Boston University
Electrical & Computer Engineering
EC464 Capstone Senior Design Project

User's Manual

EZRider

Submitted to

Professor Alan Pisano
Room 522, 8 St. Mary's Street,
Boston, MA 02215,
+ 1 (617) 353-6264
apisano@bu.edu

by

Team 27
EZRider

Team Members

Luke Bacopoulos baco@bu.edu
Aymane El Jerari aymanelj@bu.edu
Balaji Sathyanarayanan balajis@bu.edu
Jason Wang jaswang@bu.edu
Haoming Yi yiha@bu.edu

Submitted: 04/18/2023

EZRider User's Manual

Table of Contents

Executive Summary

EZRider is a road roughness detection app that uses machine learning and smartphone technology to assess the quality of road surfaces. The app collects images and accelerometer data from smartphone sensors while the driver moves along the road. The accelerometer data is fed into a machine learning algorithm to determine road roughness, and the data is displayed on a map within the app, showing where the road roughnesses are located. EZRider was specifically designed to help drivers identify areas that need repair, thereby contributing to making the city's roads safer and more sustainable. The app is user-friendly and intuitive, making it easy for drivers to navigate and use, and it helps municipal authorities identify and prioritize roads that need maintenance, ultimately saving them time and money in the long run.

1 Introduction

As urbanization continues to expand, the number of vehicles on the road is also growing at an alarming rate. As a result, this has led to more wear and tear on road surfaces, which can result in a bumpy and uncomfortable driving experience. Poor road conditions can lead to increased vehicle maintenance costs and even accidents, which can be detrimental to drivers, passengers, and other road users.

According to a report by the American Society of Civil Engineers (ASCE), one out of every five miles of highway pavement in the United States is in poor condition, resulting in increased wear and tear on vehicles and a greater risk of accidents. The ASCE report estimates that the cost of driving on roads that are in poor condition amounts to over $500 per year in extra vehicle operating costs. Boston has some of the worst road conditions in the United States, impacting the driving experience, and having economic and environmental consequences. According to a study by TRIP, a national transportation research group, 56% of major roads and highways in the Boston area are in poor condition, leading to increased vehicle operating costs and a greater risk of accidents.

To address this issue, we developed a road roughness detection app called EZRider, as our senior design project. The app utilizes machine learning and smartphone technology to assess the quality of road surfaces. Developed on Flutter, a popular cross-platform mobile application development framework, EZRider uses Firebase as its database. To detect road roughness, the app collects images and accelerometer data from the smartphone sensors while the driver moves along the road. The accelerometer data is then fed into a machine learning algorithm to determine what constitutes road roughness. The app displays the data on a map within the app, showing where the road roughnesses are located.

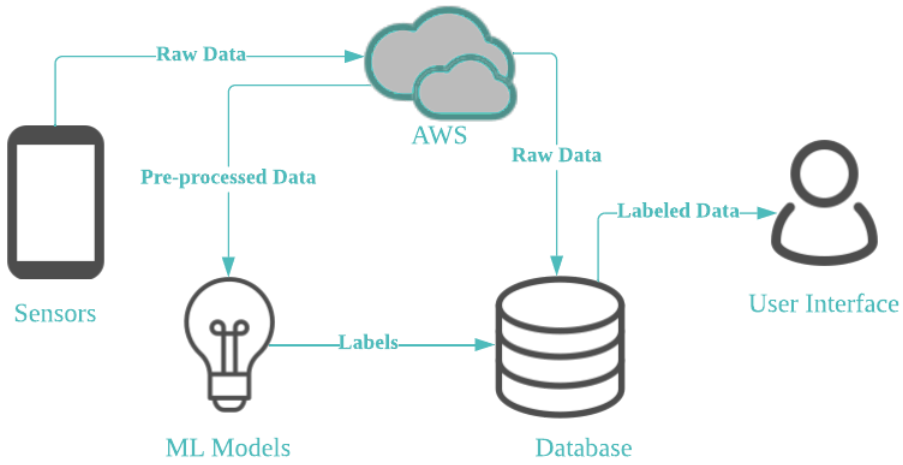
The EZRider app was specifically designed to help drivers identify areas that need repair, thereby contributing to making the city's roads safer and more sustainable. One of the benefits of this app is that it makes it easier for drivers to detect poor road conditions before they become a significant problem. The use of machine learning algorithms to analyze data from sensors in the phone helps to identify road roughness accurately. This information can then be used by municipal authorities to determine which roads require maintenance and repair, ultimately saving them time and money in the long run.

Moreover, EZRider is a user-friendly app that can be used by anyone, regardless of their technical expertise. The app's interface is intuitive and straightforward, making it easy for drivers to navigate and use. By using EZRider, drivers can help municipal

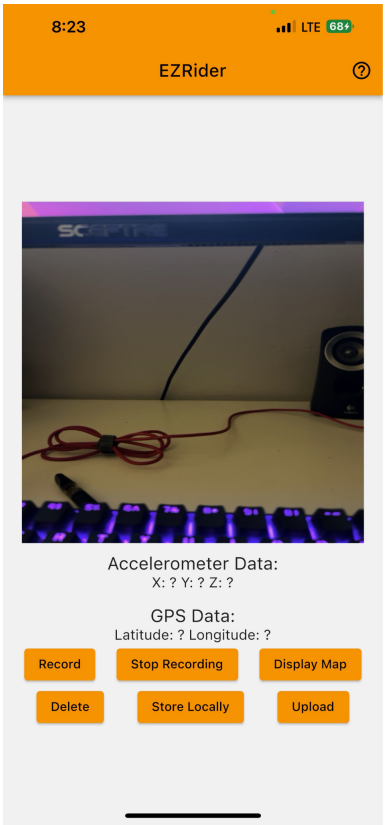
authorities identify and prioritize roads that need maintenance by simply placing their smartphones in their vehicles against the windshield, drivers can contribute to the collection of data. This app not only improves driving comfort and safety but also helps to reduce the environmental impact of unnecessary repairs and replacements.

2 System Overview and Installation

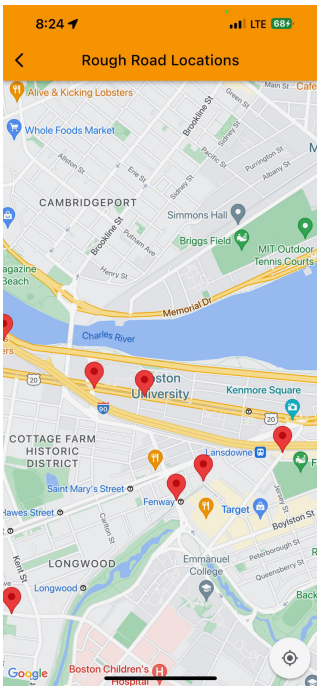
2.1 Overview block diagram



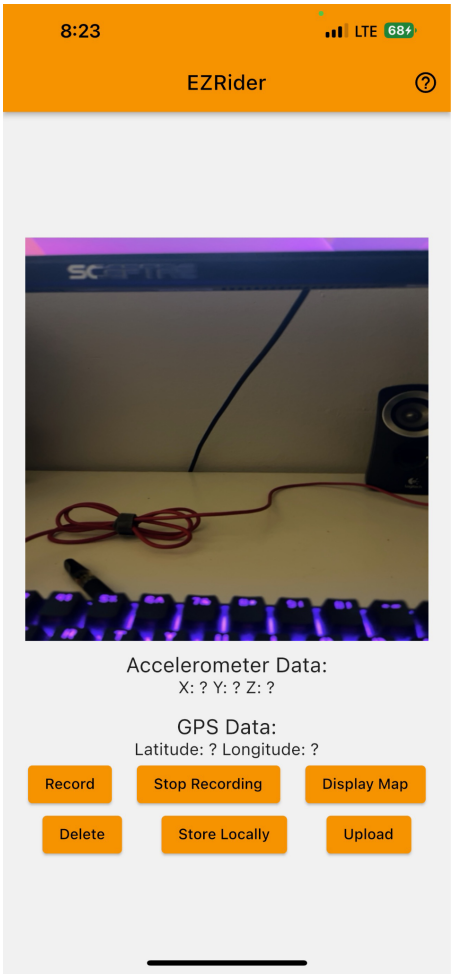
2.2 User interface.



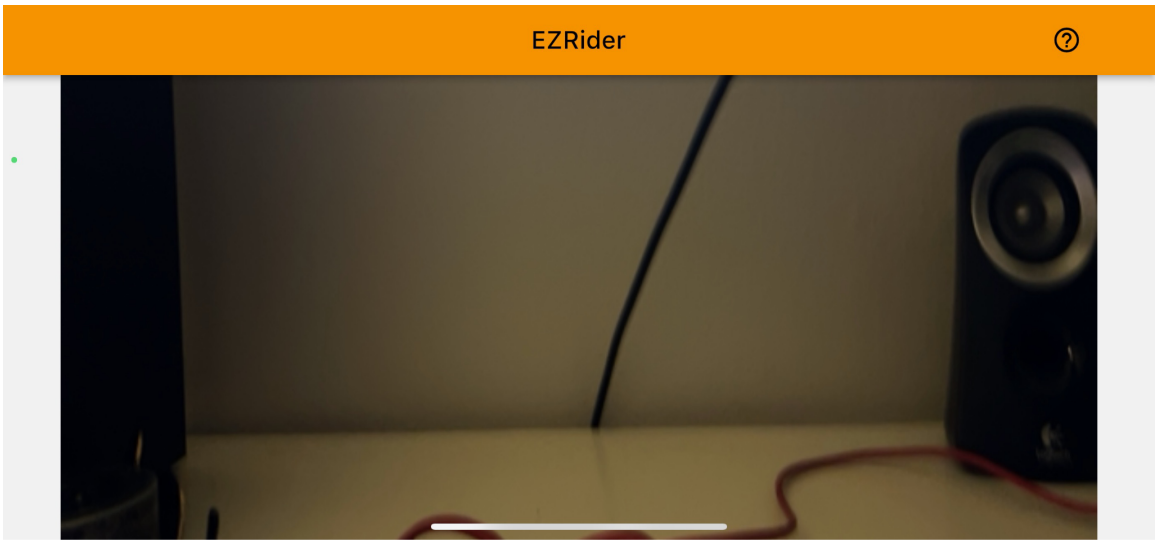
When viewing the main interface of our app, you will notice seven buttons, along with two rows of text and a square view of your rear camera. The button in the top right corner is our quick guide. The rest of the six buttons on the main interface are the main functioning points of the application. Their specific functions will be explained in detail in section 3.1. The two rows of information contain the latest reading of the phone's accelerometer data and GPS coordinates. The rear camera view should be used to ensure that the camera is taking quality pictures. Pressing the "display map" button will open the second part of the user interface:

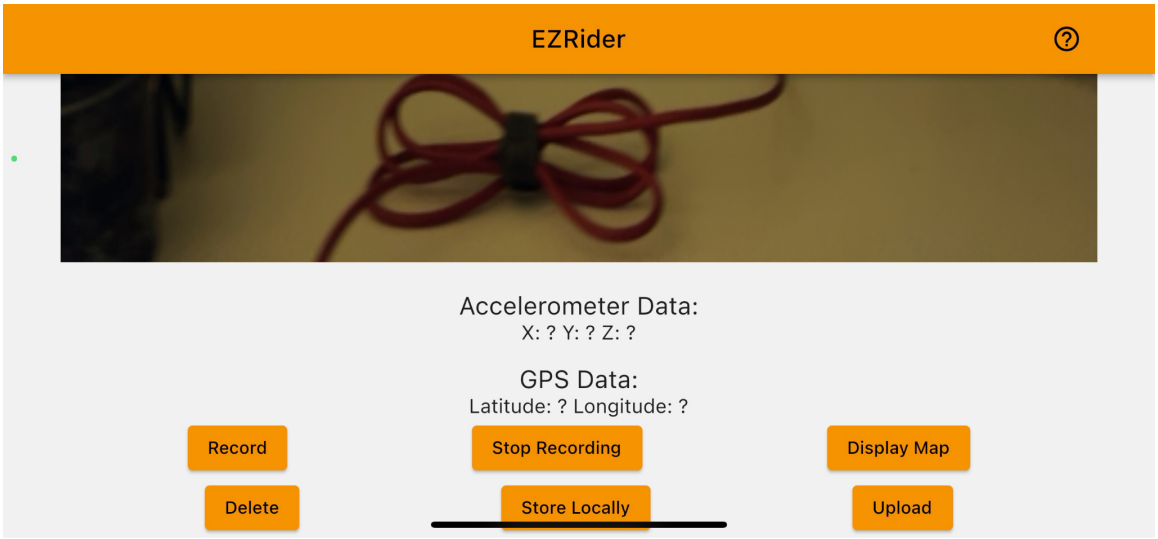


This is a basic google maps view centered in the Boston University campus, containing markers at locations of rough road readings. A button on this interface will zoom the view to your immediate location. In a future release, there will be a refresh button on this page that will update the latest rough road locations.



While in landscape orientation, the user must scroll down in order to access the buttons. The camera view in landscape mode is larger to ensure that quality pictures are being taken. Interface while in landscape orientation:





2.3 Physical description.

Our system's only hardware requirements include an iPhone running iOS version 15.0 or later, and a compatible rigid dashboard mount. The mount should allow for minimal movement of the phone when idle or driving over smooth roads. This allows for better accuracy in the accelerometer data. For best results, the iPhone should be in good condition with functioning rear camera, GPS, and accelerometer sensors.

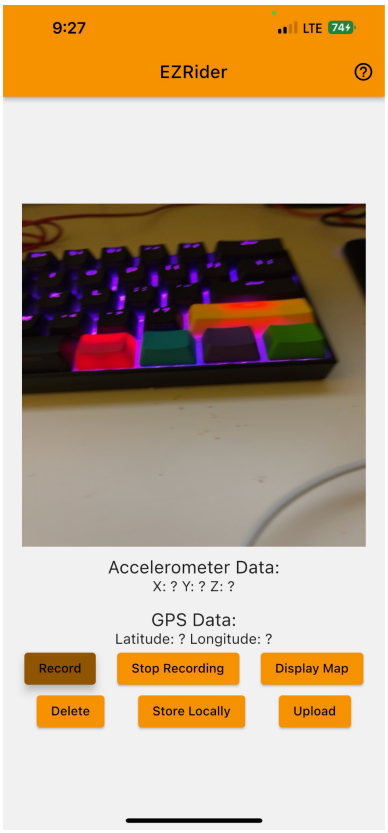
2.4 Installation, setup, and support

The first step is to install the EZRider app on your iOS device. Currently, you must be invited to test the app. In a future release, it will be readily available on the app store of any iPhone running iOS version 15.0 or later. As described above, you will also need a rigid dashboard mount. This mount will be placed on the windshield so that the iPhone's rear camera has a clear view of the road in front of you. On your first launch of the app, it will ask you for location service permission as well as access to the phone's camera. These permissions are necessary for the app to function properly. Once permissions are enabled, the phone can be secured into the dashboard mount and data recording can begin. A quick start guide is located in the upper-right corner of the interface, with quick instructions that can be conveniently accessed. A more detailed guide will be implemented within the app in a future release.

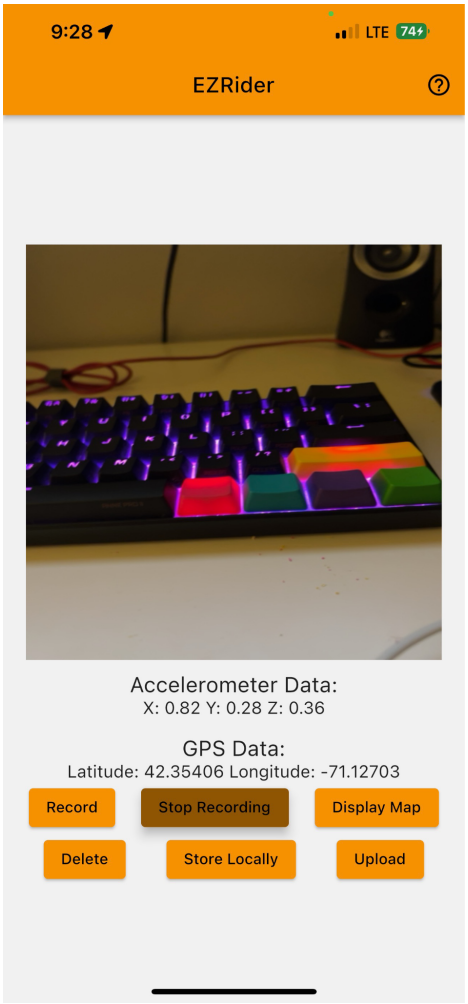
3 Operation of the Project

3.1 Operating Mode 1: Normal Operation

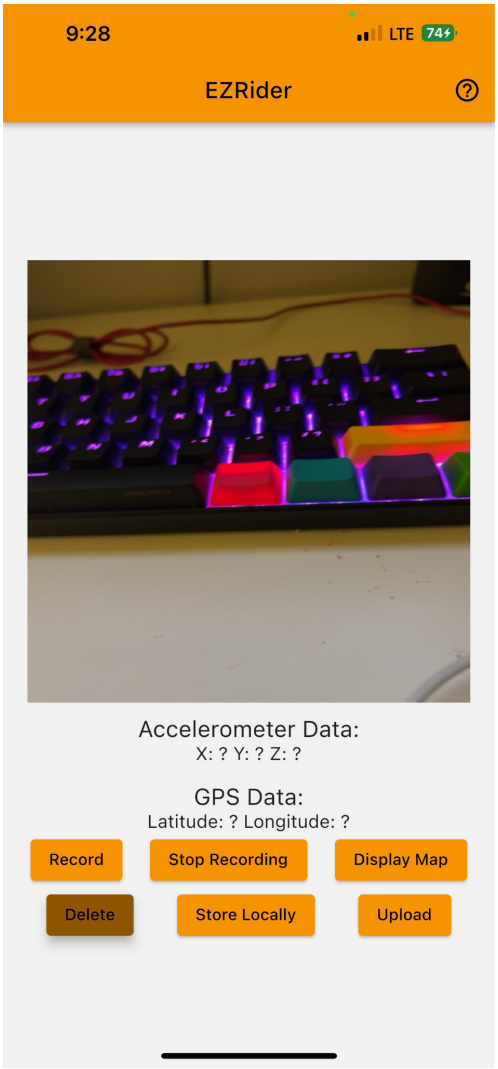
When you first open the EZRider application, it will be in an idle state. You will see the main user interface as outlined in section 2.1, with the six functionality buttons on the bottom being the main points of control. Each of these buttons have a specific use and function that will be outlined in detail here:



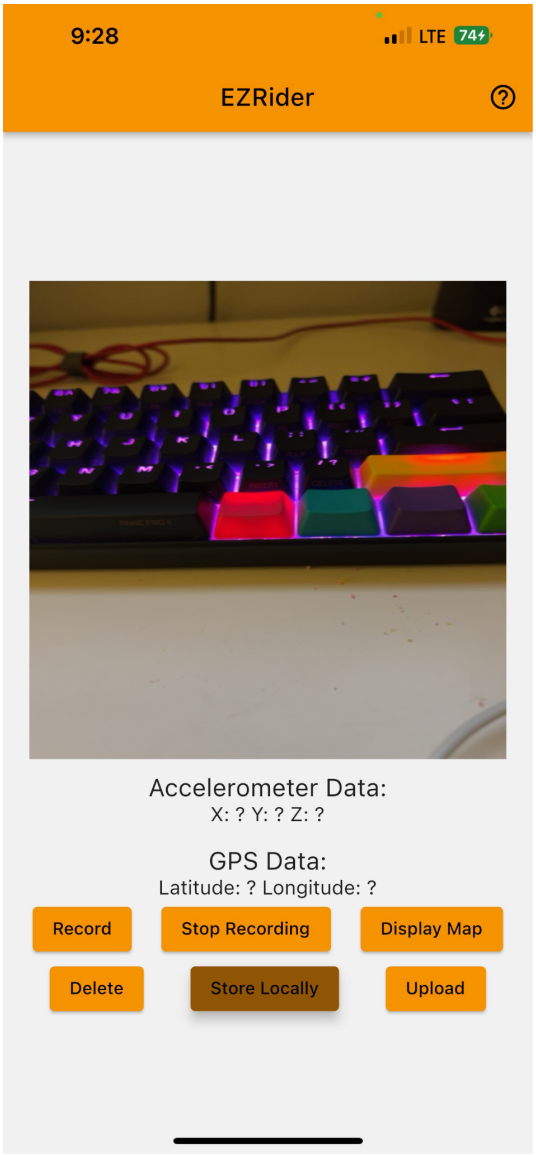
1. ***"Record":***
   Upon pressing the "Record" button, the application will start to collect data. Photos will be taken using the rear facing camera at a specified interval, which can be confirmed when the iPhone is not on silent mode, as the shutter sound will be triggered with each picture taken through the app. The phone's GPS location will also start being listened to and displayed on the interface, along with the readings from the accelerometer. Pressing the "Record" button should always be a deliberate action, as this begins the collection of what is considered sensitive data. Accidental triggers of the "Record" button should not be an issue however, as the collected data can be deleted off your local storage before it goes to our cloud storage.



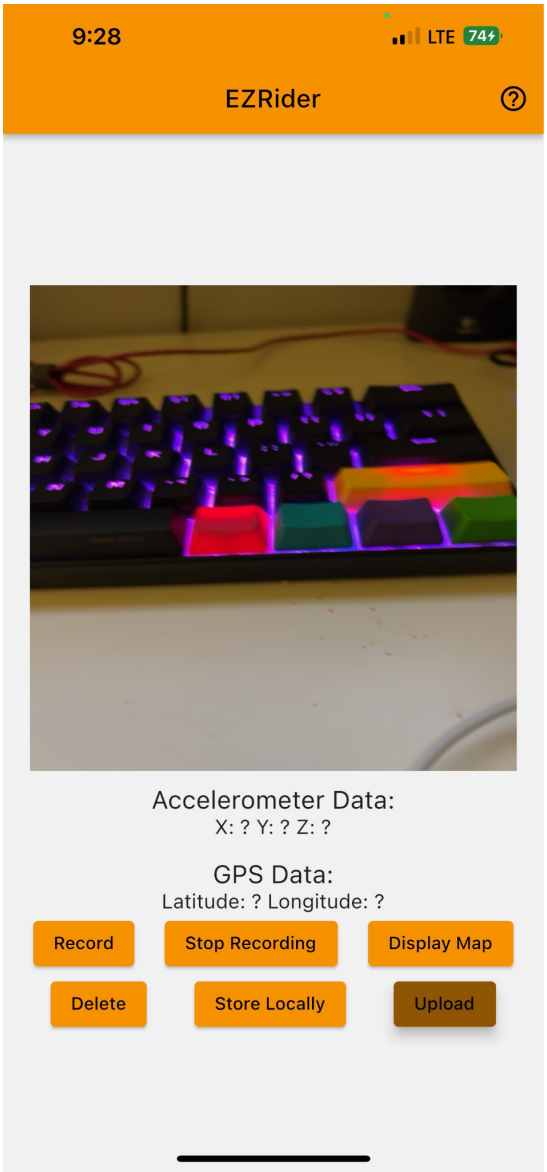
2. ***"Stop Recording":***
   This button terminates all processes that began with the press of the previous "Record" button. You should press this button as soon as you are finished collecting data, or if you have pressed "Record" by mistake. This will return the app to its original idle state.



3. ***"Delete":***
   Pressing the "Delete" button will remove all data points currently stored on the iPhone's local storage. It is recommended that the user deletes the data stored locally immediately after uploading it to the cloud, in order to reduce redundancy. This button should be pressed carefully, as the data that was previously stored will no longer be able to be uploaded to the cloud. The user should use this button to remove any low quality data or accidentally collected data, before uploading any data to the cloud.



4. ***"Store Locally":***
   When the user presses the "Store Locally" button, all data from the previously recorded session will be stored on the iPhone's local storage. The reason for this button is to allow the user to not have to upload a potentially large amount of data to the cloud using their cellular data. It is recommended that the user stops recording before storing locally, in order to minimize buggy interactions.



5. ***"Upload"*:**
   Upon pressing the "Upload" button, all data points currently stored in the iPhone's local storage will be uploaded to the cloud in a json file labeled with the timestamp of that upload. It is recommended that the user only uploads high quality data using this button in order to ensure accuracy. It is also recommended that the user is not recording when pressing "Upload", and that the user deletes data stored locally immediately after uploading.

6. ***"Display Map":***
   Upon pressing the "Display Map" button, the google maps interface that was discussed in Section 2.2 will be displayed. This display can be scrolled over, and centered at the user's location. The user may press the "x" in order to return to the main user interface.

General Use Case:

1. User will install and open the EZRider application, granting permission to all prompts.

2. User will place their iPhone in their rigid dashboard mount, ensuring that the rear-facing camera is viewing the road in front of the vehicle.

3. Once ready to drive, the user will begin collecting data by pressing "Record". The user will drive to their desired location in a normal manner while the application automatically records data.

4. Once the user is at their destination, or simply wants to stop recording, they will park their vehicle and press "Stop Recording". Now, they will store this batch of data on their iPhone in order to get ready to upload it to the cloud.

5. The user will upload their newly collected data to our cloud storage by pressing the "Upload" button. Depending on how large the files are, this may take a moment. In a future release, a progress bar will be included here. For now, please just wait for a couple of minutes if you have an especially large amount of data.

6. Once the upload is complete, the user will delete the data that is currently stored locally by pressing "Delete". This ensures that the data being uploaded to the cloud storage is fresh, and that the application does not take up too much space on the user's device.

7. The user may view locations of obtained rough roads by pressing the "Display Map" button.

8. If the user accidentally collects data or collects data that the user determines to be low-quality, the "Delete" button will be pressed in order to clear it from the iPhone's local storage and ensure that it will not be uploaded.

3.2 Operating Mode 2: Abnormal Operations

A probable issue that may present itself when operating our system can be in regards to large file sizes and memory or storage issues. It is recommended that you frequently manage your application data by frequently uploading your data points and deleting them afterwards. It is not recommended to let your stored files to build for a prolonged period of time, as this could cause instability within the application and buggy or undesired results.

3.3 Safety Issues

The most prominent safety concern in regards to the EZRider system is the use of a mobile device while operating a motor vehicle. For this reason, we ask to please only interact with the interface while pulled over or parked. Regarding data safety, we have ensured user anonymity through a handful of measures: No user login, and randomly-named, aggregated data uploads. This ensures that no data points are associated with a single user. Our algorithm that uploads collected data labels it only with the timestamp of time-of-upload, and no other user related data. If user login and other features are added in the future that may complicate issues in regards to data safety, we will continuously look to update and improve this important aspect of the system.

4 Technical Background

Our system consists of three main components: the frontend Flutter mobile application, the BaaS software integrated with Google Cloud, and the models (machine learning and statistical) running on the backend.

Mobile Application Development

We design the EZRider mobile application following a user centric approach. Our goal was to build a user friendly interface that could allow us to perform two distinct tasks; data collection and analysis. We decided to use the onboard sensors available on all modern smartphones to collect accelerometer, gps and image data. Following this step, we analyze the data collected to reflect the adequate road roughness metric.

In order to appeal to a broader user base, we had to develop the mobile using a cross platform framework. That is why we chose Flutter, an open source framework created by Google that allows us to build a mobile application and deploy it to different smartphone operating systems (Android, iOS) using a single code base. The Flutter application was designed using BLoC to separate the user frontend from the backend business logic needed to operate the application. This approach facilitates the implementation of new features while keeping the code structure organized.

Cloud Storage

To go along with Flutter, we chose Firebase as our cloud backend to store the collected data. The three data streams we collect are anonymized such that each user data collection session is given an unidentifiable random string of characters. The data is stored using key, value pair approach, for each image taken, we store a JSON map containing x, y, z accelerometer data and latitude, longitude gps location.

Mathematical and Machine Learning Models

Data Collection:

The user fixes a smartphone mount on the dashboard of the vehicle. The smartphone is placed on the mount (landscape mode) while keeping our application open. The data collection process is initiated according to the steps outlined in the previous sections. Our application collects accelerometer data points, GPS coordinates, and image data at a given sampling rate. The data is stored locally and then uploaded to Firebase storage. The accelerometer data points (accelerometer values in the x, y, and z directions) and the GPS coordinates (latitude and longitude) are stored in Firebase as JSON files.

Models Summary:

The accelerometer data points are used to train a simple mathematical model called the Z peak algorithm with iterative threshold selection. Every element in the JSON file comes with 5 values: accelerometer values in the x, y, and z directions (3) and GPS coordinates (2). This makes it easy to tag a GPS location with an accelerometer data point. The predicted rough road locations are found using the GPS coordinates of the accelerometer data points classified as anomalies by the Z peak model.

The Inception V3 neural network is trained on a pothole dataset found online. Then, we conducted model inferencing (testing after deployment) using the deep learning model on the images collected through the application. It should be noted that the neural network is not trained on the collected image data.

Z Peak Algorithm:

We collected enough accelerometer data points (around 3000) and images by driving around Boston. The data is stored on Firebase. We created a separate python notebook to implement the model for rough road detections. The collected data was imported from Firebase to Google Drive using a module called Pyrebase. We combined multiple JSON files into one large Pandas dataframe.

We plotted the accelerometer points on a 3D graph, plotted the x-z accelerometer values, and graphed the accelerometer values in the x direction. Since the phone was kept in landscape mode while collecting data, the accelerometer values in the x direction would change the most instead of the values in the z direction.

We decided to use the Z Peak algorithm to predict rough road locations using accelerometer data points. The threshold value for the Z Peak algorithm was decided using the iterative selection threshold method. The Z Peak algorithm identifies rough road locations by checking if the x direction accelerometer values exceed the threshold value or not. The iterative selection method for finding the threshold is based on identifying peak points (maxima) in the x-axis accelerometer values. The median of the peak points is calculated and set as the initial threshold. The dataset of peak points is split into two sets depending on whether each point exceeds the initial threshold or not. Now,

the new threshold is set as the average of the means of the two separate datasets. The process converges when the new threshold and old threshold only differ by a certain amount. The predicted rough road locations are saved as a dataframe and uploaded to Firebase storage as a JSON file.

Inception V3 Binary Classifier:

The Inception V3 neural network is pre-trained on the ImageNet dataset. We used this model for classifying images as having rough roads (positive) or not (negative).

The following two datasets were used for training. The two sets were concatenated to create a larger dataset. The combined data set contains 10,662 training images, 1184 validation images, and 984 testing images. A few samples from the training set are shown below. Both the datasets were generated by the same person for a paper. The images were collected by mounting a camera on the vehicle's dashboard. The images in the online dataset resemble the images taken through our application. Hence, the model will be able to generalize well when testing on the images collected through the application. The images (RGB) are rescaled (480x640) using the image_dataset_from_directory function offered by Keras.

1. https://www.kaggle.com/datasets/felipemuller5/nienaber-potholes-1-simplex
2. https://www.kaggle.com/datasets/felipemuller5/nienaber-potholes-2-complex

Negative sample from the training set

Positive sample from the training set

We downloaded the dataset from Kaggle to Google Drive through Kaggle's API and a JSON file containing the username and key. We imported the dataset to a Google Colab notebook using the image_dataset_from_directory function offered by Keras. The dataset was split into training, validation, and test sets. The Inception V3 pre-trained model was imported from Keras without including the top layer. We added a set of new layers on top of the imported base model. We trained the net for 10 epochs (after freezing imported base layers and only changing the parameters of the newly added layers) with a large learning rate. Then, we trained the net for 15 epochs (after unfreezing all layers and changing parameters of both the base layers and newly added layers) with a small

learning rate. We used the binary cross entropy loss and the sparse categorical cross entropy loss with the Adam optimizer.

The neural network gave an accuracy of 91% with the binary cross entropy loss and a test accuracy of 90% with the sparse categorical cross entropy loss.

5 Relevant Engineering Standards

ISO/IEC/IEEE 12207: Software life cycle processes:

- An international standard for software lifecycle processes. First introduced in 1995, it aims to be a primary standard that defines all the processes required for developing and maintaining software systems, including the outcomes and/or activities of each process.
- This standard can provide guidance on how to design and develop our application in a systematic and disciplined way.

ISO/IEC 27001: Information security management systems

- A standard that specifies the requirements for an information security management system (ISMS), including the development of security policies, procedures, and controls.
- This would allow us to ensure the security of our firebase cloud storage bucket.

ISO/IEC 29183: Method for measuring digital copying productivity for a single one-sided original

- A standard that provides guidance on privacy issues related to mobile applications, including how to protect personal data and ensure user privacy.
- This would provide guidance on how to ensure the privacy of user data.

IEEE 802.11: Wireless Local Area Networks (WLANs)

- A set of local area network (LAN) technical standards, and specifies the set of media access control (MAC) and physical layer (PHY) protocols for implementing wireless local area network (WLAN) computer communication.
- These standards and protocols can be used to ensure secure data transmission from users to the cloud, enabling user data to be kept private.

Apple Human Interface Guidelines

- Contains guidance and best practices that can help you design a great experience for any Apple platform.
- As one of our priorities is a user friendly environment, these guidelines and practices will help ensure our app is intuitive and easy to use for all.

OWASP Mobile Top 10

- A list of the top 10 security risks for mobile applications published by the Open Web Application Security Project (OWASP).
- Will allow us to identify and preemptively mitigate potential security risks towards our application.

6 Cost Breakdown

Project Costs for Production of Beta Version (Next Unit after Prototype)				
Item	Quantity	Description	Unit Cost	Extended Cost
1	12	Mobile phone stand	$14.99	$179.88
2	4.5	Additional GB of storage	$0.18	$0.81
3	1	Document writes (per 100,000 documents)	$0.18	$0.18
4	0	Document reads (per 100,000 documents)	$0.06	$0
5	1	Document deletes (per 100,000 documents)	$0.02	$0.02
			Beta Version-Total Cost	$179.88 + $1.01/day

Our application usage is based upon the assumption that users will already possess their own mobile cellular device and have access to a stable internet connection. Thus, those two requirements will not be included within the cost breakdown.

Additionally, quotas for our firebase storage are charged daily, so everything besides the mobile phone stand is a recurring charge. An initial test of the application revealed that in 23 minutes, roughly 174 MBs of data was collected. Thus, on the assumption that we would collect roughly 12 hours of data per day collectively via all of our drivers (12 drivers with an average commute time of about one hour round trip to work in Massachusetts) , the total comes out to approximately 5.5 GBs of storage needed per day. At the end of each day, the images and data will be fed into our machine learning model and subsequently deleted. The test also revealed that 174 MB of collected data equates to 927 image documents and 1 data document. As such, given that we anticipate 5.5 GBs of data being collected per day, that means that we will accrue roughly 30,000 documents per day. In total, that equates to 30k writes to the cloud, 30k reads to the model, and finally 30k instances of deleting each document.

Firebase, for no additional charge, provides us with 1 GB of storage per day, 50,000 document reads per day, and 20,000 document writes and deletes per day. Thus, we can subtract these complementary quotas from our totals, and the results are listed in the table above.

7 Appendices

7.1 Appendix A - Specifications

Team 27
Team and Project Name: EZRider

Requirements	**Value, range, tolerance, units**
Location Accuracy	Accurate to 5m radius around iPhone
Accelerometer Accuracy	Accurate to within 1% error
Binary Classification	Inception V3 neural network classifies images collected through EZRider application with >90% accuracy.
Predicting Rough Road Locations	● The Z peak algorithm with iterative threshold selection predicts rough road locations from accelerometer data points with >80% accuracy. ● The accelerometer data points are unlabeled (we do not know if the data point corresponds to a rough road location). Since we do not have true labels, it is hard to determine the accuracy of the model. The tentative accuracy was calculated by manually inspecting the rough roads predicted by the algorithm.
Cost	The overall cost of our final EZRider system is less than 200 dollars plus additional storage fees.

7.2 Appendix B – Team Information

Balaji Sathyanarayanan is a graduating Computer Engineering student with a concentration in machine learning. He hopes to leverage his knowledge and skills in developing reliable deep learning systems aimed at solving world's leading challenges. He will pursue a machine learning internship at Carnegie Mellon University.

Luke Bacopoulos is a graduating Computer Engineering student with knowledge in both hardware and software. He is from Western Massachusetts, where he grew up spending much of his time working at his family's restaurant. He is currently pursuing a career after college in which he can combine the knowledge gained from engineering school with the practical experience gained through many different areas of work.

Aymane El Jerari is a Computer Engineering student interested in the overlap between computer architecture and artificial intelligence. He will be joining the University of Toronto Master's program in Electrical and Computer Engineering before pursuing a career in the AI hardware acceleration industry.

Jason Wang is a graduating Computer Engineering student. with experience in multiple high and low level programming languages. He hopes to pursue a career which allows him to utilize the skills he has gained from his four years in school and apply them to any presented problem.

Haoming Yi is a computer engineering student interested in software engineering and practical applications of machine learning. He hopes to gain more hands-on experience through his upcoming software development internship at a tech start-up in Palo Alto. He will start pursuing a masters degree in electrical and computer engineering at Rice University this September.