MARMARA UNIVERSITY

BACHELOR'S GRADUATION REPORT

Expert System Based AI Software on Vibration Based Machinery Fault Diagnoses

---

Author:
Berkay ATAÇOĞLU

Supervisor:
Assoc. Prof. İ. Sina KUSEYRI

A thesis submitted in fulfillment of the requirements for the Bachelor's Degree of Mechanical Engineering

in the

Marmara University

June 23, 2025

MARMARA UNIVERSITY

Abstract

Engineering Faculty

Mechanical Engineering Department

Bachelor's Degree of Mechanical Engineering

Expert System Based AI Software on Vibration Based Machinery Fault Diagnoses

by Berkay ATAÇOĞLU

This thesis introduces a web based expert system software for vibration-based machinery fault diagnosis in Turkish. Currently the application only diagnoses centrifugal pumps with electrical motors. The expert knowledge extracted from my supervisor and various ISO standards, were put in a vector form that relates various diagnoses to various symptoms. There were total of 22 possible diagnoses, and 24 possible symptoms considered. As for the inference, the application relied upon the concept of cosine similarity. Every symptom vector was scored based on their relation to the answers user provided. The diagnoses with the highest similarity scores were then listed as the actual diagnosis of the application.

Acknowledgements

Firstly, I would like to thank my supervisor Assoc. Prof. İ. Sina Kuseyri for taking his time and effort to guide me through this project. Without his knowledge and guidance this project would not be possible.

I would also like to thank my other teachers from the university. Even though I was not the most responsible and active student, their teachings and understanding have led me to where I am as a soon to be mechanical engineer.

Finally, I would like to thank my friends and family for supporting me through hardships.

Thank you all ...

Contents

List of Figures

List of Tables

List of Abbreviations

UI	**U**ser **I**nterface
HTML	**H**yper**T**ext **M**arkup **L**anguage
CSS	**C**ascading **S**tyle **S**heets
JS	**J**ava**S**cript
PC	**P**ersonal **C**omputer
AI	**A**rtificial **I**ntelligence
CBS	**C**ase **B**ased **R**easoning
e.g.	**F**or **E**xample
etc.	**Et C**etera

List of Symbols

ω	angular frequency	rad/s
x	displacement	m
t	time	s
C_1, C_2	integration constants	unitless
ζ	damping ratio	unitless
c	damping coefficient	$\frac{Kg}{s}$
m	mass	Kg
k	stiffness coefficient	$\frac{Kg}{s^2}$
ω_n	natural frequency	$\frac{rad}{s}$
F_0	amplitude of harmonic force	$\frac{Kg \cdot m}{s^2}$
X	amplitude of oscillation	m
ϕ	phase angle	rad
θ	angle between vectors	rad
S_c	cosine similarity	unitless

1. INTRODUCTION

Fault diagnosis refers to the process of detecting, isolating, and identifying faults in a system to ensure reliability, safety, and optimal performance. In both mechanical and electrical systems, faults can lead to reduced efficiency, unexpected downtime and damage to equipment. These will then cost money or even cause harm to people. Because of this, fault diagnosis has become a critical aspect of system monitoring and maintenance across a wide range of industries, including manufacturing, aerospace, automotive, and power generation.

Mechanical systems, such as engines, pumps, and gearboxes, are prone to faults like misalignment, imbalance, wear, or bearing failures. These faults often develop gradually and manifest through physical indicators such as vibration, noise, or temperature changes. Electrical systems, including motors, transformers, and power electronics, may suffer from faults such as short circuits, open circuits, insulation degradation, or overheating. These can result in anomalies in current, voltage, and power signals.

Traditionally, fault diagnosis relied heavily on manual inspection and scheduled maintenance, which were often inefficient and costly. With the advancement of sensors, data acquisition systems, and signal processing techniques, condition-based and predictive maintenance strategies have emerged. These methods utilize real-time monitoring and diagnostic algorithms to detect early signs of failure, allowing for timely interventions. Though now, we have access to advanced methods of fault diagnosis, they require extra capital, knowledge and effort to set up. Many people still use the traditional fault diagnosis methods.

Our method for fault diagnosis will not be as advanced or as costly to set up as modern advanced techniques, or as inefficient and costly to maintain as the traditional approach. We have created an algorithm that can utilize already established empirical data for various machinery, and knowledge and experience gathered through the years by an expert. Though it relies on accurate answers to our expert questions for accurate fault diagnosis, one big upside is that it can be scaled to various types of machinery easily. As of now, for this report, we've only coded into our algorithm, the expert questions and diagnosis for a centrifugal pump.

2. LITERATURE REVIEW

2.1 Mechanical Vibrations

Vibration is the repetitive, periodic, or oscillatory response of a mechanical system. Typically, all rotating machines produce vibrations due to dynamic forces, which mostly come from imbalanced, misaligned, worn, or improperly driven rotating parts.

Mechanical vibrations can only exist if the order of the system model is at least of the second order. Second order systems include stiffness and damping elements. Stiffness elements are often modeled as springs, while damping elements are often modeled as dashpots since they're linear and easy to do calculations on. Even if the order of the system is higher than two, a second order model will still yield satisfactory results.

2.1.1 Modeling second order systems

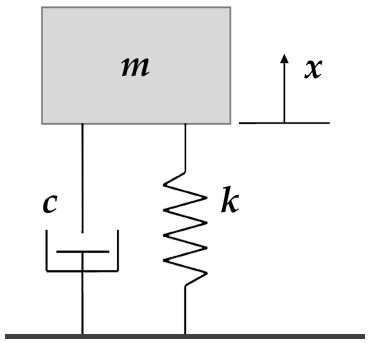

FIGURE 2.1: Simple Mass Spring Damper Model

Figure 2.1 shows the simple free body diagram of a second order system. Note that $x = 0$ corresponds to the system's equilibrium state, meaning the effect of gravity has already compressed the spring a small amount, so won't be included again. These kinds of systems are more often modeled using Newton's 2nd Law. Assuming no external exciting force acting on the system;

$$m\ddot{x} = -c\dot{x} - kx \tag{2.1}$$

And in standard form,

$$\ddot{x} + \frac{c}{m}\dot{x} + \frac{k}{m}x = 0 \tag{2.2}$$

This is a second order, linear, ordinary differential equation in standard form. There are two very important characteristic parameters that can be extracted from this. First of which, is called the natural frequency ω_n.

$$\omega_n = \sqrt{\frac{k}{m}} \quad (rad/s) \tag{2.3}$$

Natural frequency is defined as the frequency at which a system naturally oscillates when it is disturbed from its resting position and is allowed to vibrate freely. This is one of the two most important characteristics of a second order system, its implications for the system will be discussed briefly.

The other highly important characteristic of a second order system is called damping ratio ζ. It is unitless. It describes how much damping a particular system has relative to the amount needed to prevent oscillations.

$$\zeta = \frac{c}{2m\omega_n} \tag{2.4}$$

Combining Equation 2.2 with the characteristic parameters found in equations 2.3 and 2.4 yields the very useful formula;

$$\ddot{x} + 2\zeta\omega_n\dot{x} + \omega_n^2 x = 0 \tag{2.5}$$

Note that our initial model has no inputs. That's why the right-hand side of Equation 2.5 is zero. We know from our intuition that the system will oscillate. Because there is oscillation, we will assume a solution in the form of;

$$x(t) = C_1 e^{s_1 t} + C_2 e^{s_2 t} \tag{2.6}$$

This is called the homogenous solution, where C_1 and C_2 are some constants calculated from initial conditions and s_1 and s_2 are the solutions to the characteristic equation;

$$s_1, s_2 = -\zeta\omega_n \pm j\omega_n\sqrt{1-\zeta^2} \tag{2.7}$$

In Equation 2.7, complex term j only applies for underdamped systems.

2.1.2 Types of systems

We often categorize second order systems by their damping ratio.

TABLE 2.1: Types of Systems Based on Their Damping Ratio

Damping Ratio	Response Type	Description
$\zeta = 0$	Undamped	No energy loss, purely oscillatory
$0 < \zeta < 1$	Underdamped	Oscillatory with decaying amplitude
$\zeta = 1$	Critically damped	Fastest response without overshoot. No oscillation;
$\zeta > 1$	Overdamped	No oscillation. Slow return to equilibrium due to excess damping

No system can be truly undamped, since it is impossible to prevent energy loss in any form of motion. And we don't often want overdamped or critically damped responses on our dynamic machinery, since it will mean that the component will quickly get drained of its energy and will be unable to move. So, most of the systems that we use are actually underdamped systems. Modifying Equation 2.7 for underdamped systems;

$$s_1, s_2 = -\zeta\omega_n \pm j\omega_d \tag{2.8}$$

Where

$$\omega_d = \omega_n \sqrt{1 - \zeta^2} \tag{2.9}$$

This is called the damped frequency. This is the actual frequency that a system oscillates in. One might think that keeping the damping ratio as low as possible would be beneficial, since less energy will be drained from the system. But this might also be catastrophic for the system because it will affect the system's stability.

2.1.3 Forced Vibrations

When a dynamic system is subjected to an external, time-varying excitation, Equation 2.2 is then modified to include the input to the system.

$$\ddot{x} + \frac{c}{m}\dot{x} + \frac{k}{m}x = \frac{F_0 \sin \omega t}{m} \tag{2.10}$$

Where F_0 is the amplitude for the harmonic force, m is the mass, c is the damping coefficient, k is the stiffness coefficient and ω is the forcing frequency.

Solution for Equation 2.10 consists of two parts. Homogeneous solution, and particular solution.

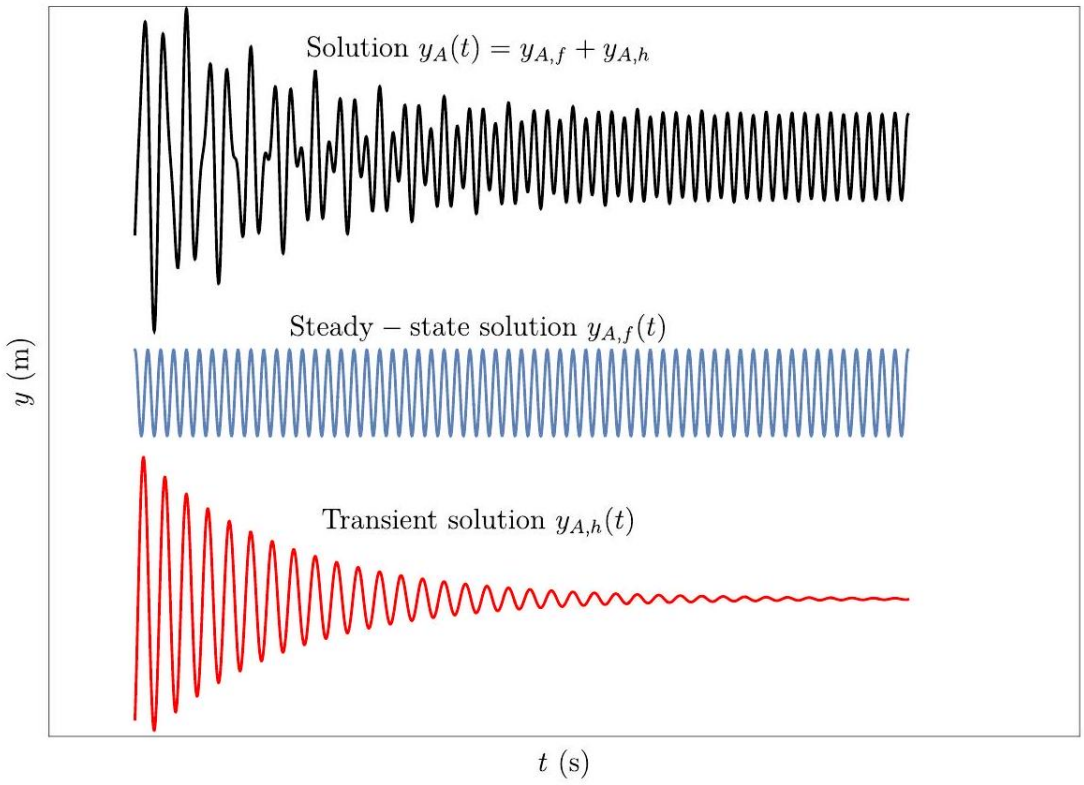

FIGURE 2.2: Transient and Steady-state Solutions of Forced Vibrations

Homogeneous solution is the same as Equation 2.6. For the particular solution, we know that system with oscillate with the excitation frequency. We assume particular solution in the form of;

$$x = X \sin(\omega t - \phi) \tag{2.11}$$

Where X is the amplitude of oscillation and ϕ is the phase of the displacement with respect to the exciting force.

The amplitude and phase in Equation 2.11 can be found by substituting it into Equation 2.10. Though we need to remember that in harmonic motion the phases of velocity and acceleration are ahead of the displacement by 90 deg and 180 deg, respectively.

$$X = \frac{F_0}{\sqrt{(k - m\omega^2)^2 + (c\omega)^2}} \tag{2.12}$$

And

$$\phi = \tan^{-1} \frac{c\omega}{k - m\omega^2} \tag{2.13}$$

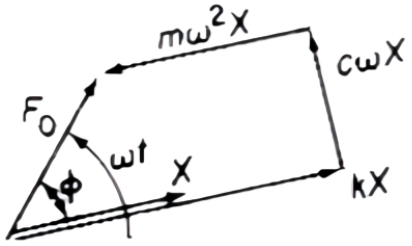

FIGURE 2.3: Phase Differences Between Displacement, Velocity and Acceleration

Plugging Equations 2.3 and 2.4 into Equations 2.12 and 2.13, and modifying it a little into a dimensionless form;

$$\frac{Xk}{F_0} = \frac{1}{\sqrt{[1 - (\frac{\omega}{\omega_n})^2]^2 + [2\zeta(\frac{\omega}{\omega_n})]^2}} \tag{2.14}$$

And

$$\tan\phi = \frac{2\zeta(\frac{\omega}{\omega_n})}{1 - (\frac{\omega}{\omega_n})^2} \tag{2.15}$$

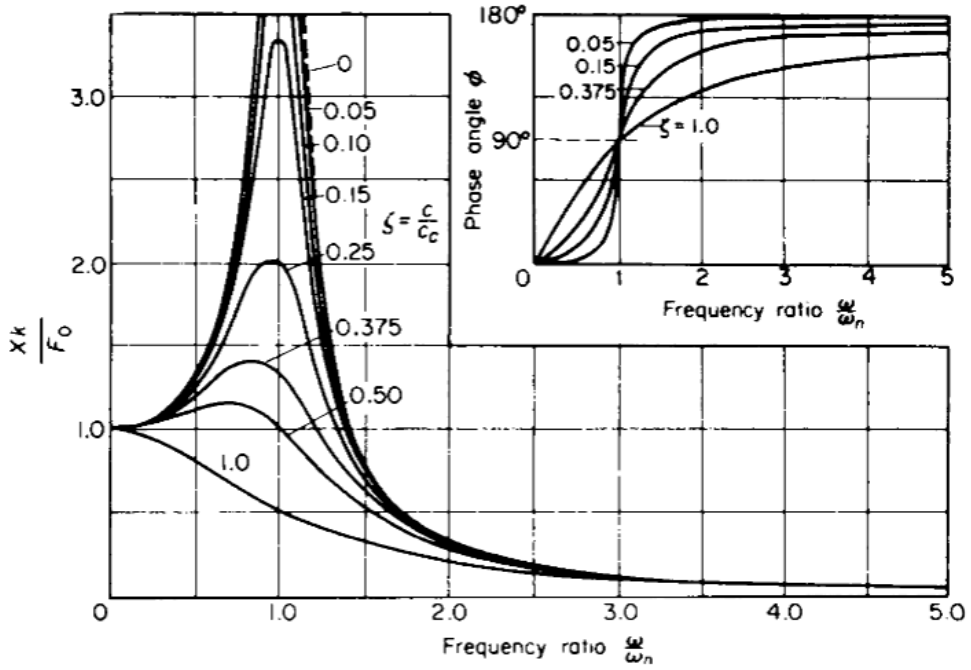

FIGURE 2.4: Plot of Equations 2.14 and 2.15

2.1.4 Resonance

Another crucial concept regarding vibrations is the effect of resonance. When a system is excited externally by a frequency close to its natural frequency, the system will oscillate with maximum amplitude. This phenomenon occurs in any system that can store and transfer energy between two forms. As you can see from Figure 2.4, as the excitation frequency closes around the natural frequency, that is $\frac{\omega}{\omega_n} \simeq 1$, the magnitude of the vibrations becomes maximum. This is catastrophic for the system and is never desirable. You can also see from the same plot, the effect of damping on system's maximum amplitude.

2.1.5 Stability

Stability refers to how a system behaves when it is disturbed from its resting position. Unstable systems, upon some disturbance, fail to go back to their equilibrium positions, their motion often gets bigger in amplitude and eventually result in catastrophic failure. Stable systems, on the other hand, can return to their equilibrium positions. Their desired states are easier to maintain and control. We often ensure stability by extracting energy from the system with the form of damping.

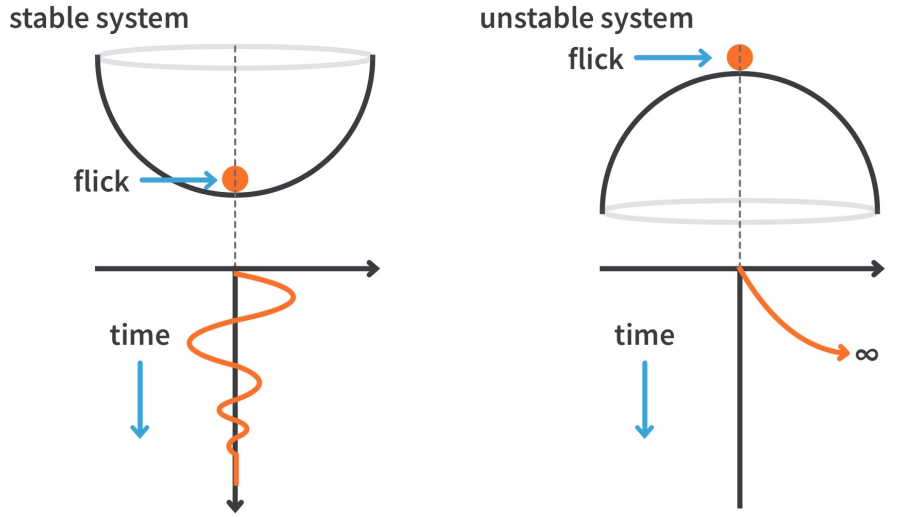

FIGURE 2.5: Analogy for Stability

Figure 2.5 is an analogy of stable versus unstable systems. For the system on the left-hand side, if we are to initially disturb the ball, the ball will then oscillate but will also continuously lose energy because of the friction. It will then come to a stop at its equilibrium position. For the system on the right-hand side, on the other hand, upon an even the slightest of disturbance, the ball would not be able to return to its equilibrium position. It's not desirable to have an unstable system.

2.1.6 Vibration measurements

Vibration measurements can be taken by various methods. Those methods include the use of displacement probes, velocity transducers, accelerometers and even high speed video camera or lasers. The most common way to measure vibrations however is to use accelerometers.

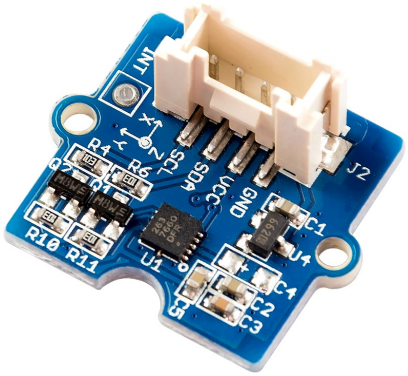

FIGURE 2.6: SeeedStudio Grove 3 Axis Digital Accelerometer

Often times, these compact accelerometers come with a PC software that can show measurement results and plot useful graphs. Those data then can be exported to Excel, and from Excel to any desired programming language. Note that these accelerometers measure acceleration, not position. If the bundle software does not do it automatically, we can calculate position data by integrating the acceleration data with respect to time, twice.

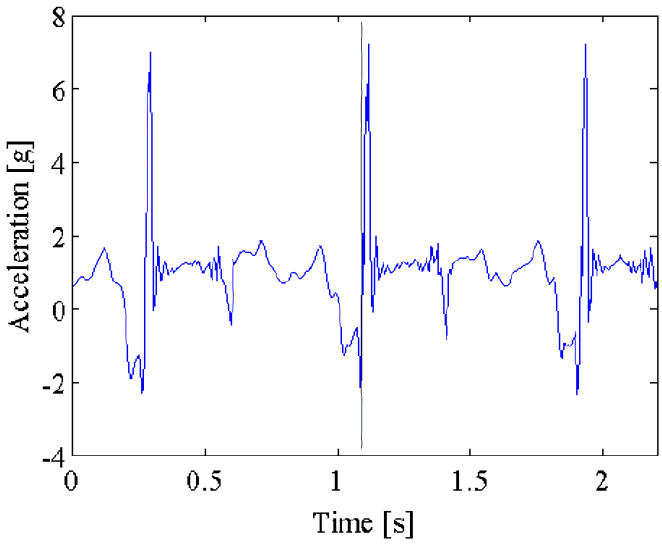

FIGURE 2.7: Plot of an Example Vibration Data out of an Accelerometer

Placement of accelerometers also affect the measurement readings. Accelerometers should be mounted as close to the component of interest as possible. They should also be mounted on hard, rigid surfaces. Best practices include mounting these on bearing housings, coupling guards etc.

2.2 Expert Systems

An expert system is a computer program that uses AI to emulate the judgment and behavior of a human that has expertise and experience in a particular field. They can be used to both replace or aid the human expert. They perform the best if they are limited to a small scope. It's not possible to create an expert system to replace a human's every function, but as an example, it is possible to create one to replace various tasks. A very good but basic example is the task of filtering and sorting objects. A human, in his mind, will categorize objects by their color, size, shape, etc., and depending on the rules that he compares those attributes to, he makes a decision. This and similar processes can be mimicked by an expert system.

2.2.1 How expert systems work

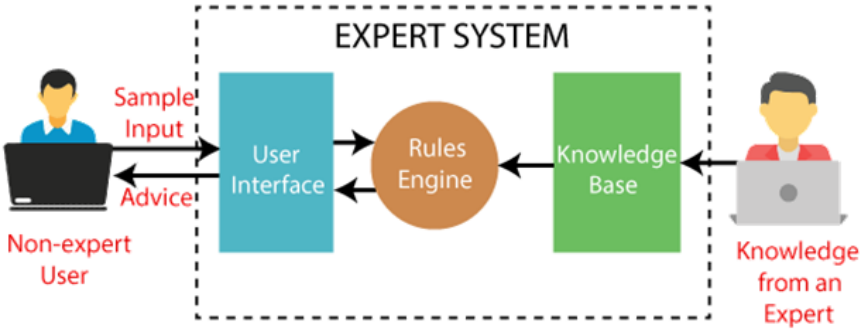

FIGURE 2.8: High Level Process Diagram for Expert Systems

Figure 2.8 shows how the components of an expert system communicate with each other. It all starts when a non-expert user interacts with the user interface. Then, the user interface, designed for collecting data from the non-expert user, sends this data to the rules engine. There, the rules engine starts to compare the data received with the information that is stored within a knowledge base. Instead of a human expert deducing diagnosis and offering advice, the expert

system imitates the human expert. Based on the rules and algorithm found in the rules engine, the system reaches a conclusion. This conclusion will be sent back to the UI, and the non-expert user receives his diagnosis and/or advice.

Though different types of expert systems exist, most of them are categorized by their methods used in their rules engine (also called inference engine). Most common type of expert system is called Rule Based Expert System. These kind of systems often consist of numerous if-then rule blocks. Our rule engine, however, will utilize Case Based Reasoning. This type of expert systems are sometimes called Similarity Based Expert Systems. Though we will also use some if-then rule blocks in our project.

2.2.2 User interface

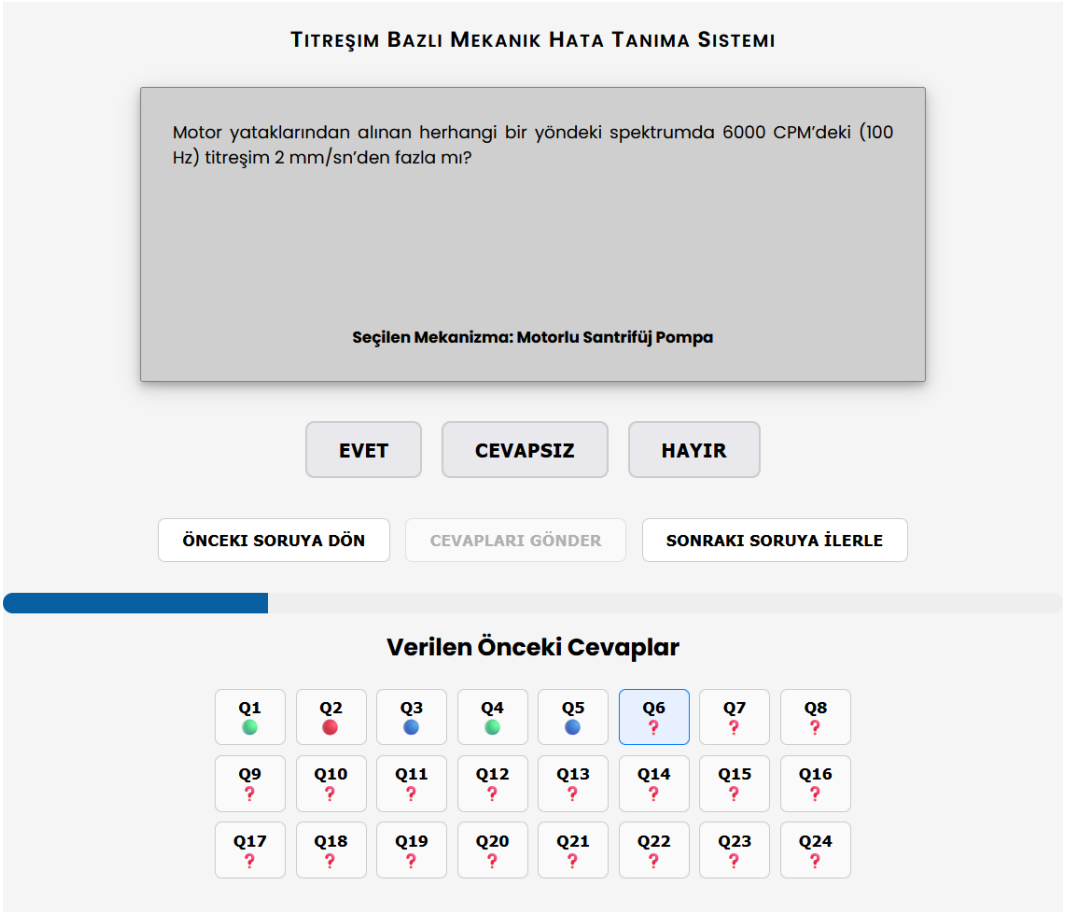

FIGURE 2.9: Example of the UI Where the User is Answering Questions

Figure 2.9 shows a part of the UI where the user is answering questions. As can be seen, the user has many ways to interact with the UI. The UI facilitates communication between the user and the program. Every action a user should or can take will be converted into a form that the program can understand and operate. Because of the UI, functions and responses from the program can be better understood by the user.

2.2.3 Knowledge base

Knowledge base of an expert system contains crucial information that are supplied by the human expert. This information can be in the form of instructions, advice, relations, rules, attributes, etc. Often most important part of an expert system is its knowledge base. The entries inside the knowledge base should be in a form that machines can read and understand.

2.2.4 Rules engine (Inference engine)

Rules engine is an algorithm that operates on the rules provided inside the knowledge base to reach a conclusion. Most common methods rules engines use are forward chaining and backward chaining. Briefly, forward chaining starts from the known facts in the knowledge base and applies every rule whose conditions are met to derive all possible conclusions in a data-driven, bottom-up fashion. Conversely, backward chaining begins with a target conclusion or hypothesis and works backwards through the rule set, seeking the specific rules and premises that must hold true to justify that outcome. Our inference engine, on the other hand, will utilize a concept called Cosine Similarity that will be explained in the next section.

2.2.5 Some influential expert systems

Deep Blue

Deep Blue was a chess-playing expert system that marked the point where a machine exceeded the capabilities of men on the battlefield of a chess board. It was run on a supercomputer specifically designed and built for this task by IBM. It cost $10,000,000 to build in 1995-1997. Adjusted for inflation, that's $20,000,000 of today's money. Today, the chess engines' capabilities far exceed that those of the best chess grandmasters. And even a $100 GPU that is modern can outperform the Deep Blue supercomputer in computational power by orders of magnitude.

FIGURE 2.10: Chess Playing Supercomputer Deep Blue

In 1996, then world chess champion, Garry Kasparov, still regarded as the greatest chess player of all time, managed to win against Deep Blue. But after it sparked so much interest and IBM improving the supercomputer, a rematch in 1997 was arranged. In that rematch, the Deep Blue beat the world champion. Now, the chess playing expert systems are far superior to even the best human chess players. Even your smartphone can beat every chess world champion that ever lived.

FIGURE 2.11: Chess match between Garry Kasparov and Deep Blue

MYCIN

MYCIN is one of the earliest and most famous expert systems, developed in the 1970s at Stanford University. It was designed to assist medical professionals in diagnosing and treating bacterial infections, particularly blood infections and meningitis. It also considered patients body weight in determining correct antibiotic dosage.

It's a Rule Based Expert System. So, it operated on IF-THEN type of rules. Below is an actual piece of code from its program. The actual code for MYCIN consisted of more than 600 rules.

```
(defrule 52
 if (site culture is blood)
  (gram organism is neg)
  (morphology organism is rod)
  (burn patient is serious)
 then .4
  (identity organism is pseudomonas))
Rule 52:
 If
  1) THE SITE OF THE CULTURE IS BLOOD
  2) THE GRAM OF THE ORGANISM IS NEG
  3) THE MORPHOLOGY OF THE ORGANISM IS ROD
  4) THE BURN OF THE PATIENT IS SERIOUS
 Then there is weakly suggestive evidence (0.4) that
  1) THE IDENTITY OF THE ORGANISM IS PSEUDOMONAS
```

FIGURE 2.12: Piece of Code from MYCIN

Looking at Figure 2.12, we can see that there is a rule being defined. Rule above in plain English;

IF the site of the culture is blood **AND** the gram organism is neg **AND** the morphology of the organism is rod **AND** the burn of the patient is serious **THEN** there is a %40 chance that the identity of the organism is pseudomonas.

Though MYCIN was regarded as one of the first successful Expert Systems. And though it achieved on par if not higher performance than those of a group of specialists at trials, it was never adopted and saw actual real- world use. This wasn't because of any negative results but was over ethical concerns.

2.3 Linear Algebra

2.3.1 Vectors

Vectors in linear algebra are mathematical entities with both magnitudes and directions. They differ from scalars as scalars only have magnitudes and not directions.

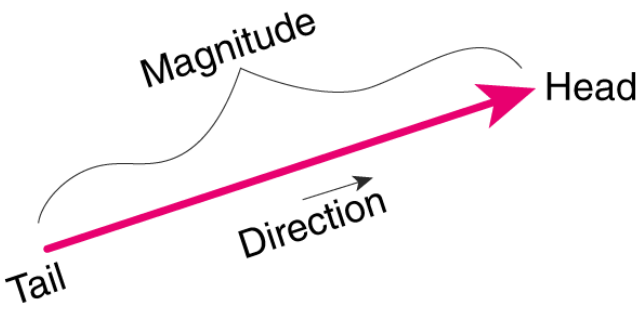

FIGURE 2.13: Vector Defined

In linear algebra, vectors are often expressed as;

$$\mathbf{v} = \begin{pmatrix} v_1 \\ v_2 \\ \vdots \\ v_n \end{pmatrix} \tag{2.16}$$

Vectors only have one column and are $[n \times 1]$ in size. Magnitude of a vector can be calculated using this formula;

$$\|\mathbf{v}\| = \sqrt{v_1^2 + v_2^2 + \cdots + v_n^2} \tag{2.17}$$

2.3.2 Dot product

Dot product is one of the fundamental operations that combine two vectors and produce a scalar. The dot product can be seen as one vector's length multiplied by the length of the component of the other vector along it.

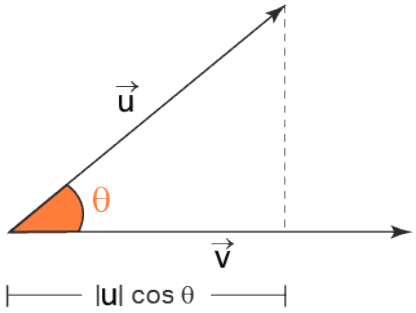

FIGURE 2.14: Dot Product

$$\mathbf{u} \cdot \mathbf{v} = \|\mathbf{v}\|(\|\mathbf{u}\| \cos\theta) \tag{2.18}$$

Equation 2.18 shows the mathematical expression for dot product of two vectors. As we can observe, if the angle between these two vectors is $\theta = 90\,\text{deg}$, then these vectors are linearly independent of each other and $\mathbf{u} \cdot \mathbf{v} = 0$. On the other end, if the angle between these two vectors is $\theta = 0\,\text{deg}$, then these vectors are parallel and codirectional. So we can say that as the angle between two vectors decreases, the amount of similarity between them increases. This will be useful.

2.3.3 Cosine Similarity

Cosine similarity S_c is essentially a normalized version of the dot product.

$$S_c = \frac{\mathbf{u} \cdot \mathbf{v}}{\|\mathbf{u}\| \ \|\mathbf{v}\|} \tag{2.19}$$

Or in the scalar form;

$$S_c = \frac{\|\mathbf{v}\|(\|\mathbf{u}\| \cos\theta)}{\|\mathbf{u}\| \ \|\mathbf{v}\|} \tag{2.20}$$

The reason why we introduce cosine similarity is, whilst the dot product can give an idea of alignment between two vectors, its results are highly influenced by the magnitude of its vectors. Two large vectors that are 20 deg apart can yield a higher result than two small vectors that are 10 deg apart. Even though the small vectors are more aligned. Cosine similarity solves this problem, so that if the

cosine similarity score between two vectors are higher than those of other two, then they are definitely more aligned. Note that cosine similarity scores can be in the range $[-1 < S_c < 1]$.

TABLE 2.2: Meaning of Different Similarity Scores Based on Figure 2.14

Similarity Score	θ	Description
$S_c = -1$	$\theta = 180\,\text{deg}$	Two vectors are parallel but in directly opposite directions
$-1 < S_c < 0$	$90\,\text{deg} < \theta < 180\,\text{deg}$	Two vectors face towards opposite directions
$S_c = 0$	$\theta = 90\,\text{deg}$	Two vectors are linearly independent
$0 < S_c < 1$	$0\,\text{deg} < \theta < 90\,\text{deg}$	Two vectors face towards similar directions
$S_c = 1$	$\theta = 0\,\text{deg}$	Two vectors are parallel and codirectional

3. Implementation

To recall, the aim of this project is to offer an alternative to traditional fault diagnosis methods, and to replace a human expert with an artificially intelligent expert program. We want to ask a series of expert questions to a user. Depending on the answers, we will then use stored expert knowledge to try and identify the most likely fault diagnosis. This makes sure that every device can access our program as long as they have an internet connection.

3.1 Software and Libraries Used

Since this is not a report for a software engineering degree, I will not go into detail on software used, but briefly explain what they are and what they do. As can be seen on Figure 3.4, the project consists of a backend and a frontend.

3.1.1 Backend

For the backend, I will use a programming language called Python. I will both code the actual algorithm and store the required knowledge in Python. As for Python libraries, we mainly rely on NumPy and Python's own math library.

3.1.2 Frontend

For the frontend, since this is a web application, I have used HTML, CSS and JavaScript. HTML and CSS allowed me to markup and style everything that you see in the UI. JavaScript via an app framework for it called SvelteKit, allowed me to handle frontend data, function, animations, relations, etc. SvelteKit also requires Node.js to be present on the host system.

3.1.3 Connecting frontend and backend

to connect frontend and backend, I have used a Python library called FastAPI. It allowed me to send and receive requests between frontend and backend, and vice versa.

3.1.4 Hosting the web application

To host the entire stack of application, I have rented a virtual private server from a hosting company. My application bundled with an application called Docker is then uploaded into that virtual server. Again, the application will be available to be tested with the URL;

www.batacoglu.com/GraduationProject

3.2 Main Idea

The algorithm for this application is fairly simple. The idea is that, if we can create vectors for every possible fault diagnoses, and the **n***th* row entries of these vectors correspond to **n***th* question/answer set, then we can utilize the cosine similarity principle (Equation 2.19) to check how aligned these vectors are. Based on these similarity scores, we can rank all diagnoses for their likelihood of occurrence. We have used Python to code the logic.

Imagine three vectors $\mathbf{A}$, $\mathbf{B_1}$ and $\mathbf{B_2}$.

Where $\mathbf{A} = \begin{pmatrix} 3 \\ \\ 3 \end{pmatrix}$, $\mathbf{B_1} = \begin{pmatrix} 2 \\ \\ 5 \end{pmatrix}$, $\mathbf{B_2} = \begin{pmatrix} -2 \\ \\ 1 \end{pmatrix}$

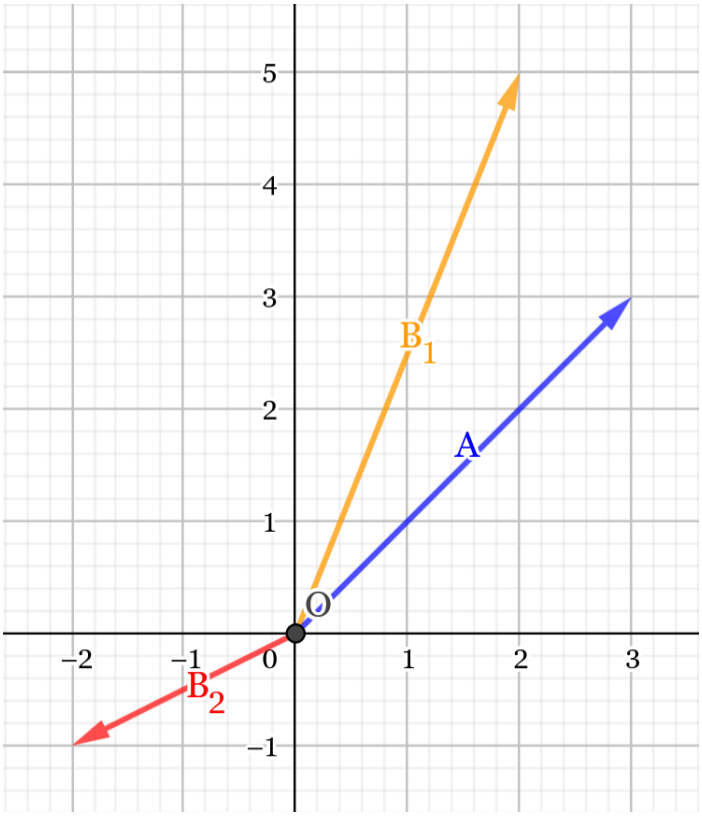

FIGURE 3.1: 3 Vectors Showing the Main Idea Behind the Project

As we can deduct from Figure 3.1, while $\mathbf{B_1}$ looks somewhat aligned with $\mathbf{A}$, while $\mathbf{B_2}$ looks to be in the opposing direction. So we would expect similarity score of $\mathbf{A}$ and $\mathbf{B_1}$ to be in high positives, while similarity score of $\mathbf{A}$ and $\mathbf{B_2}$ to be in low negatives. If $\mathbf{B_1}$ and $\mathbf{B_2}$ were to represent two different possible fault diagnoses, and $\mathbf{A}$ were to represent the answers provided by the user, then we could say that Diagnosis represented by $\mathbf{B_1}$ had a high probability of occurrence and Diagnosis represented by $\mathbf{B_2}$ were impossible to occur.

In actual code, for the mechanism of centrifugal pumps, we have 22 possible fault diagnoses and 24 possible symptoms. That means with the addition of vector $\mathbf{A}$, and 22 other fault diagnosis vectors $\mathbf{B_1}$ through $\mathbf{B_{22}}$, we have a total of 23 $[24 \times 1]$ vectors. Let's put everything in a matrix.

$$DiagnosisMatrix; \mathbf{B} = \begin{bmatrix} \vdots & \vdots & \vdots & \vdots & \vdots & 0 & 0 \\ \vdots & \vdots & \vdots & \vdots & \vdots & 0 & 0 \\ \mathbf{B_1} & \mathbf{B_2} & \cdots & \cdots & \mathbf{B_{22}} & 0 & 0 \\ \vdots & \vdots & \vdots & \vdots & \vdots & 0 & 0 \\ \vdots & \vdots & \vdots & \vdots & \vdots & 0 & 0 \end{bmatrix} [24 \times 24] \tag{3.1}$$

In Equation 3.1, the last two columns of zeros were added to ensure compatibility of size during matrix multiplication. Each column represents a diagnosis vector.

```
844  DIAGNOSIS_MATRIX = np.vstack(
845      (
846          DIAGNOSIS_1,
847          DIAGNOSIS_2,
848          DIAGNOSIS_3,
849          DIAGNOSIS_4,
850          DIAGNOSIS_5,
851          DIAGNOSIS_6,
852          DIAGNOSIS_7,
853          DIAGNOSIS_8,
854          DIAGNOSIS_9,
855          DIAGNOSIS_10,
856          DIAGNOSIS_11,
857          DIAGNOSIS_12,
858          DIAGNOSIS_13,
859          DIAGNOSIS_14,
860          DIAGNOSIS_15,
861          DIAGNOSIS_16,
862          DIAGNOSIS_17,
863          DIAGNOSIS_18,
864          DIAGNOSIS_19,
865          DIAGNOSIS_20,
866          DIAGNOSIS_21,
867          DIAGNOSIS_22,
868          np.zeros(24),
869          np.zeros(24),
870      )
871  )
872
873  DIAGNOSIS_MATRIX_TRANSPOSED = DIAGNOSIS_MATRIX.T
```

FIGURE 3.2: Creating of Equation 3.1 in Python

Remember that vector **A** represents the user answers. Users answers to question can only be one of three ways, the user can either say yes, no or decide not to answer that question.

$$\mathbf{A} = \begin{bmatrix} Answer1 & Answer2 & \cdots & \cdots & Answer23 & Answer24 \end{bmatrix} [1 \times 24] \tag{3.2}$$

Now, We will apply Cosine Similarity to answers vector **A** and each column of **B**.

```
10  def CosineSimilarity(ANSWERS, MATRIX, DOTTED):
11      MAGNITUDE_A = np.linalg.norm(ANSWERS)
12      COSINE_SIMILARITY_MATRIX = DOTTED.copy()
13
14      for index in range(len(COSINE_SIMILARITY_MATRIX)):
15          MAGNITUDE_B = np.linalg.norm(MATRIX[index])
16          DENOMINATOR = MAGNITUDE_A * MAGNITUDE_B
17          if DENOMINATOR != 0:
18              COSINE_SIMILARITY_MATRIX[index] = (
19                  COSINE_SIMILARITY_MATRIX[index] / DENOMINATOR
20              )
21
22      return COSINE_SIMILARITY_MATRIX
```

FIGURE 3.3: Cosine Similarity Function Used in The Project

Figure 3.3 shows the implementation of cosine similarity in Python. This function also filters out negative scores, since they are of no use to us.

3.3 Creating the Knowledge Base

The Knowledge Base is a part of the backend and is unavailable for the user to directly access. It contains information supplied by the human expert. The information stored here include possible fault types, causes, symptoms, relations, etc. This information is stored in a database. Knowledge for this project are mainly stored in the form of vectors. For example, vector $\mathbf{B}_1$, corresponding to the fault diagnosis of pump unbalance is given below.

$$\mathbf{B_1} = \begin{bmatrix} 2 & 1 & -1 & -1 & -1 & -2 & -1 & 0 & \cdots & 0 \end{bmatrix}^T [24 \times 1] \tag{3.3}$$

Equation 3.3, carries with itself much useful information. Firstly, since its index is 1, we can associate with it the title pump unbalance. Also, each of its entry corresponds to how a question relates to the probability of this diagnosis occurring.

TABLE 3.1: Possible Entries of Diagnosis Vectors and Their Effects

Entries	Description
-2	Answer corresponding to a symptom with this index strongly decreases the chance of this diagnosis
-1	Answer corresponding to a symptom with this index slightly decreases the chance of this diagnosis
0	Answer corresponding to a symptom with this index has no effect on this diagnosis
1	Answer corresponding to a symptom with this index slightly increases the chance of this diagnosis
2	Answer corresponding to a symptom with this index strongly increases the chance of this diagnosis

```
# 22. Pompada kavitasyon
DIAGNOSIS_22 = np.array(
    [
        0,  # Correlation to Answer 1
        0,  # Correlation to Answer 2
        0,  # Correlation to Answer 3
        0,  # Correlation to Answer 4
        0,  # Correlation to Answer 5
        0,  # Correlation to Answer 6
        0,  # Correlation to Answer 7
        1,  # Correlation to Answer 8
        0,  # Correlation to Answer 9
        0,  # Correlation to Answer 10
        2,  # Correlation to Answer 11
        2,  # Correlation to Answer 12
        0,  # Correlation to Answer 13
        0,  # Correlation to Answer 14
        -1,  # Correlation to Answer 15
        -1,  # Correlation to Answer 16
        0,  # Correlation to Answer 17
        0,  # Correlation to Answer 18
        -1,  # Correlation to Answer 19
        -1,  # Correlation to Answer 20
        0,  # Correlation to Answer 21
        0,  # Correlation to Answer 22
        -1,  # Correlation to Answer 23
        -1,  # Correlation to Answer 24
    ]
)
```

FIGURE 3.4: An Example on How Relation Data is Stored Between Questions and a Diagnosis

3.4 User Interface

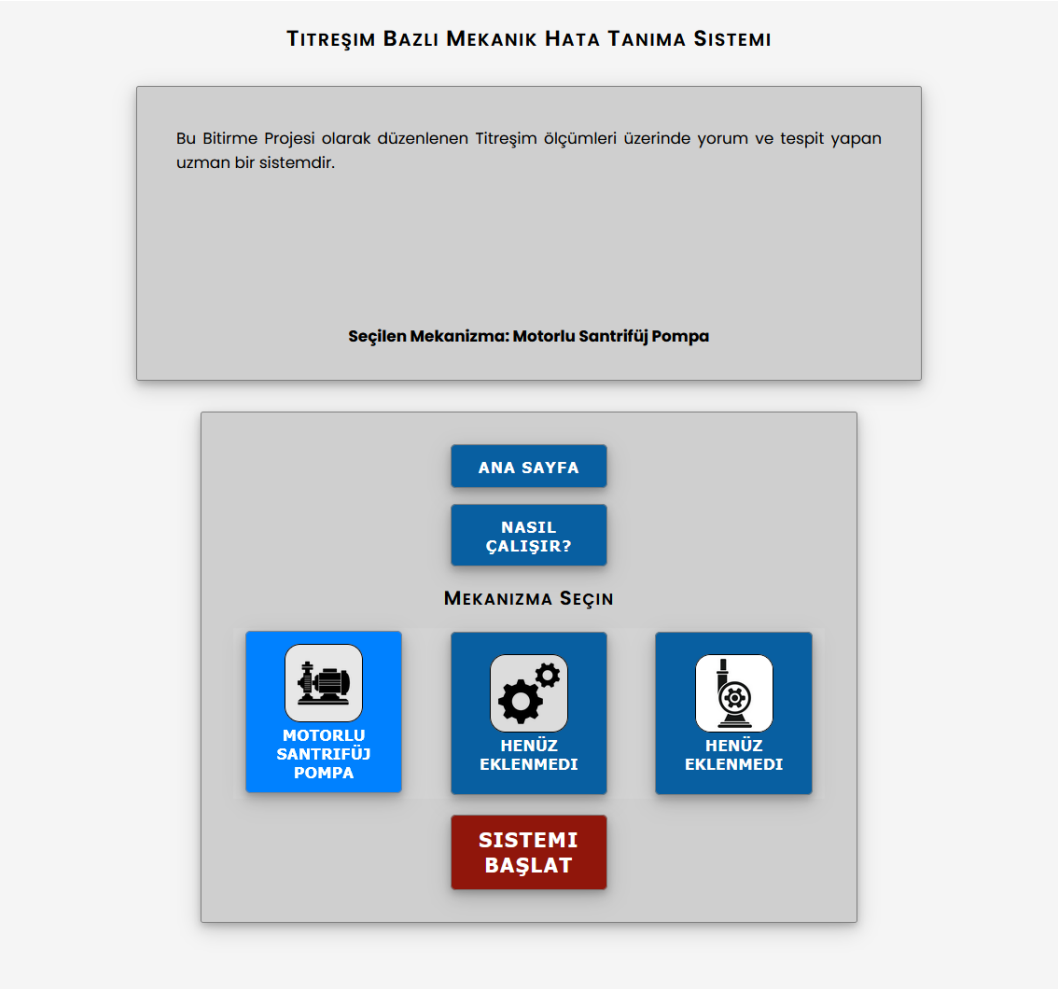

FIGURE 3.5: Main Page for the Web Application

Figure 3.5 shows the main menu for the project. At the top there is the project title, right below is the text box. Content inside this text box dynamically changes depending on the state of the application. For example, if you were to start the program, there would be questions, and if you were to click on the how it works button, it would list you instructions on how to operate the application. As you can see, there's only centrifugal pump with included electric motor as a mechanism is implemented in the current state of the program.

Going a bit back, Figure 2.9 shows a part of the UI where the user is answering questions. The answers are stored locally on the user's web cache. The user can at any time go back to previously answered questions and check for his own

answers. After the user is satisfied with his answers, he then submits his results, and the frontend bundles the answers and sends it to the backend.

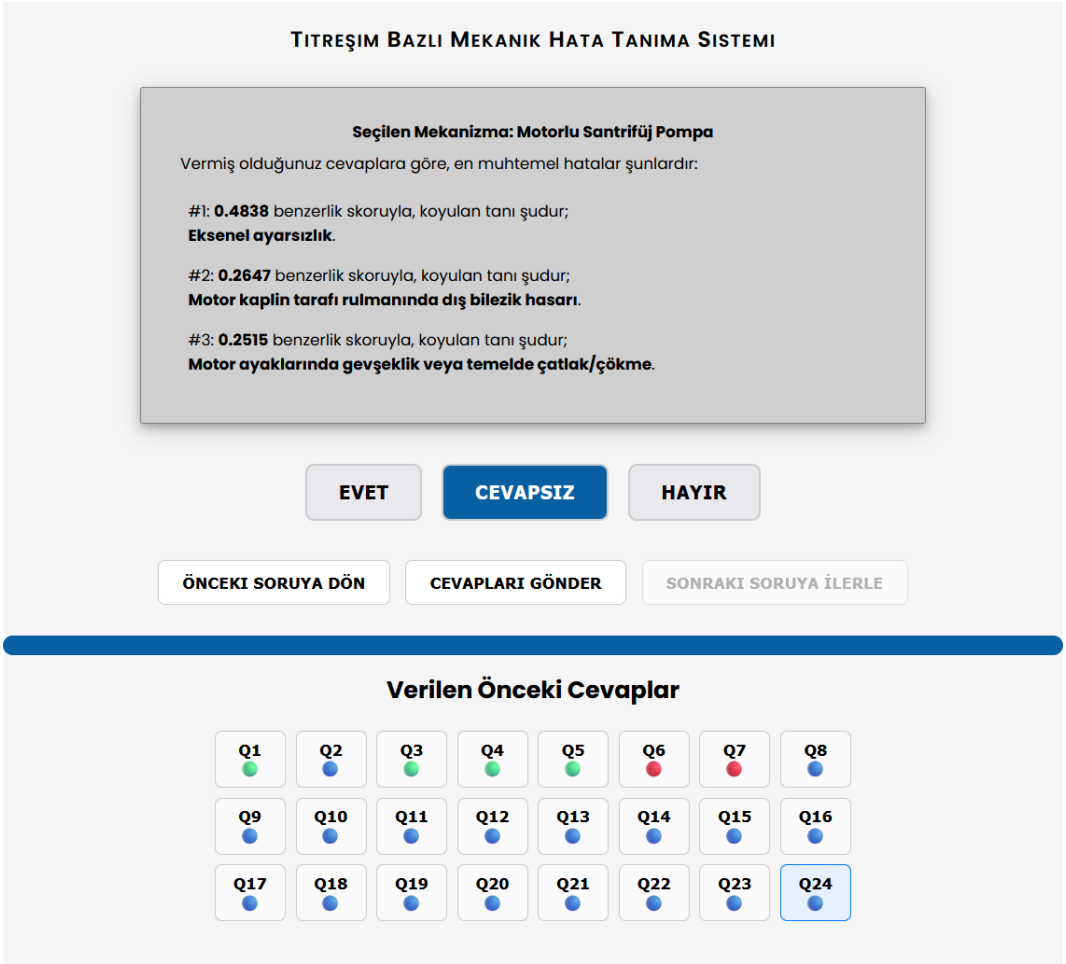

FIGURE 3.6: Example Results Being Shown to the User

Figure 3.6 shows one possible conclusion to user input. As can be seen, according to the user's input, the most likely result is axial misalignment.

4. Conclusions

In this work, we built a simple web-based expert system for vibration-based fault diagnosis of centrifugal pumps with electric motors. We gathered expert knowledge and relevant ISO standards, and turned each possible fault into a 24-entry vector. By asking the user 24 yes/no questions, converting their answers into a vector, and computing cosine similarity against each fault vector, the system ranks and displays the most likely causes.

The application uses a Python backend with NumPy for vector math, and a SvelteKit frontend connected via FastAPI. Docker is used for deployment. Through a clean, question-and-answer interface, non-expert users can quickly receive a list of probable faults without needing deep vibration-analysis skills.

While the prototype works well for centrifugal-pump diagnostics, it has two main limits:

- It covers only one machine type and depends on manually coded expert vectors.
- It uses only user-entered answers; no live sensor data is yet integrated.

To make the system more useful, future work could:

- Add other machines (gears, fans, motors) by creating new fault vectors.
- Connect to real-time vibration or acceleration streams to auto-generate answer vectors.
- Introduce simple learning: let the system update its vectors based on real diagnosis feedback.

Overall, this project shows that a lightweight AI approach can offer a low-cost, easy-to-deploy tool for basic machinery fault diagnosis. With further expansion and input of more data, it could become a practical aid in everyday maintenance tasks.

The web application can be accessed via:

www.batacoglu.com/GraduationProject

References

Robert C. Eisenmann Sr. and Robert C. Eisenmann Jr., *Machinery Malfunction Diagnosis and Correction: Vibration Analysis and Troubleshooting for the Process Industries*, MACHINERY DIAGNOSTICS, Inc. (Minden, Nevada) and HAHN & CLAY (Houston, Texas), 1998.

Hosameldin Ahmed and Asoke K. Nandi, *Condition Monitoring with Vibration Signals: Compressive Sampling and Learning Algorithms for Rotating Machines*, Brunel University London, UK, 2020.

William T. Thomson and Marie Dillon Dahleh, *Theory of Vibration with Applications*, 5th ed., Pearson Education, 1997.

Peter G. Steeneken, "Forced Vibrations," in *Introductory Dynamics: 2D Kinematics and Kinetics of Point Masses and Rigid Bodies*, LibreTexts.

ISO 13373-3:2015, *Condition Monitoring and Diagnostics of Machines — Vibration Condition Monitoring — Part 3: Guidelines for Vibration Diagnosis*.

ISO 10816-1:1995, *Mechanical Vibration — Evaluation of Machine Vibration by Measurements on Non-Rotating Parts — Part 1: General Guidelines*.

ISO 10816-7:2009, *Mechanical Vibration — Evaluation of Machine Vibration by Measurements on Non-Rotating Parts — Part 7: Rotodynamic Pumps for Industrial Applications*.

ISO 10816-8:2014, *Mechanical Vibration — Evaluation of Machine Vibration by Measurements on Non-Rotating Parts — Part 8: Reciprocating Compressor Systems*.