DEVELOPMENT OF A MODEL AND IMBALANCE DETECTION SYSTEM FOR THE CAL POLY WIND TURBINE

A Thesis

presented to

the Faculty of California Polytechnic State University,

San Luis Obispo

In Partial Fulfillment

of the Requirements for the Degree

Master of Science in Mechanical Engineering

by

Ryan Takatsuka

April 2019

COMMITTEE MEMBERSHIP

TITLE: Development of a Model and Imbalance Detection System for the Cal Poly Wind Turbine

AUTHOR: Ryan Takatsuka

DATE SUBMITTED: April 2019

COMMITTEE CHAIR: John Ridgely, Ph.D.
Professor of Mechanical Engineering

COMMITTEE MEMBER: Glen Thorncroft, Ph.D.
Professor of Mechanical Engineering

COMMITTEE MEMBER: Patrick Lemieux, Ph.D.
Professor of Mechanical Engineering

ABSTRACT

Development of a Model and Imbalance Detection System for the Cal Poly Wind Turbine

Ryan Takatsuka

This thesis develops a model of the Cal Poly Wind Turbine that is used to determine if there is an imbalance in the turbine. A theoretical model is derived to estimate the expected vibrations when there is an imbalance in the rotor. Vibration and acceleration data are collected from the turbine tower during operation to confirm the model is useful and accurate for determining imbalances in the turbine.

Various signal processing techniques for analyzing the vibration data are explored and tested with the simulation. This includes frequency shifts, lock-in amplifiers, phase-locked loops, discrete Fourier transforms, and various filters. The processed data is fed into an algorithm that determines if there is an imbalance.

The detection algorithm consists of a machine learning classification model that uses experimental data to train and increase the success rate of the imbalance detection. The model uses the maximum frequency component and magnitude as an input to classify the data as "balanced" or "not balanced" using the K-nearest neighbors algorithm. While this classification algorithm requires slightly more computational power than others, it is simple to implement and has a high accuracy rating.

TODO: Add a quick summary of the results.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

Chapter 1

INTRODUCTION

TODO: Add a short section about the significance of wind turbine failures and the advantages of using conditioning monitoring.

1.1 System Overview

The wind turbine that is analyzed in this thesis is part of the Cal Poly Wind Power Research Center. This is designed for research into smaller wind turbines and to educate engineers in all aspects of the wind power industry. The Cal Poly Wind Turbine Tower supports a 3 kW Horizontal-Axis Wind Turbine, and the tower was analyzed by Tae-gyun (Tom) Gwon[11]. In Gwon's thesis paper, an ABAQUS model of the tower is developed to analyze natural frequencies and vibrations. An FEM model, such as this one, is too complicated to run on a cheap microcontroller; however, the results of this model can be compared with the simple lumped-parameter model to determine the validity of the simplifications.

The Cal Poly Wind Turbine has no current method of detecting an imbalance. It is possible to install an intrusive and expensive device that monitors all many tower parameters (such as multiple acceleration measurements at various positions on the tower and blades), but it would be much more desirable to have an inexpensive, device that can accurately detect an imbalance with zero to no setup/installation effort.

1.2 Objective

The primary objective of this thesis is to develop a method for identifying a blade imbalance in the field. This method must be simple enough to perform on a micro-controller in real time, while maintaining the ability to be mounted to any small scale wind turbine. This will help to prevent any catastrophic failures resulting in damaged blades and an inoperable wind turbine. Additionally, maintenance costs can be reduced because the turbines in good condition won't have to be inspected as often. The main focus of the paper will be introducing a simplified turbine tower model and providing a digital signal processing method for analyzing the data.

A simplified tower model will help to develop and test various signal processing methods. It is difficult and time consuming to obtain experimental data from the tower, so having a tower model will speed up the algorithm design process. Once the signal processing method has been tested and refined on the analytic model, it can then be tested on experimental tower data will minimal tuning.

1.3 Background

Cal Poly's wind turbine is in the Escuela Ranch in an unpopulated area. The tower is a tapered tubular pole made of ASTM A572 Grade-50 Steel. The tower has a tilting feature which allows relatively easy access to the nacelle. The tower is rotated about 2 ball bearings at the base via a winch attached to the CPWPRC truck. More details about the tower design and analysis can be found in *Structural Analyses of Wind Turbine Tower for 3 KW Horizontal-Axis Wind Turbine* [11].

SCADA System	WT Make	Technology	Rated Capacity (kW)	Nb of Turbines	Failures per Turbine	Alarms per Turbine
1	A	Geared	1500	55	0.709	4170.07
2	B, C	Dir. Drive	2000	57	0.632	1120.35
3	D	Geared	850	77	2.208	2778.78
4	E	Geared	2000	168	1.780	4704.57
5	F, G	Geared	1800 & 2000	83	1.313	572.14

Figure 1.1: Data used for the SCADA Alarms and Failure Analysis [15]. TODO: Describe/explain the results in the SCADA table.

1.4 Turbine Failures

The goal of this project is to develop a method for detecting a rotor imbalance that could be potentially harmful to the turbine. This would allow the turbine to be shut down before any failures occur.

A common method for tracking and preventing turbine failures is the use of Supervisory Control and Data Acquisition (SCADA) alarms[9]. SCADA is a control system architecture that uses high-level graphical user interfaces networked with peripheral devices (such as PLCs and PID controllers). This is an effective method for predicting turbine failure, but can be expensive and complicated to implement. An example of SCADA alarm results are shown in Figure 1.1.

According to a report on UK wind farms[19], there have been 1500 wind turbine-related accidents within 5 years. Of those accidents, there were a total of 400 injuries to workers. Many of these accidents are preventable if there is a reliable self-diagnosing system installed on the turbine. This report also mentions incidents where ice has build up on the blades. Most of these failures should be preventable if the rotor imbalance was detected right away and the turbine was shut down.

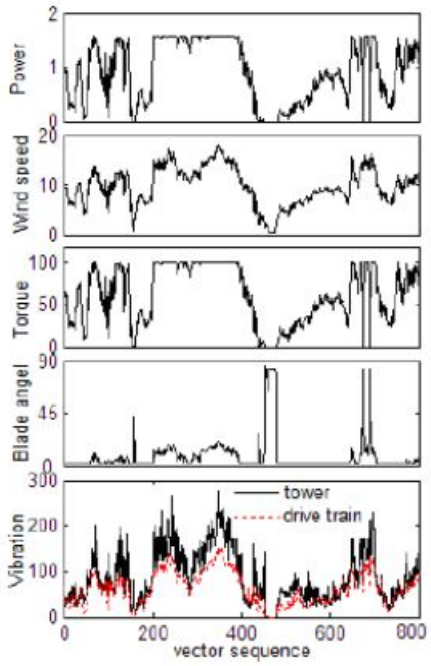

Figure 1.2: Correlation between different turbine parameters [17]. TODO: get less blurry picture and add some more explanation.

1.5 Tower Vibration Research

Wind turbine tower vibrations are important to the health of turbines and have been studied before. In one study, a nonlinear state estimation technique[17] (NSET) is used to attempt to predict wind turbine failures. This model uses SCADA data to relate different operating parameters to the health of the turbine. Figure 1.2 shows the correlation of power, wind speed, torque, blade angle, and tower vibration. According to this paper, the tower vibrations are not enough information to predict turbine failures.

The research in [17] attempts to detect *any* turbine failure, while detecting only rotor imbalances may be slightly easier.

Most tower vibration analyses are empirically derived because of the stochastic nature of the wind speed and the amount of variables affecting the turbine. Data-driven models[20] are very common in the wind power space, especially with monitoring systems such as SCADA.

Chapter 2

MATHEMATICAL MODEL

2.1 Assumptions/Problem Statement

The turbine tower is modeled as a cantilever beam with a concentrated mass at the end. The tower is assumed to be the cantilever beam, and the nacelle is assumed to be the concentrated mass at the end of the simplified beam.

This problem is treated as forced vibration, where the forcing function is a periodic force caused from an eccentric mass on the rotor. This eccentric mass represents an imbalance in the turbine, and will cause vibrations in the tower. Figure 2.1 shows the simplified wind turbine model with the applied forces. The cross-section of the tower is assumed to taper linearly from the base to the top.

2.2 Variable Definitions

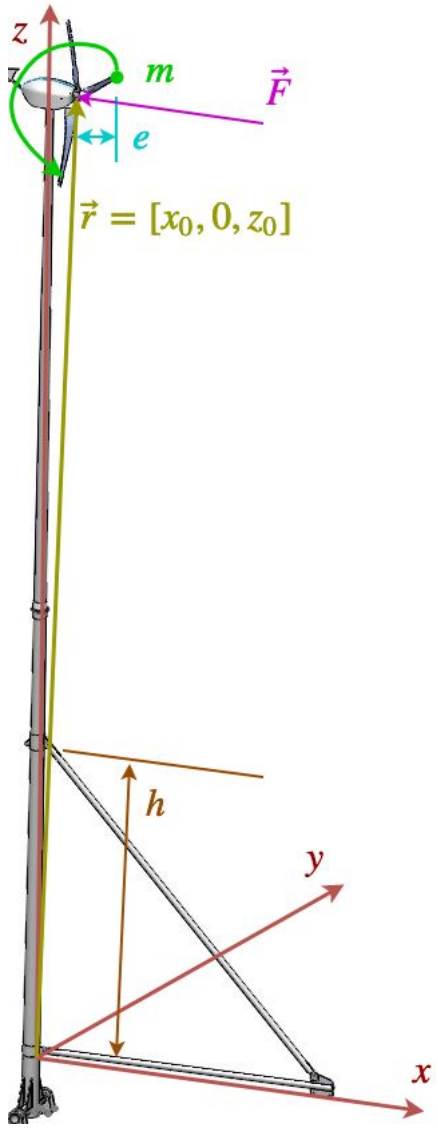

Figure 2.1: The simplified wind turbine model showing the forces from an eccentric mass on the rotor. This figure also shows the coordinate system that will be used for the tower analysis.

Table 2.1: Variable Definitions

Variable	Description
$\vec{F}_{original}$	The original force applied to the end of the rotor due to an imbalance in the blades
F_0	The magnitude of the force caused by the eccentric mass
ω	The rotational speed of the blades
t	Time
m	Eccentric mass
e	Distance the eccentric mass is from the axis of rotation of the rotor
$\vec{M}$	Moment that the eccentric force applies after the force is translated to the tower axis
$\vec{r}$	The position vector for the top of the rotor with respect to the bottom of the turbine tower ($r = [x_0, 0, z_0]$)
M_x	The moment function in the x-direction
M_y	The moment function in the y-direction
I_b	The area moment of inertia at the base of the tower
I_t	The area moment of inertia at the top of the tower
h	The distance from the base of the tower to the ginpole pin-joint on the tower
E	Modulus of elasticity of the tower (steel)
d_b	Diameter at the base of the tower
d_t	Diameter at the top of the tower

2.3 Calculating Tower Vibrations

2.3.1 Determining Effective Force

An eccentric mass, m, on the rotor causes a force defined by the following equation:

$$\vec{F}(t) = \begin{bmatrix} 0 \\ F_0 \sin(\omega t) \\ F_0 \cos(\omega t) \end{bmatrix} \tag{2.1}$$

where

$$F_0 = me\omega^2 \tag{2.2}$$

In order to treat the system as a cantilever beam, the force has to be acting in the center of the nacelle and on the same axis as the tower. When the force vector (Equation 2.1) is translated to the axis of the tower, a moment needs to be introduced to account for the force vector translation:

$$\vec{M} = \vec{r} \times \vec{F} = \begin{bmatrix} -F_0 z_0 \sin(\omega t) \\ -F_0 x_0 \cos(\omega t) \\ F_0 x_0 \sin(\omega t) \end{bmatrix} \tag{2.3}$$

The moment in (Equation 2.3) creates an effective force at the top of the tower that needs to be added to the original force (Equation 2.1). The moment about the z-axis is ignored because torsional effects on the tower are negligible when determining nacelle displacement. This results in the following effective force on the top of the tower:

$$\vec{F}_{effective} = \vec{F}_{original} + \vec{F}_{moment} \tag{2.4}$$

$$\vec{F}_{effective} = \begin{bmatrix} 0 \\ F_0 \sin(\omega t) \\ F_0 \cos(\omega t) \end{bmatrix} + \begin{bmatrix} -\frac{F_0 x_0}{z_0} \cos(\omega t) \\ F_0 \sin(\omega t) \\ 0 \end{bmatrix}$$

$$\vec{F}_{effective} = \begin{bmatrix} -\frac{F_0 x_0}{z_0} \cos(\omega t) \\ 2F_0 \sin(\omega t) \\ F_0 \cos(\omega t) \end{bmatrix} \tag{2.5}$$

2.3.2 Determining Effective Spring Constant

The tower model has a different spring constant for the x and y directions. In the y direction, the ginpole has a negligible effect on the stiffness of the tower, so the tower acts as a tapered cantilever beam.

2.3.2.1 Mathematical derivation of the y-direction spring constant, K_y

Assuming the tower acts like a cantilever beam in this direction with a concentrated force at the end of the beam, the moment, $M_x(y)$ can be expressed as:

$$M_x(y) = F_y\, z - F_y\, z_0 \tag{2.6}$$

Because the beam is linearly tapered, the diameter, d, can be written as a function of tower height, z_0.

$$d(z) = d_b - \frac{z\ (d_b - d_t)}{z_0} \tag{2.7}$$

d_b is the diameter of the tower at the bottom, and d_t is the diameter of the tower at the top. The moment of inertia, assuming a hollow tube with thickness t, can be written as a function of z using Equation 2.7.

$$I_y(z) = \frac{\pi}{64} d^4 - \frac{\pi}{64} (d - 2t)^4 \tag{2.8}$$

$$I_y(z) = \frac{\pi \left(d_b - \frac{z\,(d_b - d_t)}{z_0} \right)^4}{64} - \frac{\pi \left(2\,t - d_b + \frac{z\,(d_b - d_t)}{z_0} \right)^4}{64} \tag{2.9}$$

The deflection of the beam can be calculated using the Euler–Bernoulli equation:

$$\frac{d^2 y_d}{dz^2} = -\frac{M_x}{E\, I_y(z)} \tag{2.10}$$

$$\frac{d^2 y_d}{dz^2} = -\frac{F_y\, z - F_y\, z_0}{E \left(\frac{\pi \left(d_b - \frac{z\,(d_b - d_t)}{z_0} \right)^4}{64} - \frac{\pi \left(2\,t - d_b + \frac{z\,(d_b - d_t)}{z_0} \right)^4}{64} \right)} \tag{2.11}$$

The slope of the beam, θ_y, can be calculated by integrating Equation 2.11:

$$\theta_y = \frac{dy_d}{dz} = \int -\frac{F_y\, z - F_y\, z_0}{E \left(\frac{\pi \left(d_b - \frac{z\,(d_b - d_t)}{z_0} \right)^4}{64} - \frac{\pi \left(2\,t - d_b + \frac{z\,(d_b - d_t)}{z_0} \right)^4}{64} \right)} \, dz \tag{2.12}$$

Integrating the equation above results in the following equation for θ_y:

$$\theta_y = \frac{\ln\left(d_b\, z - d_t\, z - d_b\, z_0 + t\, z_0\right)\ \left(8\, F_y\, d_t\, {z_0}^2 - 8\, F_y\, t\, {z_0}^2\right)}{E\,\pi\,{d_b}^2\, t^3 - 2\,E\,\pi\, d_b\, d_t\, t^3 + E\,\pi\,{d_t}^2\, t^3}$$
$$- \frac{4\,\ln\left(d_b\, z - d_t\, z - d_b\, z_0 + t\, z_0\ (1-\mathrm{i})\right)\ \left(F_y\, d_t\, {z_0}^2 + F_y\, t\, {z_0}^2\ (-1+1\mathrm{i})\right)}{E\,\pi\,{d_b}^2\, t^3 - 2\,E\,\pi\, d_b\, d_t\, t^3 + E\,\pi\,{d_t}^2\, t^3}$$
$$- \frac{4\,\ln\left(d_b\, z - d_t\, z - d_b\, z_0 + t\, z_0\ (1+1\mathrm{i})\right)\ \left(F_y\, d_t\, {z_0}^2 + F_y\, t\, {z_0}^2\ (-1-\mathrm{i})\right)}{E\,\pi\,{d_b}^2\, t^3 - 2\,E\,\pi\, d_b\, d_t\, t^3 + E\,\pi\,{d_t}^2\, t^3} + C_{y1} \tag{2.13}$$

The tower is fixed at the base (no translation or rotation about the x-axis), so the slope at $z = 0$ is $\theta = 0$. Using this boundary condition, C_{y1} can be determined:

$$C_{y1} = -\frac{4\,F_y\,{z_0}^2\,Q_1(z)}{E\,t^3\,\pi\left(d_b - d_t\right)^2} \tag{2.14}$$

where,

$$\begin{aligned} Q_1(z) = {} & 2\,d_t\ \ln\left(-z_0\ \left(d_b - t\right)\right) - d_t\ \ln\left(-z_0\ \left(d_b + t\ (-1-\mathrm{i})\right)\right) \\ & - d_t\ \ln\left(-z_0\ \left(d_b + t\ (-1+1\mathrm{i})\right)\right) - 2\,t\ \ln\left(-z_0\ \left(d_b - t\right)\right) \\ & + t\ \ln\left(-z_0\ \left(d_b + t\ (-1-\mathrm{i})\right)\right)\ (1+1\mathrm{i}) \\ & + t\ \ln\left(-z_0\ \left(d_b + t\ (-1+1\mathrm{i})\right)\right)\ (1-\mathrm{i}) \end{aligned}$$

This results in the following slope equation:

$$\theta_y = \frac{4\,F_y\,{z_0}^2\,Q_2(z)}{E\,t^3\,\pi\left(d_b - d_t\right)^2} + C_{y1} \tag{2.15}$$

where,

$$
\begin{aligned}
Q_2(z) = {} & 2\,d_t\,\ln\left(d_b\,z - d_t\,z - d_b\,z_0 + t\,z_0\right) \\
& - d_t\,\ln\left(d_b\,z - d_t\,z - d_b\,z_0 + t\,z_0\,\left(1 - \mathrm{i}\right)\right) \\
& - d_t\,\ln\left(d_b\,z - d_t\,z - d_b\,z_0 + t\,z_0\,\left(1 + 1\mathrm{i}\right)\right) \\
& - 2\,d_t\,\ln\left(-z_0\,\left(d_b - t\right)\right) + d_t\,\ln\left(-z_0\,\left(d_b + t\,\left(-1 - \mathrm{i}\right)\right)\right) \\
& + d_t\,\ln\left(-z_0\,\left(d_b + t\,\left(-1 + 1\mathrm{i}\right)\right)\right) \\
& - 2\,t\,\ln\left(d_b\,z - d_t\,z - d_b\,z_0 + t\,z_0\right) \\
& + t\,\ln\left(d_b\,z - d_t\,z - d_b\,z_0 + t\,z_0\,\left(1 - \mathrm{i}\right)\right)\,\left(1 - \mathrm{i}\right) \\
& + t\,\ln\left(d_b\,z - d_t\,z - d_b\,z_0 + t\,z_0\,\left(1 + 1\mathrm{i}\right)\right)\,\left(1 + 1\mathrm{i}\right) \\
& + 2\,t\,\ln\left(-z_0\,\left(d_b - t\right)\right) + t\,\ln\left(-z_0\,\left(d_b + t\,\left(-1 - \mathrm{i}\right)\right)\right)\,\left(-1 - \mathrm{i}\right) \\
& + t\,\ln\left(-z_0\,\left(d_b + t\,\left(-1 + 1\mathrm{i}\right)\right)\right)\,\left(-1 + 1\mathrm{i}\right)
\end{aligned}
$$

At this point, it has become clear that an analytic solution for the spring constant is not practical. Integrating the slope equation (Equation 2.15) would produce the deflection equation; however this integral become extremely complicated. A cleaner option is to calculate the spring constants numerically.

2.3.2.2 Numerical calculation of the y-direction spring constant, K_y

To determine the spring constant numerically, a vector of z values must be created to be used in the numerical integration.

To calculate the slope of the tower, θ, the curvature equation needs to be numerically integrated:

$$\theta(z) = \int_0^{z_0} \frac{-M(z)}{E\,I(z)}\,dz \tag{2.16}$$

The applied force, F_y, in the moment equation (Equation 2.6) can be factored out because it is not a function of z. This results in the following equation for the slope:

$$\theta(z) = \frac{F_y}{E} \int_0^{z_0} \frac{z_0 - z}{I(z)}\, dz \tag{2.17}$$

The integral in Equation 2.17 can be approximated by a sum of n elements and calculated numerically.

$$\theta(z) = \frac{F_y}{E} \sum_{i=0}^{n} \frac{z_0 - z_i}{I(z_i)}\, \Delta z_i \tag{2.18}$$

Equation 2.18 is a much simpler method for calculating the beam slope compared to the analytic solution shown in Equation 2.15. The deflection of the tower, y_d, can be calculated by integrating the slope equation.

$$y_d = \int_0^{z_0} \theta(z)\, dz = \int_0^{z_0} \left(\frac{F_y}{E} \sum_{i=0}^{n} \frac{z_0 - z_i}{I(z_i)}\, \Delta z_i \right) dz \tag{2.19}$$

The above equation can be numerically approximated with a summation of m elements as follows:

$$y_d = \sum_{j=0}^{m} \left(\frac{F_y}{E} \sum_{i=0}^{n} \frac{z_0 - z_{ij}}{I(z_{ij})}\, \Delta z_i \right) \Delta z_j \tag{2.20}$$

$$y_d = \frac{F_y}{E} \sum_{j=0}^{m} \sum_{i=0}^{n} \frac{z_0 - z_{ij}}{I(z_{ij})}\, \Delta z_i \Delta z_j \tag{2.21}$$

To determine the spring constant, the deflection equation should be rewritten in the

Table 2.2: Calculated spring constant and natural frequency for the y-direction vibration.

Parameter	**Constant Top Area**	**Tapered Beam**	**Constant Bottom Area**	**FEA value**
k_y	1283 N/m	8966 N/m	16213 N/m	4233 N/m
f_y	0.34 Hz	0.84 Hz	1.1 Hz	0.58 Hz

form, $F = -kx$, which in this case is:

$$F_y = -k_y\, y_d \tag{2.22}$$

$$F_y = \frac{E}{\sum_{j=0}^{m} \sum_{i=0}^{n} \frac{z_{ij}-z_0}{I(z_{ij})}\, \Delta z_i \Delta z_j}\, y_d \tag{2.23}$$

From Equation 2.23, is can be seen that the spring constant is:

$$k_y = \frac{E}{\sum_{j=0}^{m} \sum_{i=0}^{n} \frac{z_{ij}-z_0}{I(z_{ij})}\, \Delta z_i \Delta z_j} \tag{2.24}$$

Table **??** shows the calculated spring constant values and natural frequencies in the y-direction for the turbine parameters. The constant top area value is the spring constant calculated assuming a uniform circular cross-section beam with a profile equal to the top of the tower. The constant bottom area value is the spring constant calculated assuming a uniform circular cross-section beam with a profile equal to the bottom of the tower. The FEA values are derived in Tom Gwon's analysis [11], although it is unclear whether these values apply for the y or x direction. The tapered beam calculation is performed in MATLAB using the following code (with turbine parameters listed in Table 2.3):

```
1 z = linspace(0, z_0, 1000); % [m] Create the vector of z values
```

```
2 M_F = (z - z_0); % [m] The moment equation with respect to the ...
      applied force (M/F)
3 d_F = d_b - z*(d_b-d_t)/z_0; % [m] The diameter as a function of ...
      height
4 I = pi/64*d.^4 - (pi/64)*(d-2*t).^4; % [m^4] The moment of inertia
5 v_F = -M_F ./ (E*I); % [s^2/m^2/kg] The curvature equation
6 theta_F = cumtrapz(z, v_F); % [s^2/m/kg] The slope equation
7 y_F = cumtrapz(z, theta_F); % [m/N] The displacement with respect ...
      to force
8 ky = 1 ./ y_F(end); % [N/m] Spring constant
```

The intermediate variables from the MATLAB calculation are shown in Figure 2.2. The curvature, slope, and displacement are all with respect to force, since it cancels out of the equation and is not used. The spring constant is calculated by taking the inverse of the displacement/force curve at the largest z value. These plots were created using a z-vector of 1000 elements.

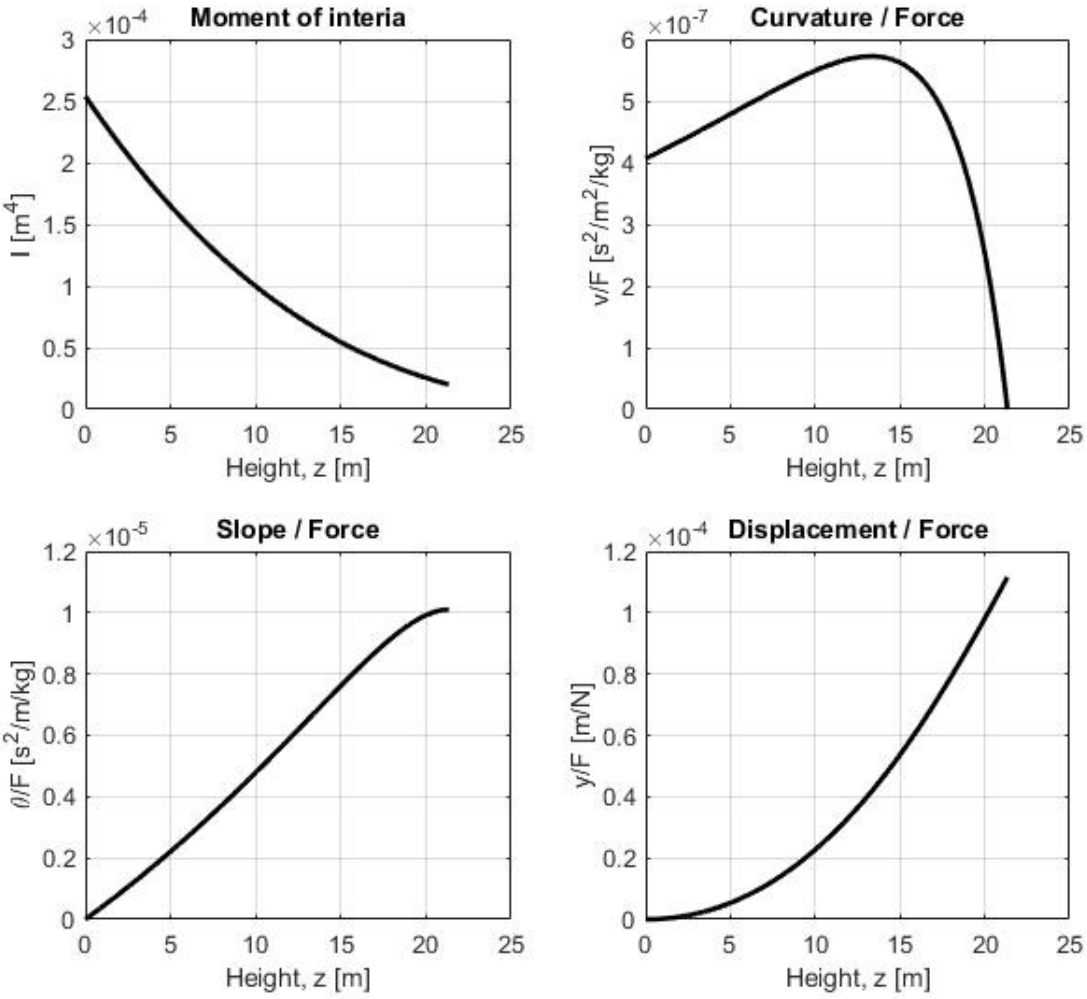

Figure 2.2: The intermediate variables from the numerical spring constant calculation in MATLAB.

Table 2.3: Turbine parameters used in the MATLAB analysis for calculating k_y

Variable	Value	Description
z_0	70 ft	The total height of the tower
d_b	20 in	The diameter of the bottom of the tower
d_t	8.7 in	The diameter of the top of the tower
E	30 Mpsi	The modulus of elasticity for ASTM A572 Grade-50 Steel
t	0.2 in	The thickness of the turbine tower tube

2.3.3 Determining Effective Mass

The cantilever beam (Figure 2.3) can be simplified to a massless beam with all of the mass concentrated at the end of the beam.

TODO: Redo this derivation assuming I is a function of x: $I(x)$

The total potential energy, P, in the beam is [16]:

$$\begin{aligned} P &= \frac{EI}{2}\int_0^L \left(\frac{d^2y}{dx^2}\right)^2 dx \\ &= \frac{1}{64}\pi^4\left[\frac{EI}{L^3}\right] y_0^2 \end{aligned} \tag{2.25}$$

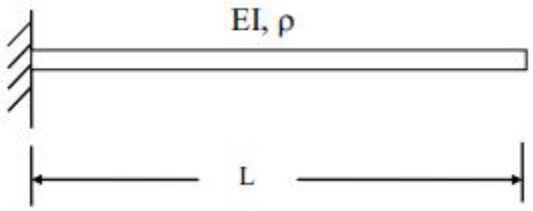

Figure 2.3: A cantilever beam model [16]

The total kinetic energy, T, in the beam is:

$$\begin{aligned} T &= \frac{1}{2}\rho_L \omega_n^2 \int_0^L y^2 dx \\ &= \frac{1}{4}\rho \omega_n^2 y_0^2 L \left(3 - \frac{8}{\pi}\right) \end{aligned} \tag{2.26}$$

Equating Equation 2.25 and Equation 2.26 results in the following equation for natural frequency f_n [Hz]:

$$f_n = \frac{\pi}{8L^2}\sqrt{\frac{EI}{\rho\left(3 - \frac{8}{\pi}\right)}} \tag{2.27}$$

The stiffness at the free end of a cantilever beam is:

$$k = \frac{3E}{L^3} I \tag{2.28}$$

The natural frequency of a beam is a function of the stiffness and effective mass:

$$\omega_n = 2\pi f_n = \sqrt{k/m_{eff}} \tag{2.29}$$

Simplifying the previous few equations results in the following equation for the effective mass at the end of a mass-less beam:

$$m_{eff} = \frac{48L}{\pi^4}\rho\left(3 - \frac{8}{\pi}\right) \tag{2.30}$$

2.3.4 Developing State Space Equations

Vibrations in the x and y directions can be split up into 2 independent 2nd order differential equations:

$$\ddot{x} + \frac{C_x}{m}\dot{x} + \frac{k_x}{m}x = \frac{1}{m}F_x(t) \tag{2.31}$$

$$\ddot{y} + \frac{C_y}{m}\dot{y} + \frac{k_y}{m}x = \frac{1}{m}F_y(t) \tag{2.32}$$

These equations can be converted to state space form:

$$\dot{\boldsymbol{x}} = \boldsymbol{A_x x} + \boldsymbol{B_x} F_x(t) \tag{2.33}$$

$$x = \boldsymbol{C_x x} + \boldsymbol{D_x} F_x(t) \tag{2.34}$$

$$\dot{\boldsymbol{y}} = \boldsymbol{A_y y} + \boldsymbol{B_y} F_y(t) \tag{2.35}$$

$$y = \boldsymbol{C_y y} + \boldsymbol{D_y} F_y(t) \tag{2.36}$$

Where the full equations are written out as follows:

$$\begin{bmatrix} \dot{x}_1 \\ \dot{x}_2 \end{bmatrix} = \begin{bmatrix} 0 & 1 \\ -\frac{k_x}{m} & -\frac{C_x}{m} \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} + \begin{bmatrix} 0 \\ \frac{1}{m} \end{bmatrix} \cdot F_x(t) \tag{2.37}$$

$$x = \begin{bmatrix} 1 & 0 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} + 0 \cdot F_x(t) \tag{2.38}$$

$$\begin{bmatrix} \dot{y}_1 \\ \dot{y}_2 \end{bmatrix} = \begin{bmatrix} 0 & 1 \\ -\frac{k_y}{m} & -\frac{C_y}{m} \end{bmatrix} \begin{bmatrix} y_1 \\ y_2 \end{bmatrix} + \begin{bmatrix} 0 \\ \frac{1}{m} \end{bmatrix} \cdot F_y(t) \tag{2.39}$$

$$y = \begin{bmatrix} 1 & 0 \end{bmatrix} \begin{bmatrix} y_1 \\ y_2 \end{bmatrix} + 0 \cdot F_y(t) \tag{2.40}$$

These equations are for a simple cantilever model with a lumped mass at the end of the beam (lumped mass accounts for nacelle and tower mass). The resulting natural frequency from this model matches the ABAQUS FEA results from Gwon's paper [11]. A comparison between the resulting natural frequency from this model and the ABAQUS FEA results in Gwon's analysis [11] can be seen in Table 2.2.

Chapter 3

DIGITAL SIGNAL PROCESSING MODEL

3.1 Model Block Diagram

The blade imbalance detection is simulated using MATLAB/Simulink. Figure 3.1 shows the block diagram for a method of imbalance detection where the rotor frequency is known. This is an accurate method, but requires the information from an encoder on the shaft or an accurate phase-locked loop (PLL) controller.

The force due to an imbalance is given in Equation 2.1. It is a periodic function with the same frequency as the rotor and is proportional to the effective imbalance mass, m. The variable forcing function is the input for the lumped parameter turbine tower state space model. These vibrations are then read by an acceleration sensor with a sampling rate of 100 Hz. The frequency of interest is demodulated using a lock-in amplifier or DFT, which should result in the acceleration from only the imbalance. The acceleration at the top of the tower should have higher intensity vibrations at the rotor frequency when there is an imbalance in the blades (causing an eccentric mass).

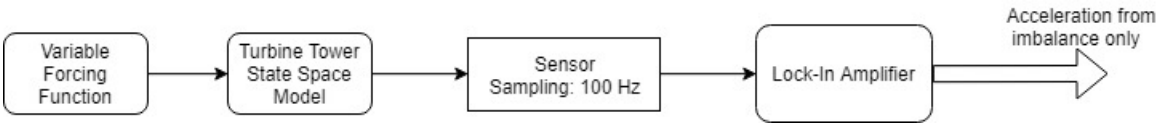

Figure 3.1: A block diagram showing the lock-in amplifier and the turbine tower simulation.

3.2 Processing the data

This paper describes a few ways to process the acceleration data from the turbine tower. One method relies on accurate frequency information to precisely demodulate the signal at the rotor frequency. An alternative method is to lock onto the strong frequency vibration of the tower using a phased-lock loop controller. Finally, a discrete Fourier transform can be calculated to transform the time domain data to the frequency domain. There are various ways to make this process more efficient, including zoom FFTs and the Goertzel algorithm.

The simulation and turbine tower state space model produce displacements shown in Figure 3.2. The same simulation data is shown in the frequency domain in Figure 3.3. From the frequency domain, it can be seen that there is resonance at 0.84 Hz (the natural frequency of the tower in the x-direction). Additionally, there is an excitation at the rotor frequency that has a magnitude proportional to the amount of mass at the end of the blade. This simplified linear model produces an displacement signal that can be represented by the sum of sines as shown in Equation 3.1 (this can be seen in Figure 3.2).

$$y = C_1 \sin(\omega_n t + \phi_1) + C_2 \sin(\omega_{drive} t + \phi_2) \tag{3.1}$$

The first method for processing the acceleration data utilizes lock-in amplifier to translate the data in the frequency domain. This new data, with the rotor frequency shifted to 0 Hz, is filtered and results in a DC acceleration that should be proportional to the imbalance mass. The second method utilizes a zoom FFT, which provides higher resolution than a typical FFT in the bandwidth of interest. The third method uses a Goertzel algorithm to calculate the bin of interest in the DFT.

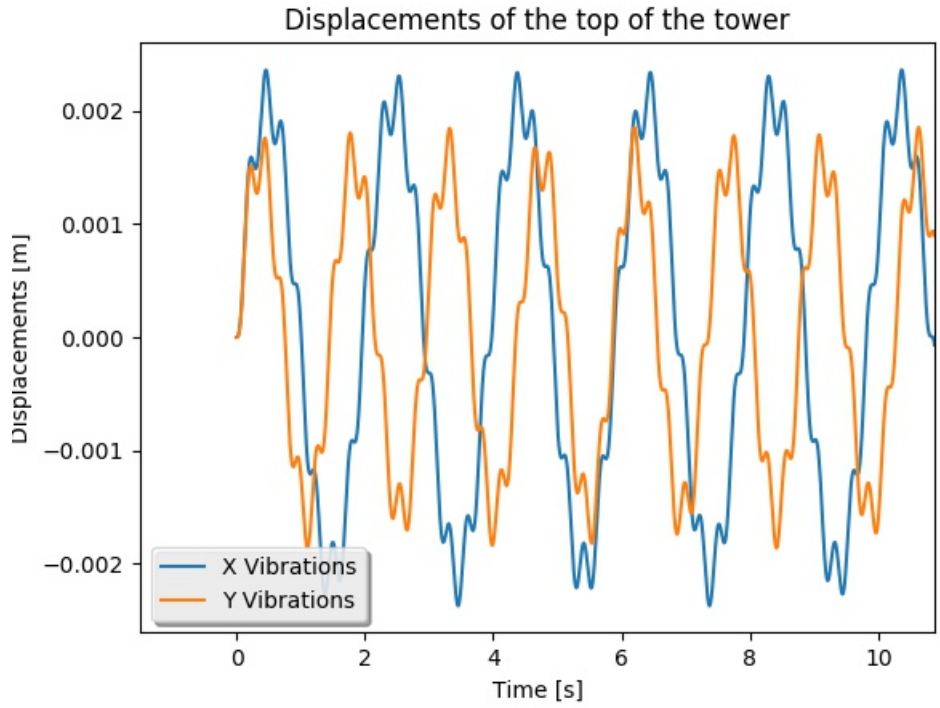

Figure 3.2: The simulated displacement for a rotor imbalance at 230 RPM. The imbalance mass is placed at the end of the blade, which causes accelerations and displacements at the top of the tower.

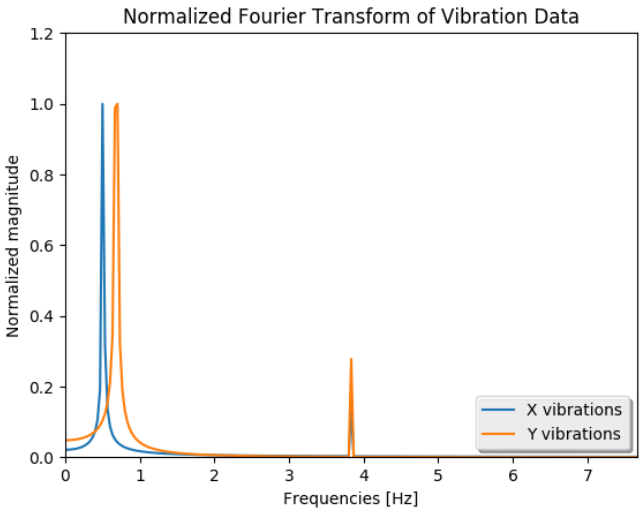

Figure 3.3: This figure shows the simulated displacement data in the frequency domain. The rotor frequency is 230 RPM in this simulation

3.2.1 DFT review

A Discrete Fourier Transform (DFT) converts a signal from the time domain to the frequency domain. Typically, this is performed with using the Fast Fourier Transform (FFT) algorithm, which is an efficient method for calculating the DFT. The DFT is defined by the formula in Equation 3.2 [**?**].

$$X_k = \sum_{n=0}^{N-1} x_n \cdot e^{-\frac{i2\pi}{N}kn} \tag{3.2}$$

$\vec{x}$ is the original signal in the time domain with N elements, and $\vec{X}$ is the signal in the frequency domain with N elements. Equation 3.2 can be rewritten using Euler's formula to show DFT equation in terms of sine and cosine values (this is important for understanding the lock-in amplifier in a future section). Equation 3.3 shows the new expression for the DFT equation.

$$X_k = \sum_{n=0}^{N-1} x_n \cdot \left[\cos\left(\frac{2\pi kn}{N}\right) - j\sin\left(\frac{2\pi kn}{N}\right)\right] \tag{3.3}$$

From Equation 3.3, it can be seen that the DFT can be calculated by mixing the signal, $\vec{x}$, with a sine and cosine signal and summing the result for each frequency. The DFT equation can be further rewritten using vector notation as in Equation 3.7 and defining some reference vectors. This equation shows that the DFT of a signal is an imaginary vector that can be created from the dot products of the original signal and some reference sine and cosine signals.

$$\vec{n}_r = \begin{bmatrix} 0/N \\ 1/N \\ 2/N \\ \vdots \\ (N-1)/N \end{bmatrix} \tag{3.4}$$

$$\vec{r}_{sin} = \sin\left(2\pi k \vec{n}_r\right) \tag{3.5}$$

$$\vec{r}_{cos} = \cos\left(2\pi k \vec{n}_r\right) \tag{3.6}$$

$$X_k = \vec{x}^T \cdot \vec{r}_{cos} - j\vec{x}^T \cdot \vec{r}_{sin} \tag{3.7}$$

Another way of looking at the DFT (from the form in Equation 3.7) is as a time domain convolution with a low pass filter. The summation (shown explicitly in the form from Equation 3.3) acts as a simple finite impulse response (FIR) decimation filter with unity coefficients and a decimation rate of $d = N$, commonly known as an averaging filter. When thinking about the DFT in this way, some possible optimization methods start to become clear. For example, we may not need to calculate the frequency component for every kth element in $\vec{X}$, and we may choose to use a better filter that has stronger attenuation at higher frequencies. This is the basis for the lock-in amplifier design described in the next section.

3.2.2 Lock-In Amplifier

A lock-in amplifier is used to measure AC signals in particularly noisy environments (Figure 3.4). It works by multiplying the noisy signal by a reference signal created by an internal oscillator. By using 2 reference signals out of phase, the real and imaginary parts of the signal at the specified carrier frequency can be calculated[3]. Let's say there is an unknown sinusoidal signal with high frequency noise ($\omega_2 >> \omega_1$):

$$y = A \sin(\omega_1 t) + B \sin(\omega_2 t) \tag{3.8}$$

The amplitude of the signal, A, can be determined by multiplying the signal, y, by a reference signal with the same frequency as ω_1 and filtering:

$$y \cdot \sin(\omega_1 t) = [A \sin(\omega_1 t) + B \sin(\omega_2 t)] \sin(\omega_1 t) \tag{3.9}$$

$$= A \sin^2(\omega_1 t) + B \sin(\omega_1 t) \sin(\omega_2 t) \tag{3.10}$$

$$= A\left(\frac{1}{2} - \frac{1}{2}\cos(2\omega_1 t)\right) + B \sin(\omega_1 t) \sin(\omega_2 t) \tag{3.11}$$

Note that Equation 3.9 resembles the imaginary part of the standard DFT equation (Equation 3.3), where the input signal is multiplied by a sine wave at a set reference frequency.

A filter is applied to Equation 3.11 (similar to the summation filter used in Equation 3.3), with a cutoff frequency that is smaller than ω_1. This results in all terms approaching zero, except for $\frac{A}{2}$, which can be calculated and solved for A to achieve the magnitude of the frequency component at ω_1. To calculate the magnitude and phase of the signal y, it must be multiplied by a reference sine signal (as in Equation 3.9) and a reference cosine signal. The following equation shows the result of the mixing step (phase-sensitive detection) in the lock-in amplifier:

$$Y = X \cos(\omega t) + jX \sin(\omega t) \tag{3.12}$$

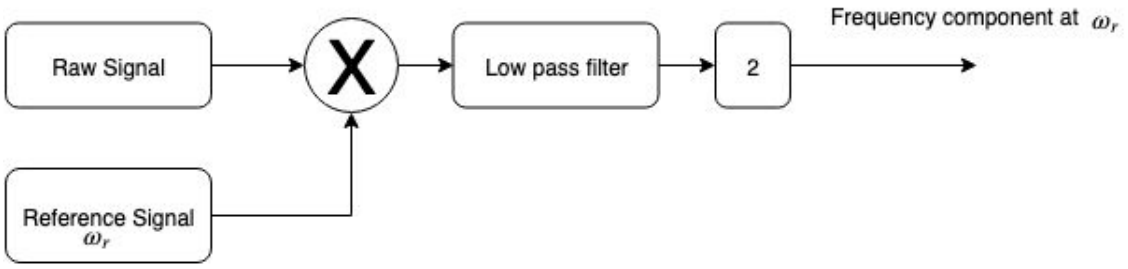

Figure 3.4: A block diagram showing the details of a lock-in amplifier [2]. This figure was created with Draw.io.

where X is the noisy AC signal and Y is the signal that is fed into a filter to remove any unwanted frequency components at ω_1 and $2\omega_1$ and produce a signal, $Y_{filtered}$ that only contains the DC component:

$$Y_{filtered} = \frac{A_{real}}{2} + j\frac{A_{imaginary}}{2} \tag{3.13}$$

Using the lock in amplifier tuned to a constant frequency, Figure 3.5 shows the resulting output. This shows that an imbalance can be detected by measuring the output of a lock-in amplifier that is tuned to the frequency of the turbine rotor. When the acceleration data is treated as an AM signal, with the carrier frequency equal to the rotor frequency, the acceleration due to the rotation of the blades can be identified. This acceleration is directly related to the mass at the end of the blade, and should be a constant value (as long as the rotor frequency is accurately known).

Lock-in amplifiers are typically used when the frequency is relatively stable and the signal-to-noise ratio is low. For this application, the rotor frequency can change, which may not give the LIA filters enough time to settle. If there is a very stable and accurate rotor speed controller, this method would be ideal because it extracts the signal of interest despite the high noise and distortions in the measurement.

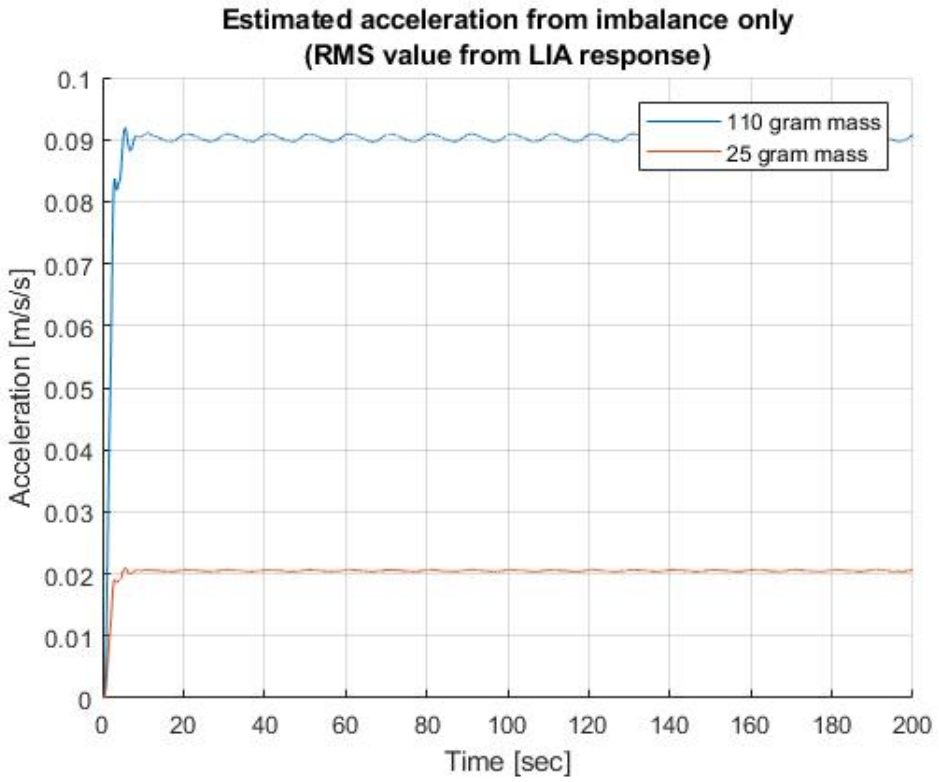

Figure 3.5: The output of the lock-in amplifier from the simulated data shown in Figure 3.2

3.2.3 Identifying the rotor frequency from the tower accelerations

If the rotor frequency cannot be measured directly it is possible to obtain the frequency indirectly from the acceleration data. A phase-locked loop is a control system that minimizes the phase between to signals. As the phase error is driven to zero, the output signal frequency approaches the input signal's frequency. Phase-locked loops (PLLs) are commonly used in telecommunications where the frequency of a noisy signal can be recreated with an internal oscillator that locks on to the frequency of the noisy signal. Figure 3.6 shows a block diagram of the PLL implementation with tower acceleration data as the input.

The first stage of the frequency detection is a narrow bandpass filter to get rid of high frequency noise and natural frequency resonance of the tower. This is multiplied by an internal oscillator that starts at an arbitrary frequency (a guess that is close to the actual frequency, but not necessarily accurate). This signal is passed through a lowpass filter to provide a steady phase error. The phase error is driven to 0 using a PID controller. This outputs the correct frequency of the signal that is then fed back and multiplied with the input signal.

One of the problems with a phase-locked loop is the PID controller gains depend on the amplitude of the signal, which changes depending on the amount of imbalance and rotor frequency. The simplest approach to this would be to normalize the accelerations with the expected frequency. This, however, is not perfect and doesn't account for the acceleration variation due to different eccentric masses. A better solution would be to use a simple envelope detector to determine the amplitude of signal. The accelerations can be divided by the output of the envelope detector to normalize the signal. Figure 3.8 shows an example of a simple envelope detector[4]. The low-pass filter in the envelope detector can be designed based on the desired accuracy. For

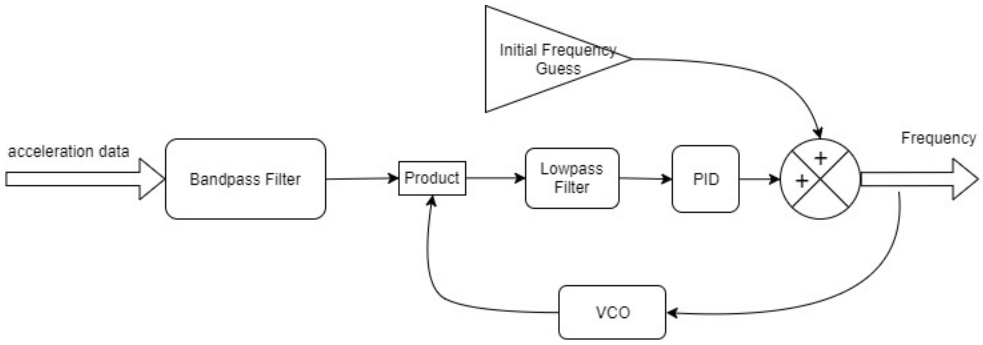

Figure 3.6: The block diagram of a phase-locked loop implementation when detecting rotor frequencies from acceleration measurements [14]. This diagram was created with Draw.io [?].

high computational efficiency, an averaging filter can be used because it minimizes the amount of multiply operations, and reduces the filter calculation to summations only.

3.2.4 Zoom FFT

A very powerful method for reducing the computational cost of the FFT calculation is to use a Zoom FFT. If the imbalance detection device contains a low-power processor, it will be able to run for a long time off of a battery. This also means that the processor could be much slower than an alternative high-power processor. A Zoom FFT will significantly reduce the computational load on the processor, allowing the low-power processor to sufficiently calculate the frequency spectrum of the input signal.

One of the main tradeoffs for the reduced computational load, is a reduced frequency spectrum range. For example, a set of 1024 data points sampled at F_s, will produce a frequency spectrum from 0 to $F_s/2$ using the standard FFT. A zoom FFT with a decimation rate of $D = 10$ can produce a frequency spectrum with a range of $F_s/20$ at 1/10th the computational cost.

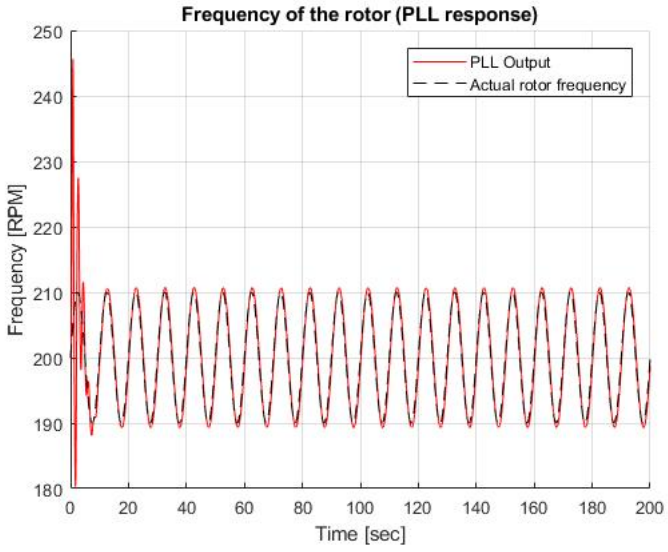

Figure 3.7: The output of the PLL when detecting frequency from the data shown in Figure 3.2. The frequency of the input data oscillates from 190 RPM to 210 RPM to simulate a real turbine controller that may oscillate between frequencies.

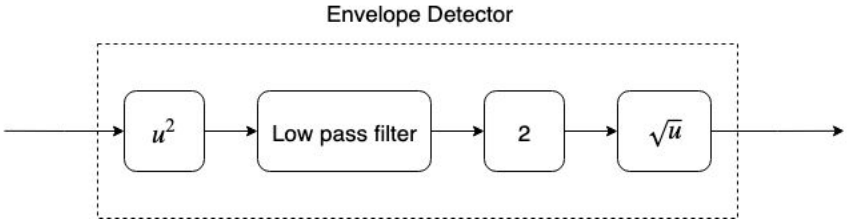

Figure 3.8: A block diagram for a simple envelope detector. The signal is first squared and multiplied by a gain of 2. This signal (which should be entirely positive) is sent through a lowpass filter to remove the high frequency information. The lowpass filter needs to reject $2f$, which is the resulting frequency when a signal with frequency, f is squared. The square root of the resulting signal produces the the amplitude of the signal. This figure was created with Draw.io.

If the frequency of the rotor is not precisely known, then a lock-in amplifier is difficult to implement. In order to detect excitation at the rotor frequency, the time-domain data can be transformed into the frequency domain. A discrete Fourier transform (DFT) is the frequency domain representation of a sampled signal; however, can be slow and difficult to calculate on a microcontroller.

To efficiently calculate the Fourier transform, the DFT matrix can be factored and the complexity reduced from $O(n^2)$ to $O(n \log n)$. This efficient algorithm is called the fast Fourier transform (FFT). Despite being much more efficient than the standard DFT, a generic FFT is still inefficient for this application because most of the frequency data occurs outside the bandwidth of interest and is thrown away. This can be fixed by applying a zoom FFT algorithm.

The zoom FFT consists of 4 main steps. First the data is convoluted with a reference signal in the time domain (translated in the frequency domain) to shift a center frequency, F_c, to 0 Hz. Second, a lowpass filter prevents aliasing when sampling at a lower sample rate. An infinite impulse response (IIR) filter can be used for simplicity; however, the phase information of the signal is lost. To solve this, a finite impulse response (FIR) filter can be used. FIR filters are more complicated but most have a flat phase response, ensuring the filter output retains the original phase information. Typically, discrete FIR filters are applied in various 2-stage filters for simplicity. Third, the data is decimated (re-sampled at a lower rate). Finally, the decimated data is passed through a FFT algorithm that produces a frequency spectrum in the specified bandwidth. Since the frequency range of the zoom FFT is much smaller, the resolution can be much higher for a FFT length. Alternatively, a much smaller FFT length can be used to produce the same resolution.

For a microcontroller sampling the acceleration of the turbine tower at 100 Hz, Figure 3.9 shows a block diagram of a zoom FFT implementation for detecting an imbalance.

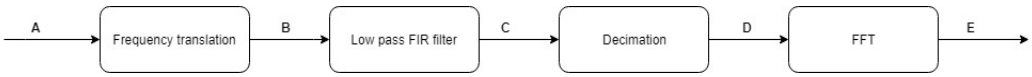

Figure 3.9: A block diagram showing the zoom FFT algorithm. This figure was created with Draw.io.

Point A represents the raw input signal, which is the acceleration of the tower sampled at $F_s = 100$ Hz in this case. The frequency translation is performed by mixing the input signal with a reference signal at F_c. The goal of a Zoom FFT is to reduce the amount of data by decimating the original signal and reducing the sampling rate. Just performing a sampling rate reduction would result in aliasing, so there needs to be a digital anti-aliasing filter that prepares the signal for the decimation step. This digital filter is a low pass FIR filter that produces a filtered signal at Point C, which can then be decimated without aliasing to achieve Point D. This new set of data is sampled at $F_{s,new} = \frac{F_s}{D}$, which means the frequency spectrum spans a much smaller range.

Zoom FFTs [8] contain the following parameters:

F_c: The frequency center (shifted to 0 Hz by the zoom FFT)

BW: The bandwidth of interest

F_s: The sampling rate

N: The length of the FFT

The decimation factor, D, is calculated as follows:

$$D = floor\left(\frac{F_s}{BW}\right)$$

Where the length, L, of the initial buffer into the decimator is:

$$L = D \cdot N$$

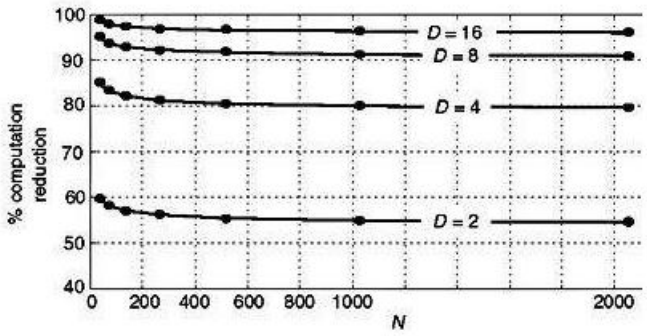

Figure 3.10: A diagram showing the percent computational workload reduction of a $\frac{N}{D}$-point Zoom FFT relative to a standard n-points FFT [14].

The computational reduction for a Zoom FFT is shown in Figure 3.10. However, it is important to note that a stronger filter is required for higher decimation factors, D. This means there is a trade-off between FFT computational load and filter computational load. A zoom FFT is just the name of a method of frequency transformation where the data is decimated before applying the FFT algorithm. The computational savings occur when a well-designed lowpass filter does not negate the workload reduction caused by the FFT. Essentially, this is a much more flexible FFT where different aspects can be modified to optimize for the application. In this case, the Zoom FFT can provide a much higher resolution frequency analysis because the frequencies of interest are much smaller than the sampling rate.

3.2.4.1 Frequency translation

Shifting data in the frequency domain is a common technique used in AM radio signals. Audio frequencies are typically mixed with a high frequency carrier that, according to the Nyquist criteria, would require an extremely high sample rate to capture the signal.

If ω is the rotor frequency, multiplying the time domain signal by $\cos(-\omega t)$ is the

same as convolving the data in the frequency domain. Since the frequency spectra for $\cos(-\omega t)$ is symmetric across the y-axis, the time-domain must be multiplied by an imaginary wave.

The equation for the frequency-shifted time-domain data is:

$$Y = Xe^{-i(\omega t)} \tag{3.14}$$

where Y is the shifted data, X is the original data, ω is the rotor frequency, and t is the time. Based on Equation 3.14, the real and imaginary parts of the shifted signal are shown below [8]:

$$Y_{real} = X\cos(\omega t) \tag{3.15}$$

$$Y_{imaginary} = -X\sin(\omega t) \tag{3.16}$$

An example of frequency translation is shown in Figure 3.11. The blue signal is the frequency-shifted signal, which has a higher frequency at 2 times the original signal frequency. In order to remove the extra component at 10 Hz, the digital filter (shown with a dotted blue line) must be applied to the shifted signal.

3.2.4.2 Decimation/downsampling

Decimation is the process of reducing the sample rate of a signal. Before the data is downsampled, the high frequency components are reduced with a lowpass filter. This filter is also called an anti-aliasing filter because it prevents high frequency data from being misinterpreted at a different sample rate. Equation 3.17 shows the equation used for calculating the output of a FIR decimator, where x is the input data, h is the impulse response, K is the length, and D is the decimation factor[13].

$$y[n] = \sum_{k=0}^{K-1} x[nD-k]\cdot h[k] \tag{3.17}$$

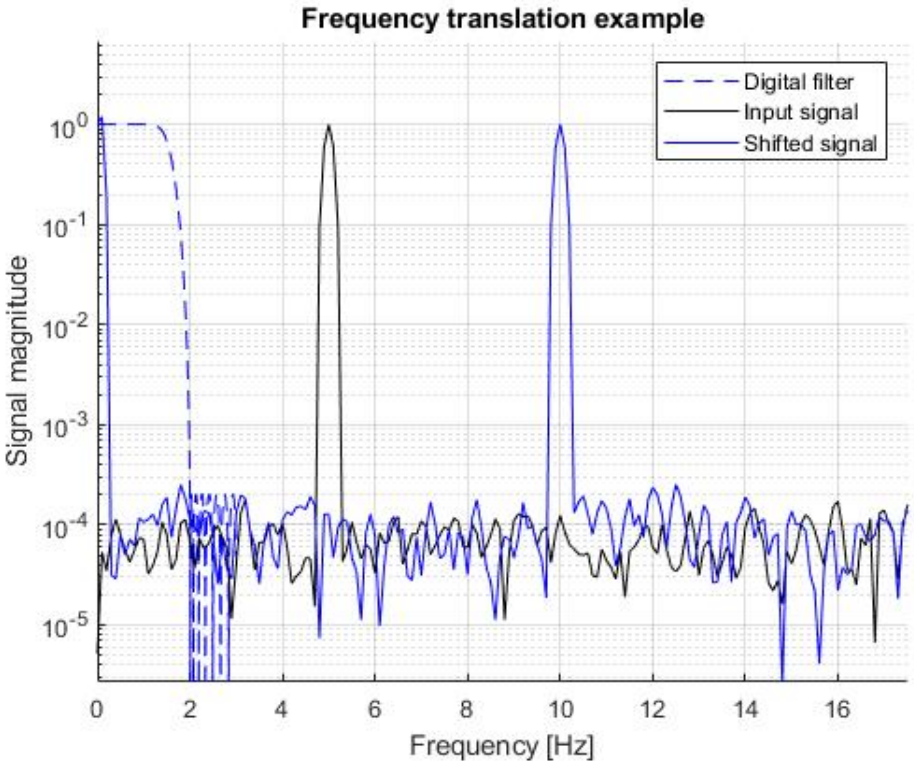

Figure 3.11: This figure shows an example of frequency translation. The black signal is the frequency spectrum of some input signal that has a strong component at 5 Hz. The blue signal shows the frequency shifted spectrum that results from multiplying the input signal by a reference signal with a frequency of 5 Hz (as shown in Equation 3.16. The dotted blue line shows the digital filter response. This figure was generated with MATLAB.

Table 3.1: Zoom FFT parameters

Variable	Value	Description
F_{min}	2 Hz	The minimum frequency for the Zoom FFT calculation
F_s	128 Hz	The original sampling rate
F_r	4 Hz	The frequency range of the Zoom FFT
N_{fft}	128	FFT length (buffer length used in the FFT calculation)
N_b	255	FIR filter order
D	$\frac{F_s}{2F_r} = 16$	Decimation rate

Figure 3.11 shows a visual representation of the decimation process and anti-aliasing filter. The shifted signal (blue) contains important frequency information in the range 0 - 2 Hz. Any higher frequencies have been significantly attenuated by the digital filter (dotted blue line). This means the blue data can be decimated to a new sampling rate of 4 Hz, without any aliasing effects (because the digital filter ensures there are no strong components above 2 Hz).

3.2.4.3 An optimized Zoom FFT implementation

Table 3.1 shows the parameters for the Zoom FFT. These values have been strategically picked to make the optimization process easier. In general, some nice optimizations can be achieved if the number of filter coefficients $(N_b + 1)$ is an integer multiple of the sampling rate (F_s).

In Table 3.1, the FFT length is the number of samples in the buffer used as the input to the FFT. An FFT length of 1024 means that 1024 samples are used as an input to the FFT algorithm. The FIR filter order is 255, which is high because it needs to strongly attenuate higher frequencies with a narrow pass band. Despite having

an extremely large order, the FIR filter is simple and computationally inexpensive to implement (this will be shown later in this section). An order of $N_b = 255$ means the filter order will have 256 $(N_B + 1)$ coefficients. F_{min} and F_r define the frequency range that the Zoom FFT is targeting. More specifically, the Zoom FFT will calculate the frequency spectrum of the input signal only from F_{min} to F_r. The filter design is highly motivated by the frequency range because it needs to attenuate the original signal sufficiently (to prevent aliasing at the decimation step) at the edge of the frequency range.

The reference signal for the Zoom FFT is created at the minimum frequency, F_{min} shown in Table 3.1 and is shown in Equation 3.18.

$$y_{ref} = \sin\left(F_{min} \cdot 2\pi \cdot time\right) \tag{3.18}$$

Note that there are only 64 unique values in a reference signal with the specified F_{min}. This means that all of these values can be precomputed at initialization and stored in memory. This removes the $\sin()$ operations from the real-time calculations, which reduces the computational time, since trig functions are one of the slowest floating-point operations (they are typically about 15X slower than a floating-point addition operation) [**?**].

One of the benefits for using a finite impulse response (FIR) filter is that there are no recursive calculations. This produces a very important property of FIR filters, which is that not all the outputs need to be calculated. Assuming there are $(N_b + 1)$ filter coefficients, a random window of $(N_b + 1)$ samples can be properly filtered at any time during operation. If we are decimating the input signal, this means we do not have to calculate the filter outputs for samples that are removed due to the decimation step.

The down-sampled data, y_{ds}, can be calculated as follows ($\odot$ is the element-wise

multiplication operator):

$$y_{ds} = \vec{y}^T \cdot \left(\vec{y}_{ref} \odot 2\vec{b} \right) \tag{3.19}$$

In Equation 3.19, y_{ds} is the value at the decimated sampling rate. $\vec{y}$ is a sliding buffer of the original input data, which is a vector of $(N_b + 1)$ values. $\vec{y}_{ref}$ is the vector of the reference signal values, which also has a length of $(N_b + 1)$. $\vec{b}$ is the vector of FIR filter coefficients. The factor of 2 in this equation is a result of the derivation of the frequency-shift process. Equation 3.19 is calculated at the decimated sampling rate (F_{sD}), which is shown in Equation 3.20.

$$F_{sD} = \frac{F_s}{D} \tag{3.20}$$

Equation 3.19 can also be used with larger filter orders. Effectively, this means that the filtering and decimation part of the Zoom FFT can be performed with just a single dot product and an element-wise vector multiplication. Using the values in Table 3.1, the following steps can be followed to create a Zoom FFT algorithm:

1. Precompute the unique values of the reference signal during initialization.
2. Design a FIR filter with an appropriate cutoff frequency and filter order. The cutoff frequency (F_{cutoff}) is equal to the frequency range (F_r) of the Zoom FFT. The coefficients of this filter are stored in $\vec{b}$.
3. Generate a sliding buffer of data with a window size of $(N_b + 1)$ and an overlap of $(N_b + 1 - \frac{F_s}{D})$, which is stored in $\vec{y}$.
4. Perform the calculation shown in Equation 3.19.
5. Calculate the FFT of the newly decimated data.

Figure 3.12 shows the results of a MATLAB simulation using the Zoom FFT to calculate the frequency spectrum. In this simulation, the standard FFT uses an

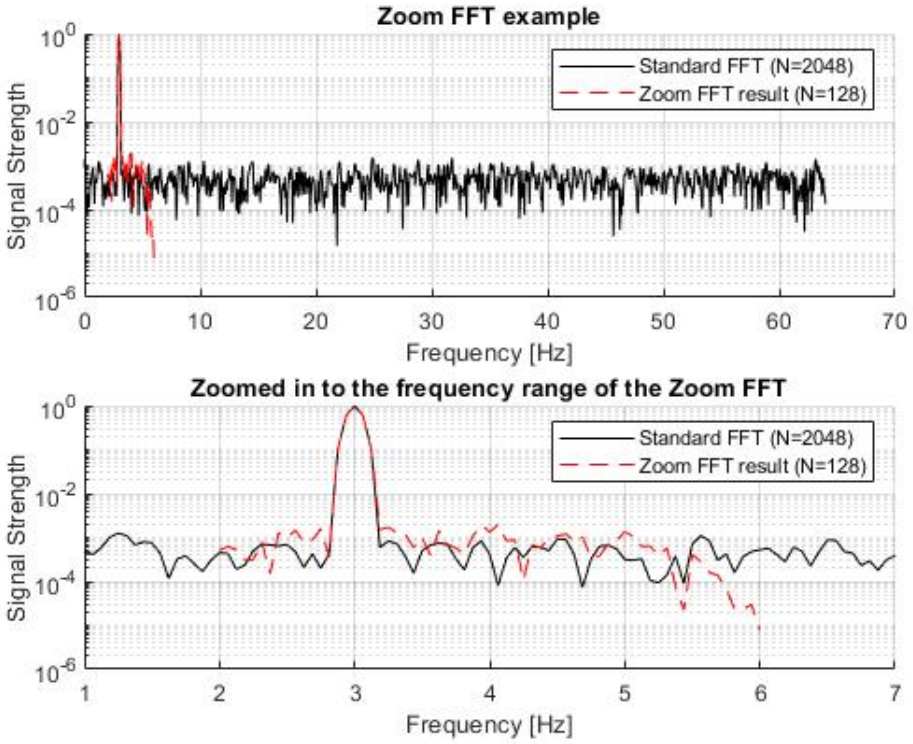

Figure 3.12: This figure shows the results of a Zoom FFT using the parameters listed in Table 3.1. The top plot shows the frequency spectrum from a standard FFT (black) and a Zoom FFT (red). The bottom plot shows the same data as the top plot, but zoomed into a narrow frequency range. The standard FFT has a length of 2048, while the Zoom FFT has a length of 128. This figure was created with MATLAB.

FFT length of 2048, while the Zoom FFT only has a length of $N_{fft} = 128$. This is a significant reduction in computation for the same frequency resolution in the target Zoom FFT range. Both a standard FFT and a Zoom FFT use the same FFT algorithm, so the difference in performance comes from the lower sampling rate, and therefore smaller frequency range.

3.2.4.4 Estimating processor performance

Comparing the computational time of different algorithms is difficult because the dominant factor is usually memory access, rather than actual floating-point operations [**?**]. Evaluating the computational cost due to memory access patterns in complicated, so this analysis will only consider the computational cost of the floating-point operations (FLOPs). To minimize the cost from memory access patterns, the algorithms will be vectorized and converted to matrix form. Most processors have highly optimized matrix operation functions, which should hopefully make arithmetic the dominant factor when determining the computational cost.

For this analysis, a STM32 processor is used as the baseline for estimating cycle counts because it is the unit used to prototype the LifeLine data acquisition device that collects experimental acceleration data from the turbine. The STM32 has its own single precision register set to handle floating-point operations with hardware (rather than using compiled C libraries to perform the floating-point operations in many complex steps). The floating-point unit (FPU) in the STM32 offers arithmetic instructions for the operations listed in **TODO: Table XXX** [**?**].

Many FPUs (like the one in **TODO: Table XXX**), have multiply/accumulate arithmetic instructions, which means they can perform a multiplication and addition all in one step. Additionally, division operations are much more expensive than multiplication operations and should be avoided as much as possible when designing algorithms.

Table 3.2: Cycle counts for a STM32 floating-point processor

Operation	Cycles
Absolute value	1
Negate of a float	1
Addition/subtraction	1
Multiply	3
Multiply and accumulate/subtract	3
Divide	14
Square Root	14
Exponential	20*

3.2.5 Goertzel Algorithm

The Goertzel algorithm is a method for calculating individual bins of a discrete Fourier transform (DFT) without calculating all of the bins. Since imbalance detection depends on the strength of the rotor frequency, this algorithm produces the particular rotor frequency bin value without calculating the entire DFT. Fundamentally, this algorithm operates similarly to the Zoom FFT previously described, and is often called a Goertzel filter because the algorithm takes the form of a digital filter. This algorithm applies 2 stages of filtering, where the first stage is an infinite impulse response (IIR) filter, and the second stage is a finite impulse response (FIR) filter.

For the first stage of the Goertzel algorithm, a 2nd order IIR filter is applied to x:

$$s[n] = x[n] + 2\cos(\omega_0)s[n-1] - s[n-2] \tag{3.21}$$

This IIR filter follows a Direct Form II structure, where 2 state variables are required for a 2nd order filter. ω_0 is defined as the normalized frequency to be analyzed.

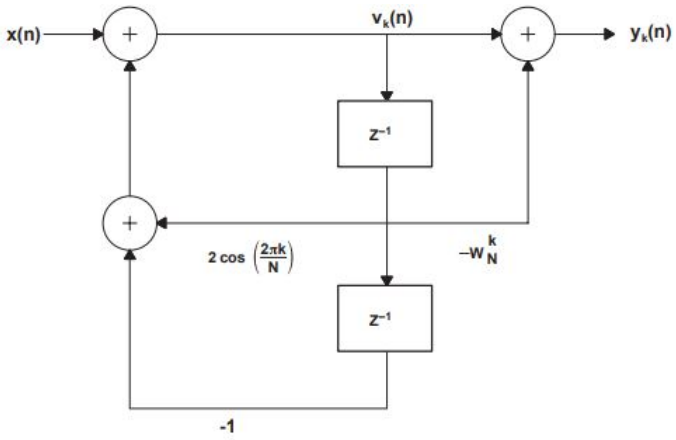

Figure 3.13: The direct-form realization of the Goertzel Algorithm[12].

The second stage of the algorithm is a FIR filter, which is known for its linear phase, and lack of feedback. This means that the filter output does not depends on any previous outputs.

$$y[n] = s[n] - e^{-j\omega_0} s[n-1] \tag{3.22}$$

Equation 3.22 shows the output of the Goertzel equation, $y[n]$.

There is an optimized version of the Goertzel algorithm that is even faster than the general form, but it produces squared magnitude components at each frequency instead of real/imaginary pairs [7]. The phase of the frequency component is not used in the imbalance detection algorithm, so the optimized Goertzel algorithm is well suited for this application. One of the benefits of the Goertzel algorithm is the buffer lengths do not need to be a power of 2. An overview of the algorithm in a block diagram form is shown in Figure 3.13.

The Goertzel algorithm is simulated with MATLAB and the results are shown in Figure 3.14. The Goertzel algorithm calculates individual frequency components of the DFT, which are identical to the values calculated with a standard FFT algorithm.

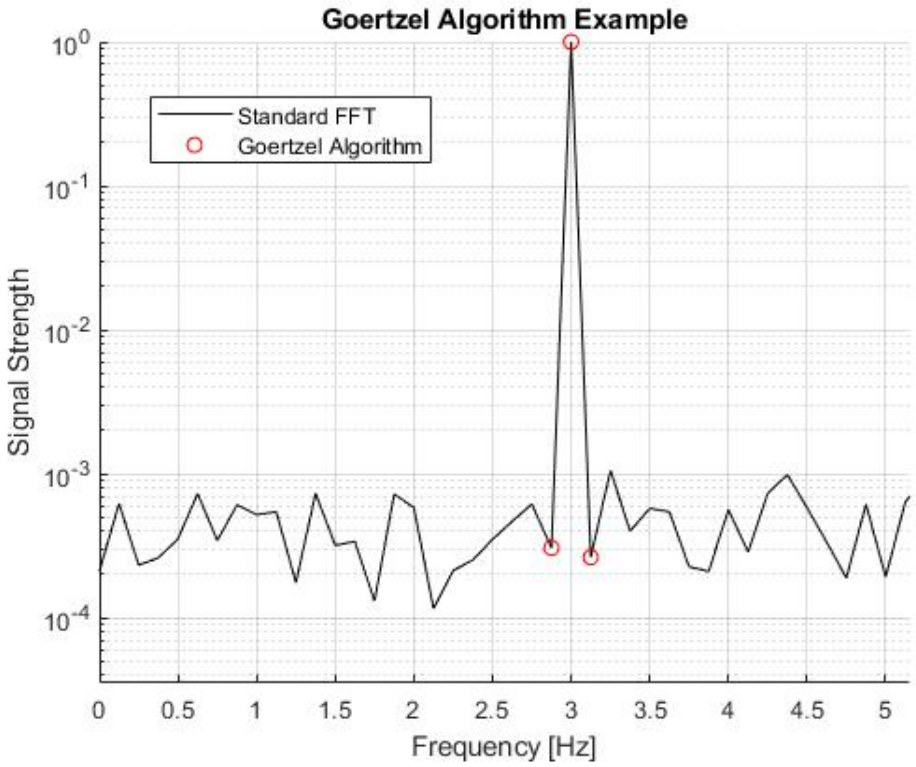

Figure 3.14: This figure shows the frequency spectrum calculated with a standard FFT (black) and a few points calculated with the Goertzel algorithm (red). This figure was created in MATLAB.

Chapter 4

IMBALANCE DETECTION ALGORITHM

4.1 Filtering the data

4.1.1 Obtaining raw data

During turbine operation, a real-time DFT calculation is performed on acceleration data. This frequency spectrum data is calculated using the LifeLine system and saved in a text file that is parsed and pre-processed using the `LifeLine_file` class. A 256-point DFT is calculated at a sampling rate of about 13 Hz and printed, along with the current time-stamp, to the text file.

4.1.2 Pre-processing the data

The raw data is converted to a series of $[frequency, magnitude]$ points at the maximum peak for each FFT. For example, in Figure 4.1, the extracted point would be [2.4 Hz, 1.0 g-sec], which is the maximum frequency component of that particular DFT. Figure 4.2 shows all of the frequency data for a single LifeLine file of a balanced turbine in a spectrogram for easy visual inspection.

The maximum frequency component and magnitude are plotted over time for the balanced rotor and shown in Figure 4.3. This data is very noisy and should be smoothed before applying any detection algorithm.

TODO: more explanation needed for the plots in this section.

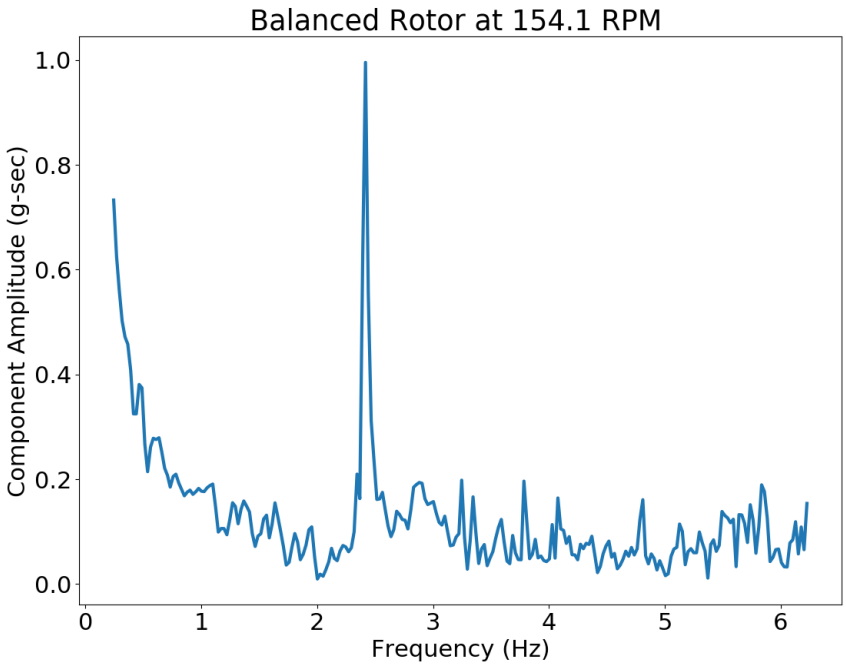

Figure 4.1: An example of a single DFT dataset from the LifeLine frequency data during a balanced rotor test.

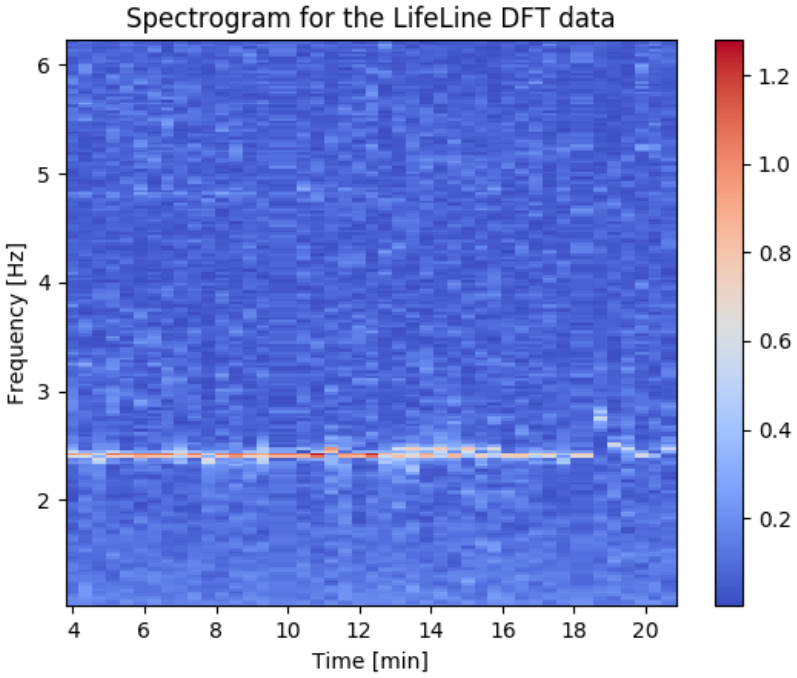

Figure 4.2: A spectrogram showing the tower frequency domain data over the length of a single LifeLine file.

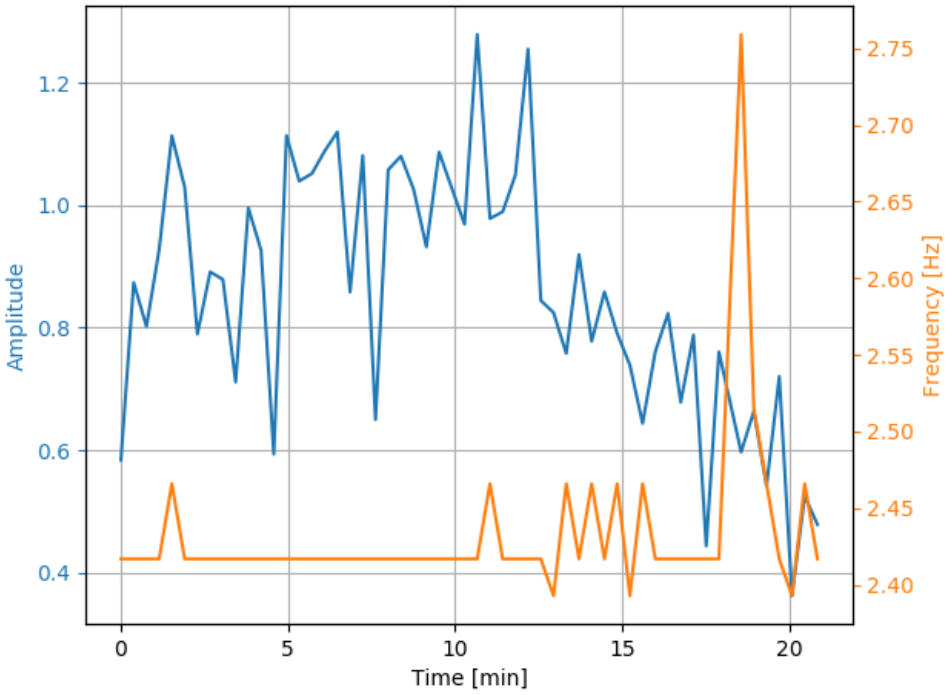

Figure 4.3: A plot of the raw maximum frequency/magnitude data of each DFT over time for a balanced rotor. This data consists of a single point extracted from each DFT and plotted over time.

4.1.3 Smoothing the data

4.1.3.1 Choosing the filter

The filter choice is very important for noise reduction in a dataset. Some of the common options are finite impulse response (FIR) filters, and infinite impulse response (IIR) filters, and the Savitzky-Golay (SG) filter.

The IIR filter is shown in Figure 4.5, which significantly smooths the data, but at the cost of some high frequency distortion. An IIR filter uses past states in the output calculation, meaning an impulse input signal will affect the filter output forever (*infinite* impulse response filter). This results in a transfer function as follows:

$$T(z) = \frac{b(z)}{a(z)} \tag{4.1}$$

Another filter alternative is the FIR filter, which does not rely on past states to calculate the filter output. This results in a finite response to an impulse input and a linear phase. A common FIR filter that is used for smoothing data is a moving average (where all of the filter coefficients are identical). To achieve a more customized filter for this data, a FIR filter is designed using the Window method, which designs a FIR filter using a specified window. Figure 4.6 shows a block diagram of a FIR filter, and Figure 4.7 shows the filter applied to the LifeLine data. Because the past states are not used, the transfer function has a denominator of 1:

$$T(z) = \frac{b(z)}{1} \tag{4.2}$$

Another filter option for this data is the Savitzky-Golay (SG) filter, which increases the signal-to-noise ratio with out significantly distorting the signal. The SG filter

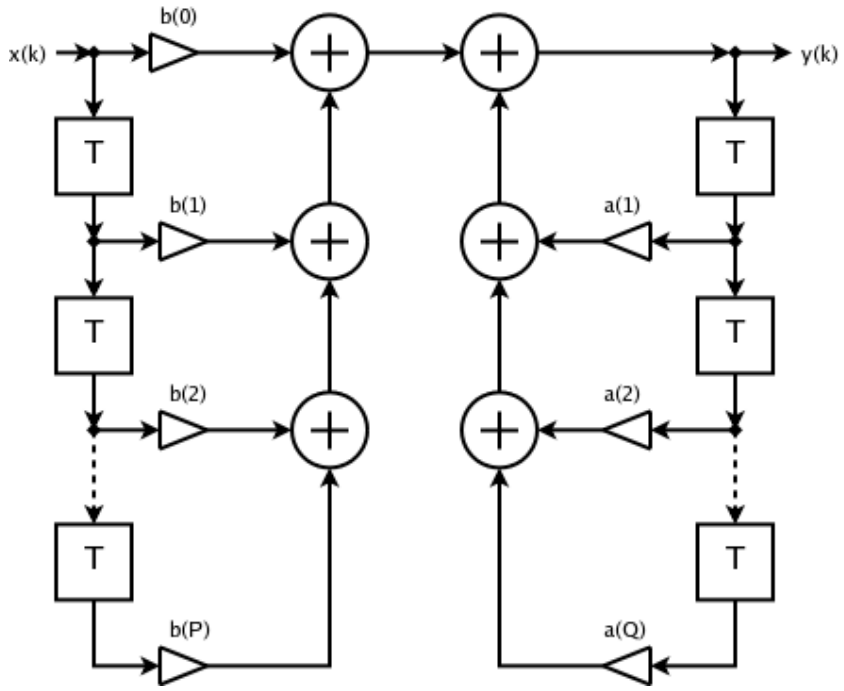

Figure 4.4: A block diagram of a standard IIR filter. [6]

uses convolution to fit a polynomial on a sliding window to the data. The output of this filter can be calculated using the following equation [18]:

$$Y_j = (\mathbf{C} \otimes y)_j = \sum_{i=-\frac{m-1}{2}}^{\frac{m-1}{2}} C_i y_{j+i} \tag{4.3}$$

C_i are the convolution coefficients (m total coefficients), y is the sampled data point, and j is the index of the data point.

The convolution coefficients can be selected from a table, but it is much more robust to calculate them prior to performing any filtering. The convolution coefficients, $\mathbf{C}$, are calculated using a variable change. The time vector for a given window, $\vec{x}$, is converted to $\vec{z}$ using the time step (Δt) with the following equation:

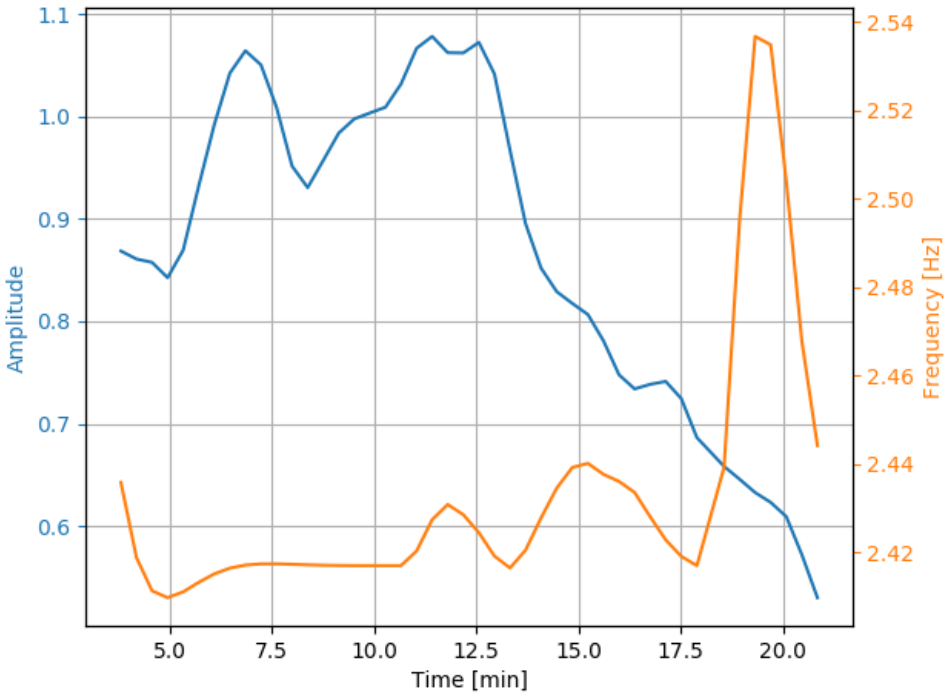

Figure 4.5: The Figure 4.3 data with a Butterworth IIR filter applied. The filter has an order of 2 and a normalized cutoff frequency of $0.1 \frac{\text{rad}}{\text{sec}}$

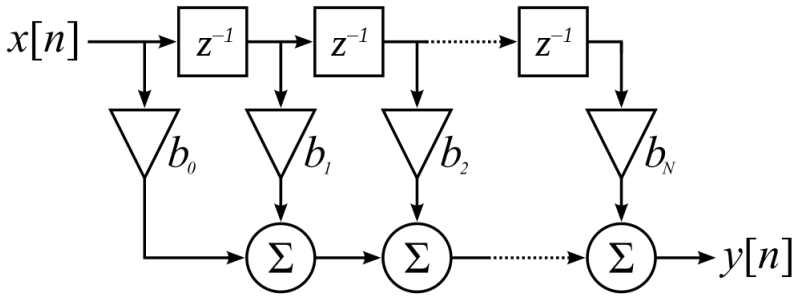

Figure 4.6: A block diagram of a standard FIR filter. [5]

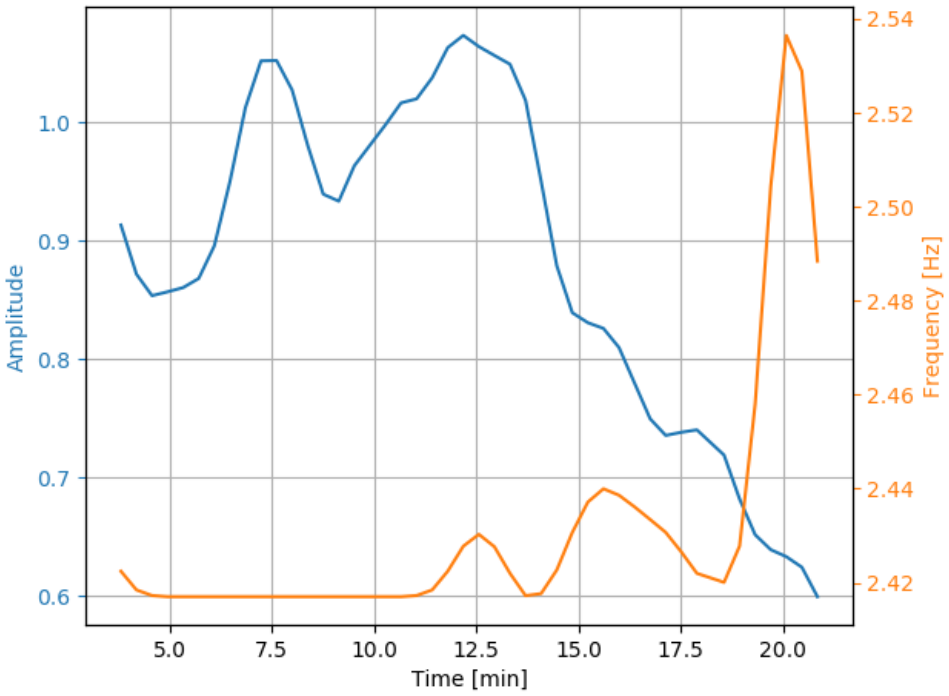

Figure 4.7: The Figure 4.3 data with an Window FIR filter applied. The filter has an order of 8 and a normalized cutoff frequency of $0.1\frac{\text{rad}}{\text{sec}}$.

$$z = \frac{\vec{x} - \bar{x}}{\Delta t} \tag{4.4}$$

$\bar{x}$ is the time value at the central point of the window. $\vec{z}$ is used to calculate the matrix $\mathbf{J}$, where each $i-$throw of $\mathbf{J}$ has the values $1, z_i, z_i^2, z_i^3, ..., z_i^n$. After $\mathbf{J}$ has been created, the convolution coefficients can be calculated as follows:

$$\mathbf{C} = \left(\mathbf{J^T J}\right)^{-1} \mathbf{J^T} \tag{4.5}$$

The filtered data can then be calculated as the convolution of $\mathbf{C}$ and the unfiltered data (Equation 4.3).

4.1.3.2 Applying the filter

A set of balanced data should have a constant maximum amplitude frequency with small amounts of high frequency signal, so A FIR or IIR filter is preferable over the SG filter. Linear phase is not required for this filter, so an IIR filter design is chosen to achieve a lower filter order. A Butterworth IIR filter, which is a type of signal processing filter designed to be flat in the passband, is ideal for this situation because the low frequency data is not distorted.

The filter parameters for this design are shown in Table 4.1. For a second order analog Butterworth filter, the following equation represents the generic transfer function [**?**]:

$$H(s) = \frac{1}{s^2 + \sqrt{2}s + 1} \tag{4.6}$$

This equation needs to be converted to the digital form, so a bilinear transformation needs to be applied, where:

Table 4.1: IIR filter parameters

Variable	Value	Description
n	2	Filter order
ω_n	0.2	Normalized cutoff frequency

$$s = \cot(\omega_n) \frac{1 - z^{-1}}{1 + z^{-1}} \tag{4.7}$$

Using the filter parameters in Table 4.1, the Butterworth IIR filter design transfer function is calculated to be:

$$H(z) = \frac{B(z)}{A(z)} = \frac{0.0675 + 0.1349z^{-1} + 0.0675z^{-2}}{1 - 1.143z^{-1} + 0.4128z^{-2}} \tag{4.8}$$

Figure 4.8 shows the filter response for the Butterworth IIR filter compared to a standard moving average filter with a window of 4 samples. The IIR filter has a much stronger rejection of higher frequency components and only uses the last 2 samples (instead of the previous 4 samples that the moving average uses). The moving average filter is essentially just a FIR filter with equal magnitude coefficients.

4.2 Detecting an imbalance

TODO: More explanation about the types of machine learning algorithms. Explain the strengths and weaknesses of each method and include some background on machine learning. Start with an explanation about the comparison between machine learning and linear regression.

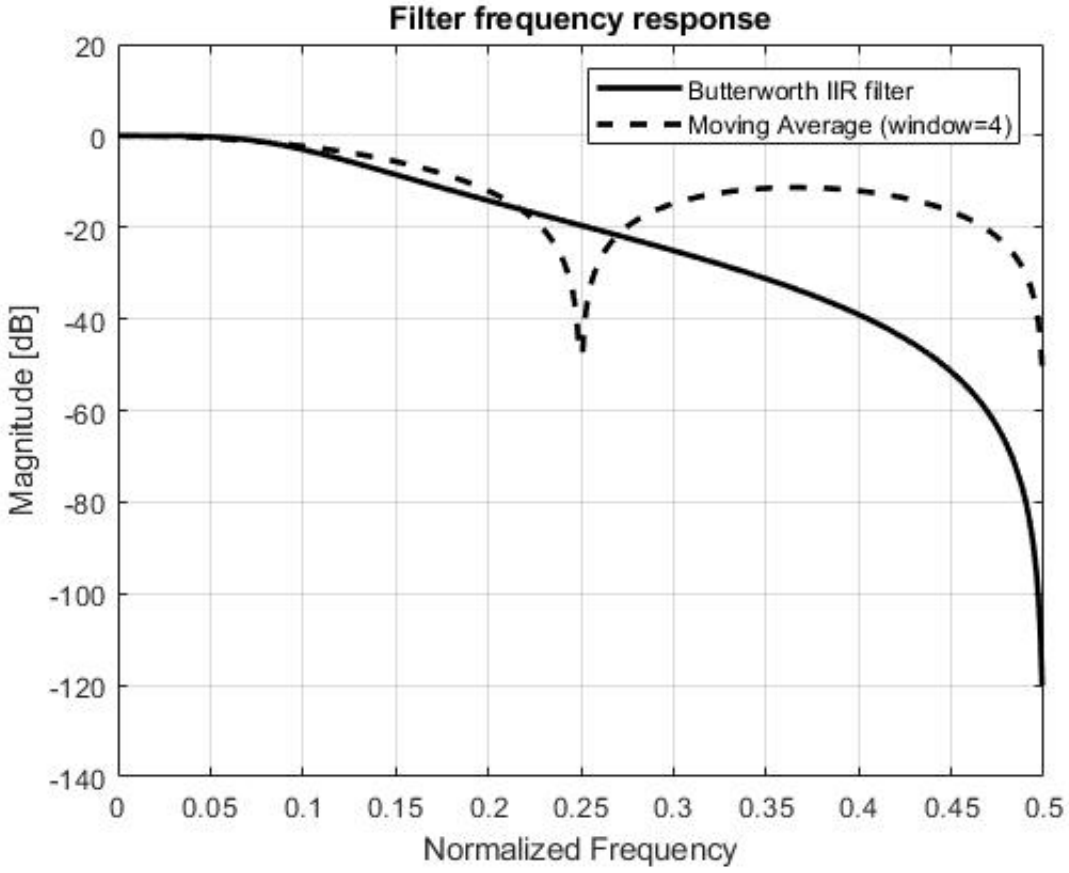

Figure 4.8: The filter response of the Butterworth IIR filter (Equation 4.8) compared to a moving average filter with a window size of 4. This figure was created in MATLAB with 2 simulated filter designs.

4.2.1 Selecting a classification algorithm

The crude and simple way to determine if there is an imbalance is to use a threshold criterion for the magnitude of the largest frequency component. This method does not utilize the frequency values that correspond to each magnitude. Since the rotor RPM is a range (not a constant value), it is likely that the magnitude will be slightly higher if the turbine is running faster and slightly lower if the turbine is running slower. Additionally, the threshold method does not adapt the algorithm to patterns and new information gained from more data.

This is a standard "classification" problem, where the algorithm needs to classify the state of the turbine as *balanced* or *not balanced*. With the modern availability of computational power, classification algorithms tend to fall under the "machine learning" category. These algorithms consist of a model that adapts to training data in order to minimize the error. A few of the common models include Logistic Regression (LR), Linear Discriminant Analysis (LDA), K-Neighbors Classifier (KNC), Decision Tree Classifier (DNC), Gaussian Naive Bayes (GNB), and Support Vector Classification (SVC).

The Logistic Regression model is commonly used to estimate discrete values (0/1 or false/true) and uses the logistic curve equation as the probability model. The Linear Discriminant Analysis model address many of the shortcomings of the Logistic Regression model, such as multi-class classification and a higher stability. Because the turbine classification problem only has 2 classes, this model is not likely to be the most optimal.

The K-Neighbors Classifier model is one of the simplest models, but can be the most accurate and powerful when dealing with fairly small datasets. This model essentially "memorizes" the training data and plots the points in n-dimensional space. The

classification of the new point is calculated based on the distances between the new point and the training points. This model is likely to be an accurate algorithm for the turbine problem.

Decision Tree Classification relies on the ability to split the data set into binary categories and subcategories, where the algorithm can follow yes/no decisions to lead to the correct class.

The Gaussian Naive Bayes model is a statistics based model that modifies a prior belief bell curve based on the training data. This is a statistical method of classifying data where the data set statistics are constantly updated with new data (as opposed to static models, such as mean, median, and standard deviation classification).

The Support Vector Classification model creates a n-dimensional space where the training data points are divided by a clear gap that is as wide as possible.

To compare the listed classification models against each other, the accuracy of each model is plotted using the same training and test data. Figure 4.9 shows that the K-Neighbors Classification algorithm has one of the highest mean accuracy ratings with the smallest variation is accuracy scores. This is also the simplest to design and implement, which makes it an ideal choice for the imbalance detection algorithm that may need to run on a microprocessor attached to the turbine tower.

4.2.2 Applying the K-Neighbors Classification Algorithm

4.2.2.1 About the algorithm

The K-Neighbors Classifier (also known as the K-Nearest Neighbors Classifier or K-NN Classifier) is a type of "lazy learning" [1], which means it memorizes the training

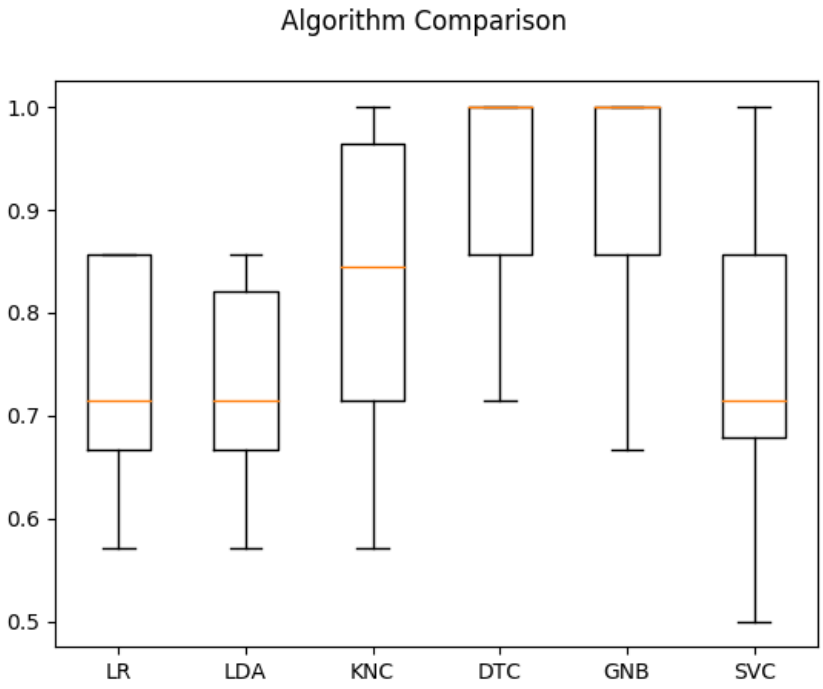

Figure 4.9: A statistical comparison of different classification algorithms. The algorithms compared are: Logistic Regression (LR), Linear Discriminant Analysis (LDA), K-Neighbors Classifier (KNC), Decision Tree Classifier (DNC), Gaussian Naive Bayes (GNB), and Support Vector Classification (SVC). TODO: Add axis label

data and uses it to calculate the test data classification. This results in a very short training time, but a longer calculation time curing the classification process.

The K-NN algorithm uses a "majority voting" method where the Euclidian distance between all the training data points and the test data is calculated. This model assumes there is a relatively equal amount of balanced rotor experimental data and imbalanced experimental rotor data, which means the sample weighting can be uniform. If there is a much higher frequency class (for example, much more balanced experimental data), this class will tend to dominate the calculations regardless of the actual class of the test data.

The number of neighbors, k, is chosen to be 5 for this algorithm (using trial and error). A larger value of k will reduce the classification noise, but the boundaries are much more general and not as tuned to the training data. A lower value of k will match the model very closely to the training data, but may add noise into the classification process.

The Euclidean distance is simply the distance between the data points. This equation is shown below:

$$d = \sqrt{\sum \left(x_{i,a} - x_{j,a}\right)^2} \tag{4.9}$$

4.2.2.2 Algorithm Results

To apply the algorithm, the data is converted into short lists containing a maximum amplitude, a corresponding frequency value, and the known class. For this test, there are 55 experimental DFT results for a balanced rotor and 55 experimental DFT results for an imbalanced rotor. Applying the algorithm on raw DFT data produces

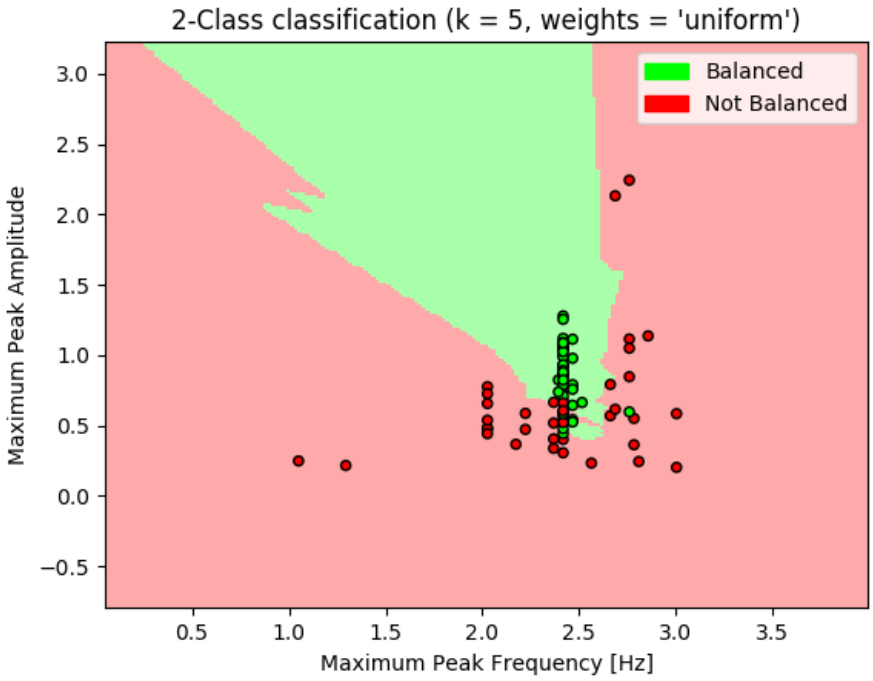

Figure 4.10: The classification boundaries for test and training data that is not filtered.

the classification boundary plot shown in Figure 4.10. Using unfiltered data is slightly faster (there is no filter lag), but the accuracy is significantly reduces. It is much more accurate to filter the data first, if the lag is an acceptable consequence. The unfiltered data produces an accuracy of about 86 percent with a confusion matrix:

TODO: Explain what a confusion matrix is.

$$C_{confusion} = \begin{bmatrix} 13 & 0 \\ 3 & 6 \end{bmatrix} \tag{4.10}$$

Figure 4.11 shows the classification boundaries with data filtered using an IIR filter. A clear trend can be seen in this data, with distinct class boundaries. This produces an accuracy of 94 percent with a confusion matrix:

$$C_{confusion,filtered} = \begin{bmatrix} 6 & 1 \\ 0 & 11 \end{bmatrix} \tag{4.11}$$

The filtered data produces a much more accurate detection model because much of the time-dependent noise is suppressed. It is difficult to draw conclusive results from these test results, since there are only 55 sets of balanced and imbalanced data. 80 percent of this data is used to train the detection algorithm, while the remaining 20 percent is used to perform the verification testing shown in the confusion matrices and accuracy results.

Both the amplitude and frequency values are important in detecting an imbalance in the turbine rotor. An imbalance will cause overall higher acceleration amplitudes, but will also cause a larger variation in rotor speeds. As shown in Figure 4.11, the imbalanced data has large variations in both the maximum amplitude and frequency of the LifeLine DFT data. When there is an imbalance in the rotor, the closed-loop transfer function of the changes, causing error in the tuned rotor speed controller parameters. This causes some instability in the controller that is identified by the DFT results of the LifeLine module.

Additionally, this algorithm can also be made more robust by looking for 10 or more positive imbalance detections in a row before flagging an alert. This will remove any false positives that could shut the turbine down unintentionally.

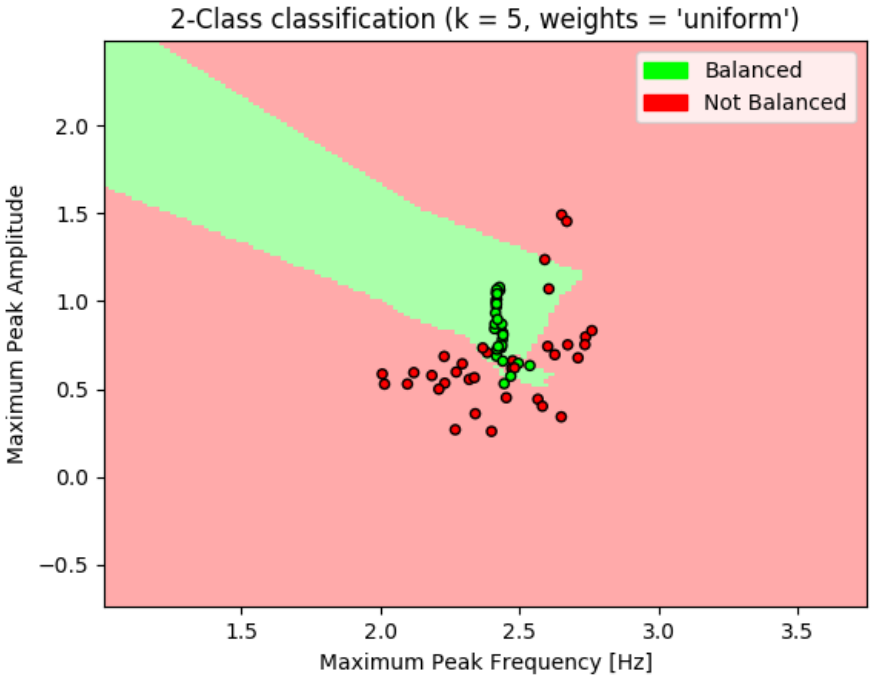

Figure 4.11: The classification boundaries for test and training data that *is* filtered.

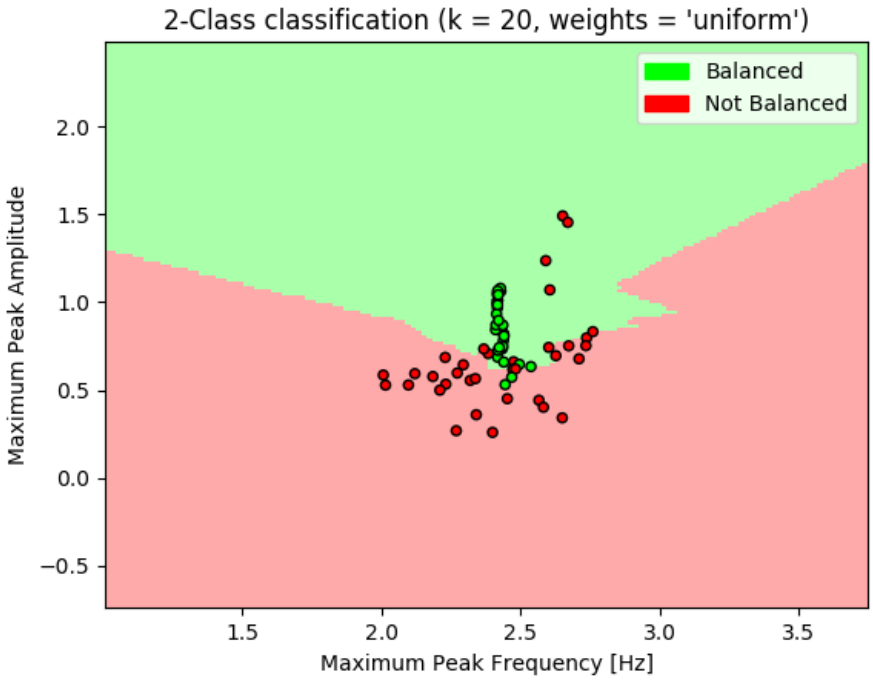

Figure 4.12: The classification boundaries with $k = 20$ and uniform weighting.

4.3 Analyzing the Detection Algorithm

4.3.1 Using Different Parameters

The main parameters to tune for the detection model are k and the weighting. In the KNN algorithm, the classification decision is based on the weighted vote from k neighboring samples. This means that a larger k value will include more data in each decision, but may be lest accurate. A smaller k value will create tight classification boundaries, but could be prone to over-fitting. Figure 4.12 and Figure 4.13 show the effect of changing the k value on the classification boundaries.

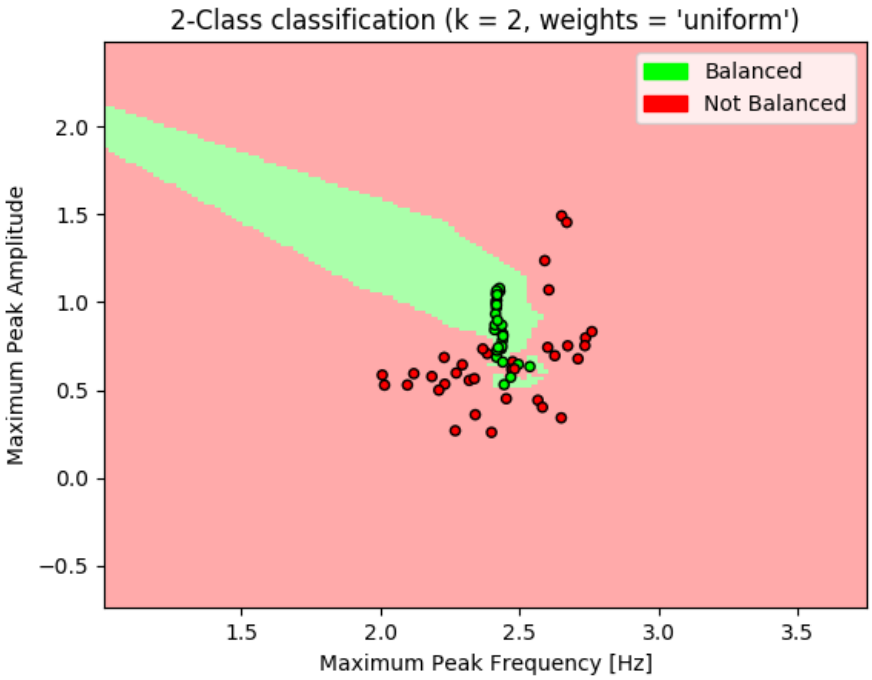

Figure 4.13: The classification boundaries with $k = 2$ and uniform weighting.

4.3.2 Potential Problems

One of the biggest problems with the KNN classification algorithm is the computational cost. Because this is a "memory-based" model, the training data is essentially locked in memory and used to classify any new inputs. By changing the model, the computational cost can be reduced; however, these are much more complicated to implement and not always as accurate.

Additionally, the IIR filter has a fairly significant and nonlinear delay that increases near DC frequency components. Faster filters can be used at the cost of accuracy because they will produce more noise or a more distorted signal. For this application, it seems like accuracy is more important than speed, and the IIR filter parameters were designed around this idea.

This detection algorithm is best developed with real experimental data, which means it requires actual data before it is of any use. If a sufficient enough model is developed, the model can produce the test data to train the algorithm instead of collecting it in the field; however, this is likely to be less accurate.

4.3.3 Future Changes

TODO: Add a section about algorithm scalability and switching to a model-based machine learning algorithm (like a neural network) if many training sets are acquired.

One of the significant benefits with this algorithm is the flexibility. It is very easy to add more data to improve the imbalance detection. For example, in addition to peak frequency and magnitude, the wind speed and generator power can be included as parameters to improve the model. With enough variables and experimental data,

the model will become more accurate and robust (although it is difficult to visualize the classification space with orders higher than 2).

In addition, the entire frequency spectrum can be used as an input to the algorithm (instead of just using the peak values), but the model becomes much more complex. With a large amount of experimental data, the KNN model might need to be changed to a less computationally intensive model, such as a neural network.

Just using a classification algorithm on raw data isn't always useful. It is very important to properly choose a data processing methods, such as the IIR filter used in this paper. Selecting the filter is just as important as choosing the model for this type of data.

TODO: Add a section with conclusions, recommendations for future work, and methods for algorithm improvement. Add code to a public repository.

BIBLIOGRAPHY

[1] K-nearest neighbors. *Scholarpedia*, 2009.

[2] Lock-in amplifier and applications. 2011.

[3] About lock-in amplifiers. *thinkSRS*, 2017.

[4] Envelope detection. *MathWorks*, 2018.

[5] Wikipedia: Finite impuse response. `https://en.wikipedia.org/wiki/Finite_impulse_response`, 2018.

[6] Wikipedia: Infinite impulse response. `https://en.wikipedia.org/wiki/Infinite_impulse_response`, 2018.

[7] K. Banks. The goertzel algorithm. *Embedded*, 2002.

[8] D. A. R. Collins. Zoom fft.

[9] R. M. Edward Malnick. 15000 accidents and incidents on uk wind farms. *Telegraph*, 2011.

[10] B. Evans. Goertzel algorithm. *Berkeley*.

[11] T. Gwon. Structural analyses of wind turbine tower for 3 kw horizontal-axis wind turbine. *California Polytechnic State University, SLO*, 2011.

[12] T. Instruments. Modified goertzel algorithm in dtmf detection using the tms320c80. *Digital Signal Processing Solutions*, 1996.

[13] I. International. Decimation. *dspGuru*, 2017.

[14] R. Lyons. Using zoom fft for spectral analysis. *Embedded*, 2011.

[15] E. G. Maik Reder, Julio Melero. Wind turbine failures - tackling current problems in failure data analysis. *Journal of Physics Conference Series*, 2016.

[16] W. McCutcheon. Deflections and stresses in circular tapered beams and poles. *Civil Engineering for Practicing and Design Engineers*, 1983.

[17] D. I. Peng Guo. Wind turbine tower vibration modeling and monitoring by the nonlinear state estimation technique (nset). *MDPI*, 2012.

[18] W. H. Press and S. A. Teukolsky. Savitzky-golay smoothing filters. *American Institute of Physics*, 1990.

[19] M. D. Reder. Wind turbine failures - tackling current problems in failure datra analysis. *Journal of Physics: Conference Series*, 2016.

[20] A. D. B. Thomas Kenbeek, Stella Kapodistria. Data driven online monitoring of wind turbines. 2016.