PiRail

Team Member: Jeff Fernandes, Ben Grimes, Maximillian Hennessey, Liqi Li
Project Sponsor: Jonathan Miner
Department of Computer Science, University of New Hampshire

Introduction

Background - Railroads face the challenge of inspecting and maintaining miles of track, often involving time-consuming manual inspection. PiRail has developed a hardware platform to collect GPS budget-friendly (location), IMU (vibration), LIDAR (laser measurement) and LPCM (acoustic) data.

Proposed Solution – Build a web-based application to analyze and plot multiple IMU datasets. Give the user the ability to pan and zoom through the dataset.

Goals

- Trade study on algorithms to process IMU data.
- Analyze and plot a single dataset.
- Analyze and plot multiple datasets.
- Build a web-based application compatible on any device that demonstrates calculated data for the user.

Results

After track data is recorded, it can be uploaded to our website and displayed on a graph with thousands of data points. The user can zoom throughout the graph and even reset it to the initial state. We also calculate outliers and other various points of interest. The website can also be used to learn more about our product and how to get in contact with us. Excluded from the final product, our ML algorithms and results.

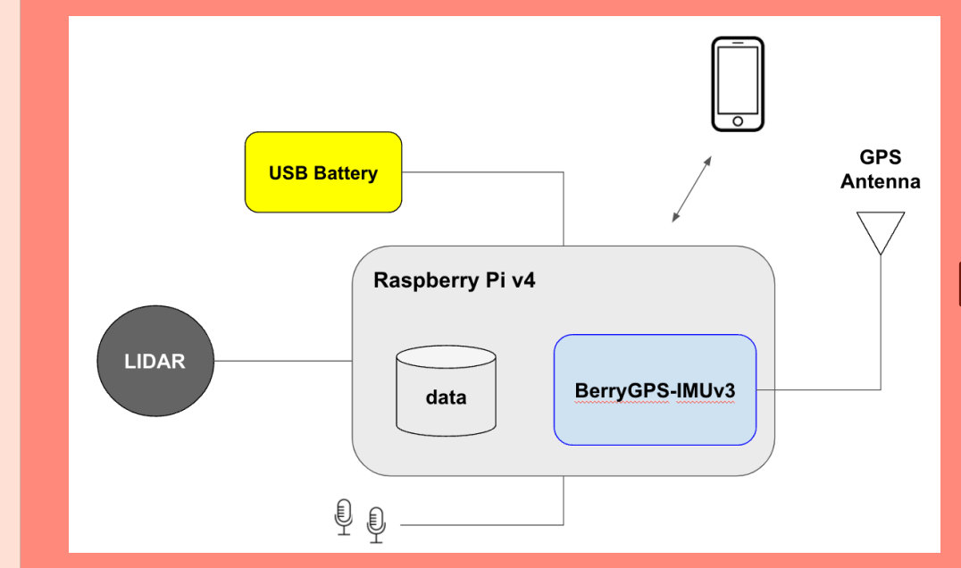
Future Work

- Enhance the web interface to display LIDAR and LPCM data.
- Apply ML algorithms to proper training and testing data.
- Enable the user to select from a wide range of datasets.
- Enhance frontend for optimal usage.

Data Process

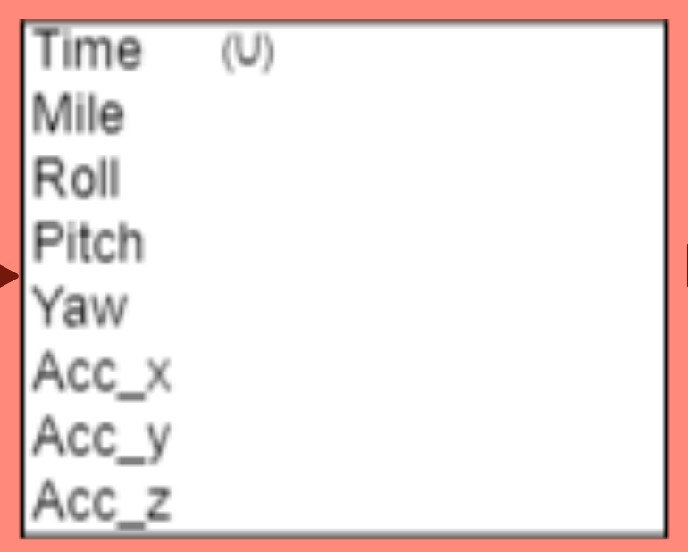
Data Collection

Data Collection Device



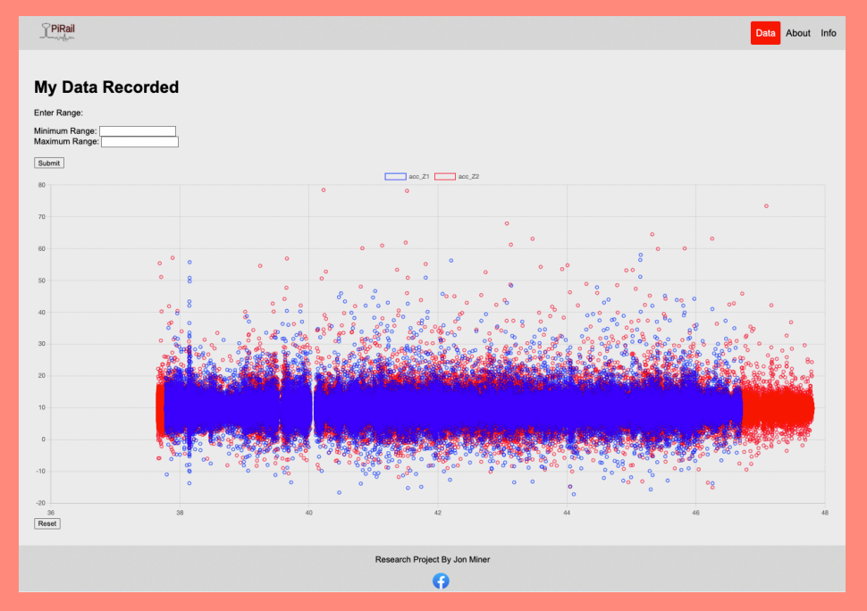
Prior Work:
A Raspberry Pi with LIDAR and GPS enables us to record track data.

Database


Time (U)
Mile
Roll
Pitch
Yaw
Acc_x
Acc_y
Acc_z

The track data is then recorded into our database which is filtered by axes.

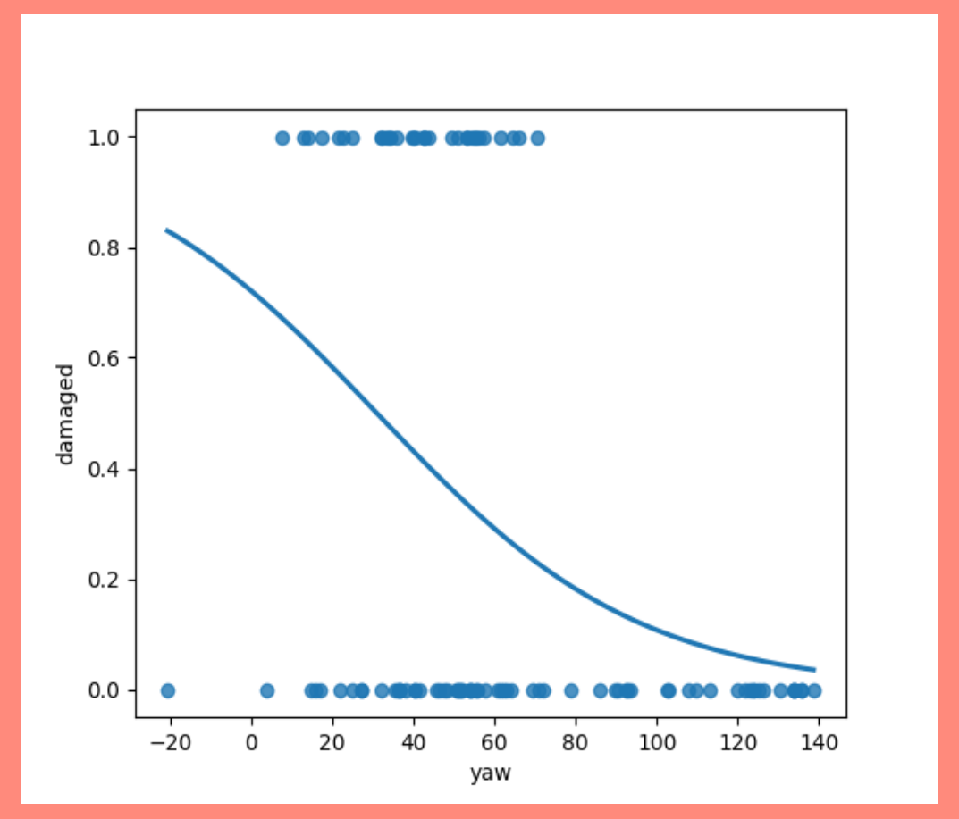
Frontend



The database is then used to display track data onto our frontend for the user.

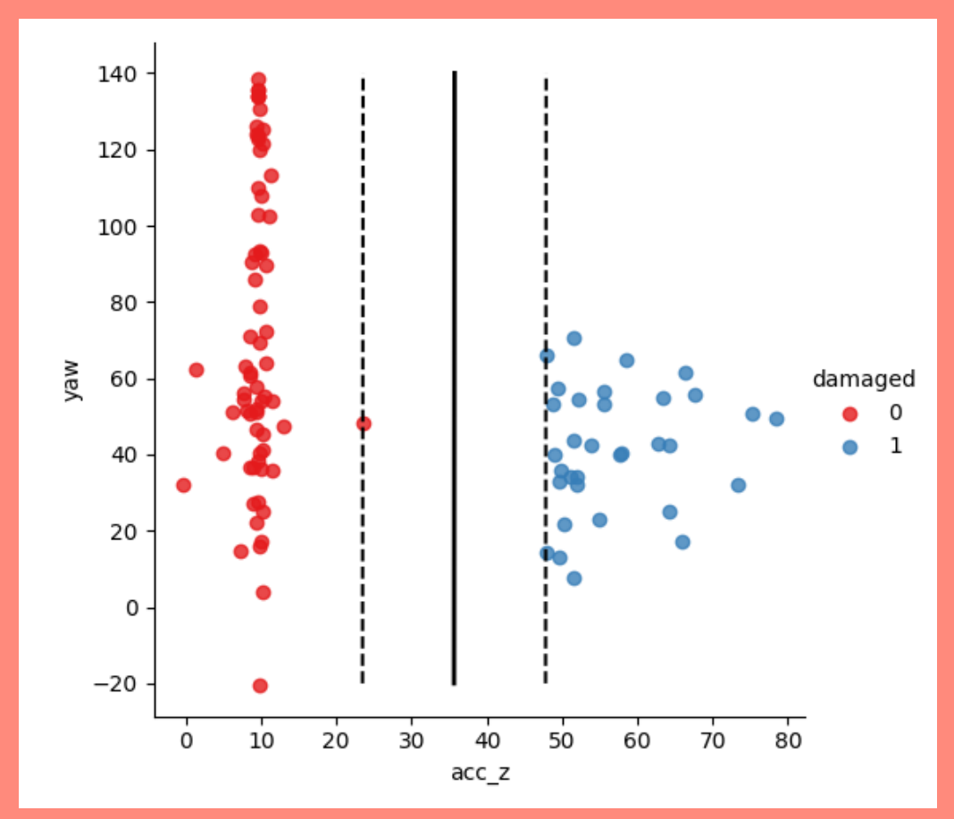
ML Results

Logistic Regression



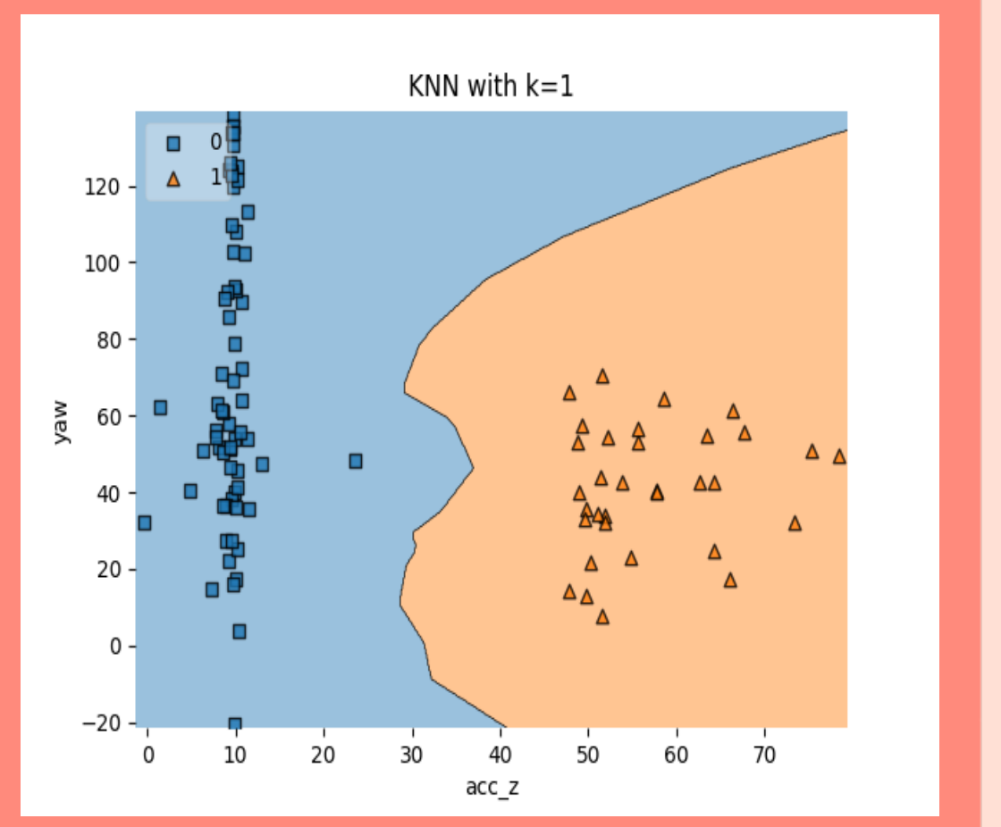
Probability curve on a binary scale constrained between 0 and 1. 0 being undamaged and 1 being damaged.

SVM



The hyperplane separates our two classes, while the margin is calculated as the perpendicular distance from the line to only the closest points, these are our support vectors that define the hyperplane.

KNN



This graph depicts KNN when K = 1. Any new data point would be classified to its 1 closest neighbor as highlighted in the graph.

Architecture Design

Frontend - Our web interface uses the open-source Chart.js library. This library is well documented, easy to use, and supports several different graph types. The best choice for this application was the "scatter" plot which enables us to plot individual data points. Chart.js is enhanced with functions that enable panning and zooming through the dataset. With this framework, we have a visual representation of two of the datasets recorded prior to our work. To enhance the user experience, we added many features and details to the website, such as images and device explanations. We also added margins around the contents to enhance the content's readability and attractiveness. The website also included a header and footer to help users navigate throughout the website.

Backend - we can access arguments passed from the client and from there either select an existing data set or create a new smaller one from a larger existing one. This is based on the arguments passed in by the client. Once the data set has been created and/or selected, the server performs a statistical analysis on the data set. It calculates basic things such as averages, minimums, maximums, and standard deviation. Once calculated it creates and stores this data on the server (as to be returned faster if the same request is made again) and then returns all information back to the client as JSON objects.

Development Tools

IntelliJ Discord Drive Github Python JavaScript

Machine Learning

- **ML Classification Algorithms:** Logistic Regression, K-Nearest Neighbors, Support Vector Machines.
  - **Features:** Linear acceleration on the x, y and z axis. Pitch, Roll and Yaw.
  - **Response:** Binary classifier: potentially damaged or undamaged.
- **Cross Validation:** k-fold cross validation.
  - **Results:** Support Vector Machines provided the best results: highest accuracy.
- **Limitations:** Insufficient amount of prior data collected for training and testing.
  - **Solution:** Created an algorithm using linear acceleration on the z axis to find outliers outside of x number of standard deviations. This became the classier for our training and test data.
  - **Issues:** Without actual track health data and using our temporary solution we are not classifying health but instead classifying outliers for the linear acceleration on the z axis. Also, because we are using the linear acceleration on the z axis we must omit it from our features as it is essentially the response.
- **Expandability:** All classification algorithms have been created using our own implementations and open-source libraries. The focus of our last sprint was making this as expandable as possible, only needing (any) features and a (any) response to utilize our ML implementations.

Testing

We will use existing PiRail software as test cases. We need to ran prior code to see the max point, then compare it with our chart to see if it is close or different. We also need to check if our chart can find outliers between multiple sets of data.

For our learning model we will gather testing data from prior data collections and split it up between test and training data. Train the model with our training data and test with testing data. We create our test and training data by finding outliers on the z-axis of the linear acceleration. In the future we hope to have explicit training and testing data.