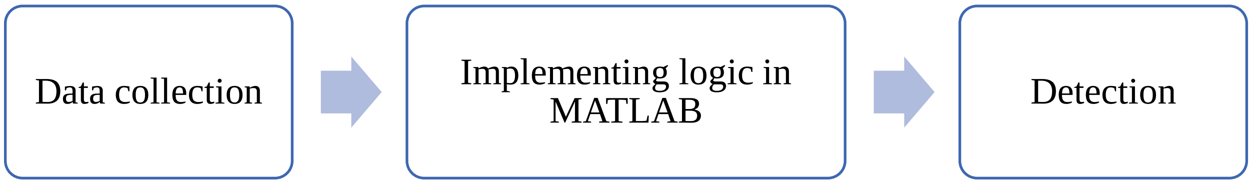
Fig1.1: Severe damage of engine parts

➢ <u>Detection techniques:</u>

<u>Role of Digital Signal Processing in Knocking Detection:</u>

Digital Signal Processing (DSP) plays a pivotal role in modern knocking detection systems. By converting analog signals from knock sensors into digital data, DSP allows for sophisticated analysis and real-time detection of knocking events. Techniques such as Fourier Transform, Wavelet Transform, and statistical analysis enable precise identification of knocking patterns, leading to timely and accurate interventions. In this project, Fourier transform has been used for detection.

➢ **Methodology:**



Implemented logic(MATLAB programming):

```
 [engineSound, fs] = audioread('engine_audio_bad_1.wav');
fprintf('Sampling Frequency: %d Hz\n', fs);

soundsc(engineSound,fs);

engineSound = engineSound / max(abs(engineSound));

figure;
subplot(3,1,1);
plot(engineSound);
title('Time-Domain Signal');
xlabel('Sample');
ylabel('Amplitude');

filterOrder = 8;
cutoffFrequency = 500;
[b, a] = butter(filterOrder, cutoffFrequency / (fs / 2), 'high');


engineSoundFiltered = filtfilt(b, a, engineSound);

subplot(3,1,2);
plot(engineSoundFiltered);
title('Filtered Time-Domain Signal');
xlabel('Sample');
ylabel('Amplitude');


engineSoundFFT = fft(engineSoundFiltered);
N = length(engineSoundFFT);

frequencies = (0:(N/2)-1)*(fs/N);
engineSoundFFT = engineSoundFFT(1:N/2);

knockingFreqRange = [5000 7000];

knockingIndices = find(frequencies >= knockingFreqRange(1) &
frequencies <= knockingFreqRange(2));
```

```
knockingPower = sum(abs(engineSoundFFT(knockingIndices)).^2);

totalPower = sum(abs(engineSoundFFT).^2);


normalizedKnockingPower = knockingPower / totalPower;

subplot(3,1,3);
plot(frequencies, abs(engineSoundFFT));
title('Frequency-Domain Signal');
xlabel('Frequency (Hz)');
ylabel('Magnitude');
hold on;

plot(frequencies(knockingIndices),
abs(engineSoundFFT(knockingIndices)), 'r.', 'MarkerSize', 10);

legend('Overall Spectrum', 'Knocking Frequencies');

hold off;

normalizedKnockingThreshold = 0.1
if normalizedKnockingPower > normalizedKnockingThreshold
    fprintf('Bad engine: Knocking detected.\n');
else
    fprintf('Good engine: No significant knocking detected.\n');
end

fprintf('Normalized Knocking Power: %f\n', normalizedKnockingPower);
fprintf('Normalized Knocking Threshold: %f\n',
normalizedKnockingThreshold);
```

➢ <u>Explanation of implemented logic</u>

- ***% Loading engine sound data***

[engineSound, fs] = audioread('engine_audio_bad_2.wav');

<u>Description:</u>

audioread: Reads the audio file 'engine_audio_bad_2.wav' and stores the audio data in engineSound and the sampling frequency in **fs**.

- ***% Normalizing the audio signal***

engineSound = engineSound / max(abs(engineSound));

<u>Description:</u> Normalizes the audio signal so that its amplitude ranges between -1 and 1. This helps in avoiding any issues related to signal scaling.

- ***% Ploting the time-domain signal***

```
figure;
subplot(3,1,1);
plot(engineSound);
title('Time-Domain Signal');
xlabel('Sample');
ylabel('Amplitude');
```

Description: Creates a new figure window and plots the normalized time-domain signal in the first subplot.

- ***% Designing a Butterworth high-pass filter to remove low-frequency noise***

```
filterOrder = 8;
cutoffFrequency = 500; % Cutoff frequency in Hz
[b, a] = butter(filterOrder, cutoffFrequency / (fs / 2), 'high');
```

Description: Designs a Butterworth high-pass filter with an order of 8 and a cutoff frequency of 500 Hz. butter returns the filter coefficients b and a.

- ***% Applying the Butterworth high-pass filter***

```
engineSoundFiltered = filtfilt(b, a, engineSound);
```

Description: Filters the normalized engine sound signal using the designed Butterworth high-pass filter. filtfilt applies the filter in both forward and reverse directions to minimize phase distortion.

- ***% Ploting the filtered time-domain signal***

```
subplot(3,1,2);
plot(engineSoundFiltered);
title('Filtered Time-Domain Signal');
xlabel('Sample');
ylabel('Amplitude');
```

Description: Plots the filtered time-domain signal in the second subplot.

- ***% Computing the Fourier Transform of the filtered signal***

```
engineSoundFFT = fft(engineSoundFiltered);
N = length(engineSoundFFT);
```

Description: Computes the Fourier Transform of the filtered signal using the fft function. N is the length of the filtered signal.

- ***% Only using the first half of the FFT results (positive frequencies)***

frequencies = (0:(N/2)-1)(fs/N);*

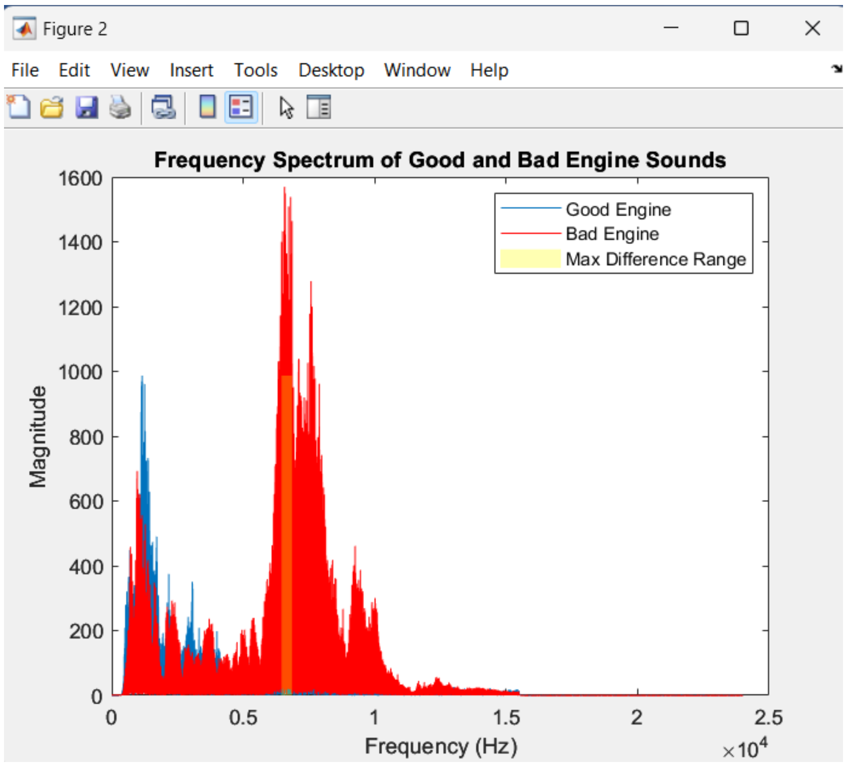
engineSoundFFT = engineSoundFFT(1:N/2);

Description: Only the first half of the FFT results are used since they represent the positive frequencies. frequencies array holds the corresponding frequency values.

- ***% Defining the frequency range for knocking (example range: 5000-7000 Hz)***

knockingFreqRange = [5000 7000];

Defines the frequency range in which knocking is expected to occur.



- **% Finding indices corresponding to the knocking frequency range**

knockingIndices = find(frequencies >= knockingFreqRange(1) & frequencies <= knockingFreqRange(2));

Finds the indices of the frequencies that fall within the defined knocking frequency range.

- ***% Calculating the power in the knocking frequency range*** *knockingPower = sum(abs(engineSoundFFT(knockingIndices)).^2);*

Calculates the power of the signal within the knocking frequency range by summing the squared magnitudes of the FFT results at the knocking indices.

- ***% Calculating the total power of the signal***

totalPower = sum(abs(engineSoundFFT).^2);

Calculates the total power of the signal by summing the squared magnitudes of all FFT results.

- ***% Normalizing the knocking power with respect to the total power***

normalizedKnockingPower = knockingPower / totalPower;

Normalizes the knocking power by dividing it by the total power of the signal.

- ***% Plot the frequency-domain signal***

```
subplot(3,1,3);

plot(frequencies, abs(engineSoundFFT));

title('Frequency-Domain Signal');

xlabel('Frequency (Hz)');

ylabel('Magnitude');

hold on;
```

Plots the frequency-domain signal (magnitude of FFT results) in the third subplot and holds the plot for further modifications.

- ***% Marking the frequencies in the knocking range with red points***

```
plot(frequencies(knockingIndices), abs(engineSoundFFT(knockingIndices)), 'r.', 'MarkerSize', 10);
```

- ***% Threshold for classifying as 'bad' engine (normalized power)***

normalizedKnockingThreshold = 0.1; % Adjust based on empirical data

Defines a threshold value for classifying the engine as 'bad' based on normalized knocking power. This value can be adjusted based on empirical data.

- ***% Classify the engine sound***

```
if (normalizedKnockingPower > normalizedKnockingThreshold)

        fprintf('Bad engine: Knocking detected.\n');

else

         fprintf('Good engine: No significant knocking detected.\n');
```

end

Description: Classifies the engine sound as 'bad' or 'good' based on whether the normalized knocking power exceeds the threshold and prints the result.

- ***% Print the results***

fprintf('NormalizedKnocking Power: %f\n', normalizedKnockingPower);

fprintf('Normalized_Knocking_Threshold:%f \n',normalizedKnockingTr-eshold);

➢ OUTPUTS:

For testing the ability of detection, we used four audio files. Two of them were knocking integrated sound and two of them were good sound. And the program perfectly detects the knocking by maintaining the process described above.

➢ Output waveshapes and comments

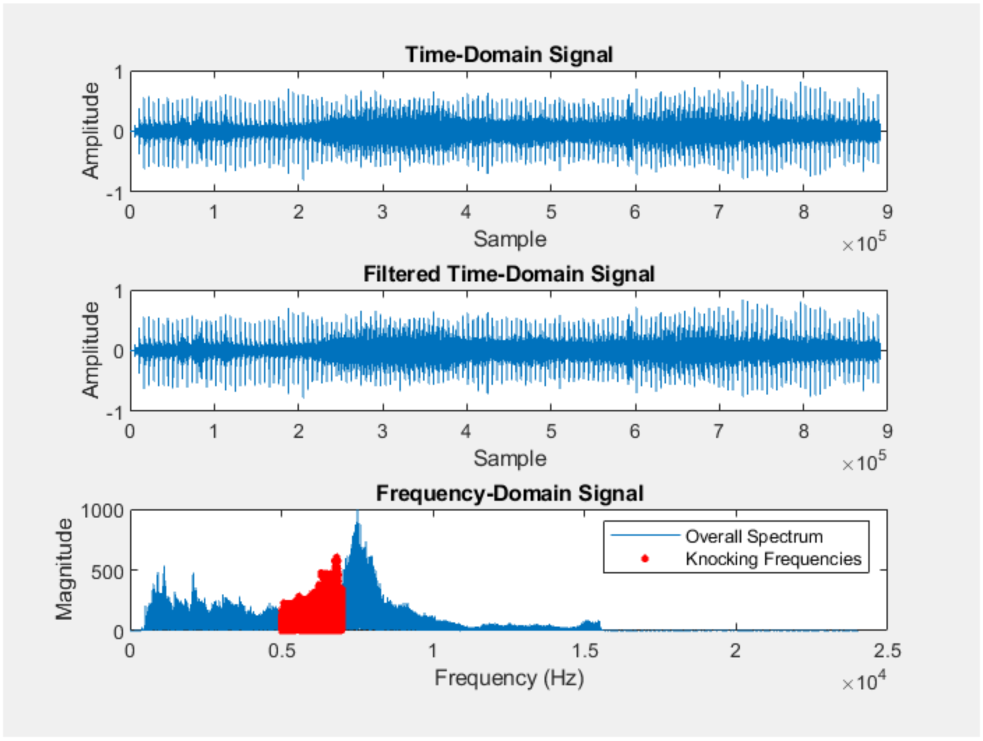
From command window:

- **After applying a out of order engine sound:**

```
Bad engine: Knocking detected.

Normalized Knocking Power: 0.199870

Normalized Knocking Threshold: 0.100000
```



- **After applying sound of a good engine:**

Good engine: No significant knocking detected.

Normalized Knocking Power: 0.098740

Normalized Knocking Threshold: 0.100000

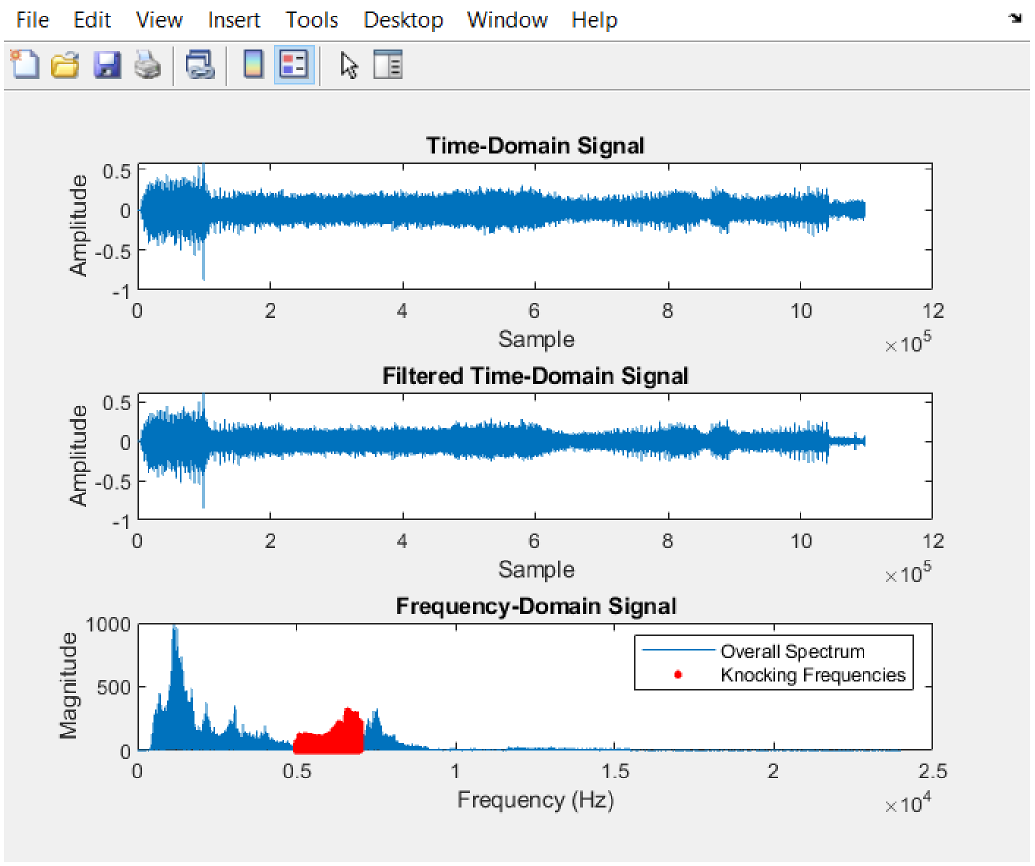
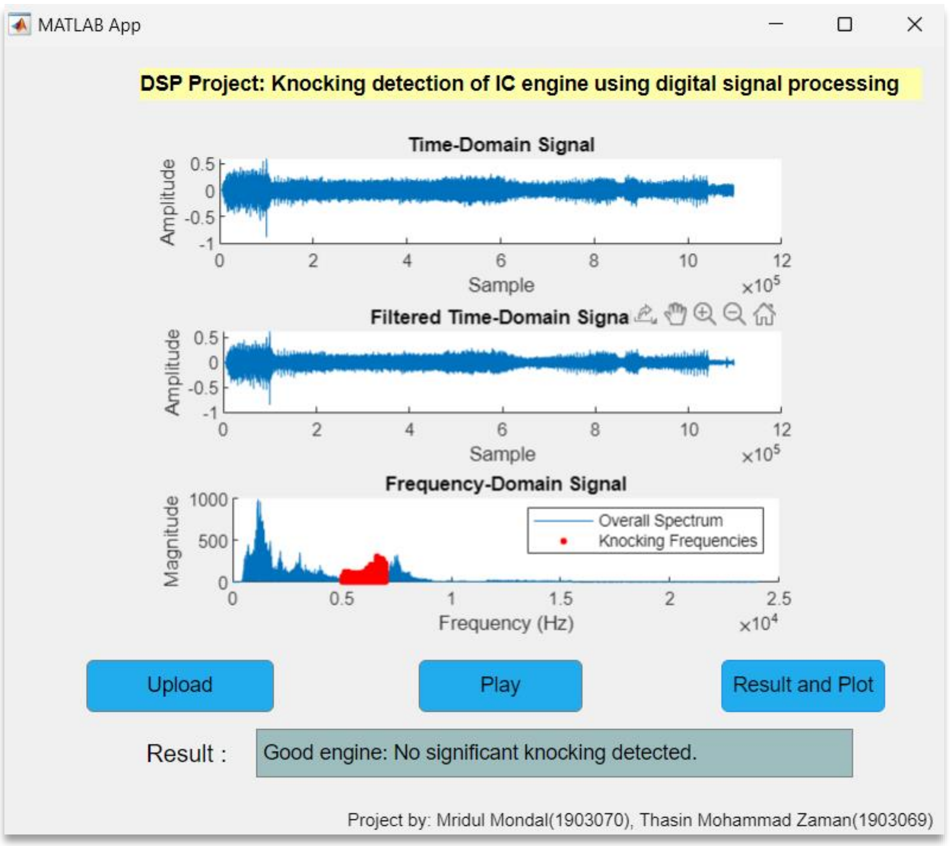


Figure: Waveshapes for good Engine

Here we developed a **GUI** (Graphical User Interface) in MATLAB for representation:



DSP Project: Knocking detection of IC engine using digital signal processing

Result : Good engine: No significant knocking detected.

Project by: Mridul Mondal(1903070), Thasin Mohammad Zaman(1903069)

Figure: Graphical Interface of the project

➢ Conclusion:

The primary objective of this project was to apply the concepts and techniques learned in five consecutive Digital Signal Processing Labs. We successfully implemented a Butterworth filter and a simple FFT to detect knocking phenomena in the sound of an IC engine. This approach effectively identified engine knocking, demonstrating the practical application of DSP methods.

➢ Used platform: MATLAB 2022b

➢ References:

[1] https://www.mathworks.com

[2] https://soundcloud.com