Smart Cricket Bat

Gavin Dahl, Jiakai Hu, Pablo Barron, Nolapat Pipitvitayakul

FULL SYSTEM REPORT

Draft – 1
20 November 2022

SYSTEM REPORT
FOR
Smart Cricket Bat

PREPARED BY:

_____
Author Date

APPROVED BY:

_____
Project Leader Date

_____
John Lusher II, P.E. Date

_____
T/A Date

Change Record

Rev.	Date	Originator	Approvals	Description
1	11/20/2022	Gavin Dahl		First Draft

Table of Contents

List of Figures

1.1. Introduction

The smart cricket bat will acquire data during the user's swing and give feedback to the user on how to further improve their swing. The system gathers data through the inertial measurement unit mounted at the handle of the bat, where it is transmitted to the consumer app via a microcontroller with a Bluetooth module. The data transferred to the app is then uploaded to the machine learning algorithm to be processed and then returns to the user the characteristics of the swing and what can be done to improve. The system is broken down into the power, control, app, and machine learning subsystems, each of which was designed and rigorously tested. Since each subsystem was validated to be working correctly and fulfilling all requirements, there is a clear path to integration for these subsystems into the full system specified in the Conops, FSR, and ICD.

1.2. System Details

The system as a whole has three main components, the actual sensing device that attaches to the bat handle, the user interface application, and the machine learning algorithm in the cloud. The first component consists of a PCB that holds an ATmega328p microcontroller that interfaces with two main modules, the MPU-6050 inertial measurement unit and the HC-05 Bluetooth module, and the charging circuit that connects to and charges the 3.7V 500 mAh Li-Po battery. This PCB and the corresponding Li-Po battery are what make up the sensing unit. The sensing unit uses its bluetooth to connect to any device available, which in this case will be the user's phone, which acts as the middle man between the data measurement device and the machine learning algorithm. The important data that the sensing device will deliver to the phone and whether it gets sent to the machine learning depends on what the user requests, there are 4 options available to them: connecting to the device, getting battery life of the device, calibrating the device, and starting or stopping the sending of IMU data. When sending data to the machine learning, the MCU sends 7 data points that the machine learning needs to properly derive the characteristics of the swing, those points being the time the current data was obtained at, the 3 values for the 3 axes of the gyroscope, and the 3 values for the 3 axes of the accelerometer. The data is sent from the application to the cloud as a compressed json file to then be processed (as of now this is done manually through a few simple terminal commands in the AWS database, however in future iterations the goal is to have quick automatic processing). First, the machine learning algorithm does preprocessing to the data collected with normalizing, chunking, FFT, PSD, peak energy detection, total energy calculation, and angular impulse calculation. Through this process, the 7 data points generated will transform into 129 features for each data collected. Then, these features will be used in the ML model training with the same process in the AWS server and produce the output with the predicted hit region, swing speed, and angles as a json file. Then, the json file is sent to the application for the user to view and interact with. The application has a home screen with buttons that will lead the user to a page that will have the output value from the machine learning that corresponds to said button. For example, if the user wants to see the swing speed and tilt after their swing, they simply touch the “speed/tilt” button and are taken to the page that will then display the speed and tilt of the bat (along the z-axis) that the machine learning algorithm calculated in meters per second and degrees, respectively.

1.2. System Validation

Hit Location 15 Data

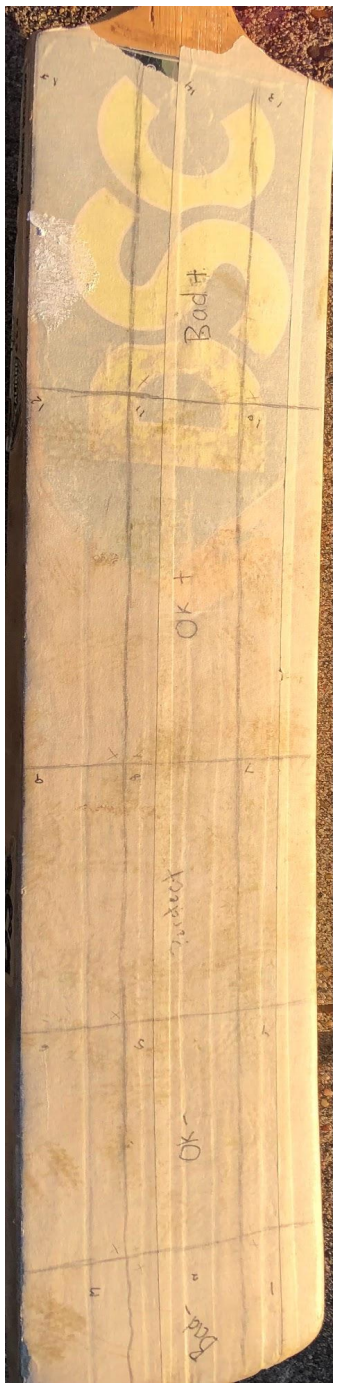


Figure 1a: Marked Physical Collision on Hit Location 15

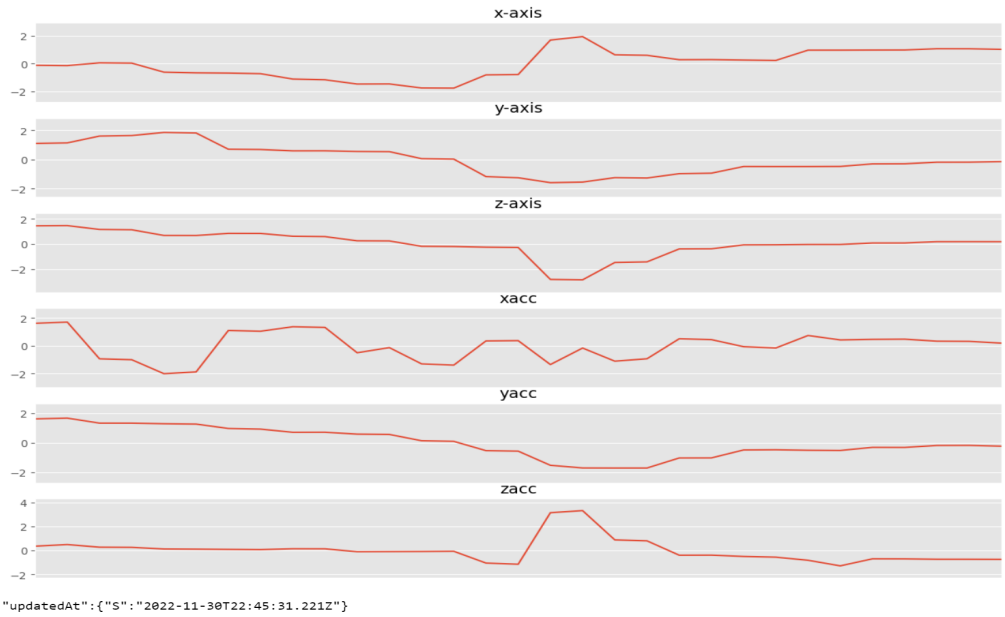


Figure 1b: Hit Data Plots After Preprocessing of Machine Learning

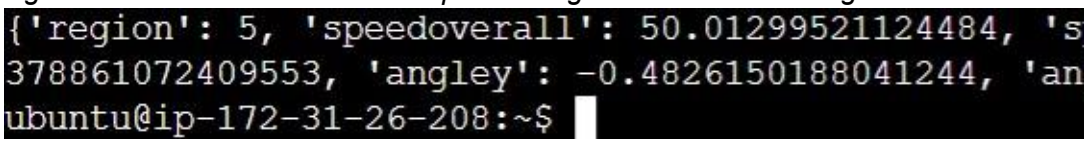


Figure 1c: Screenshot of Output Hit Location from Machine Learning

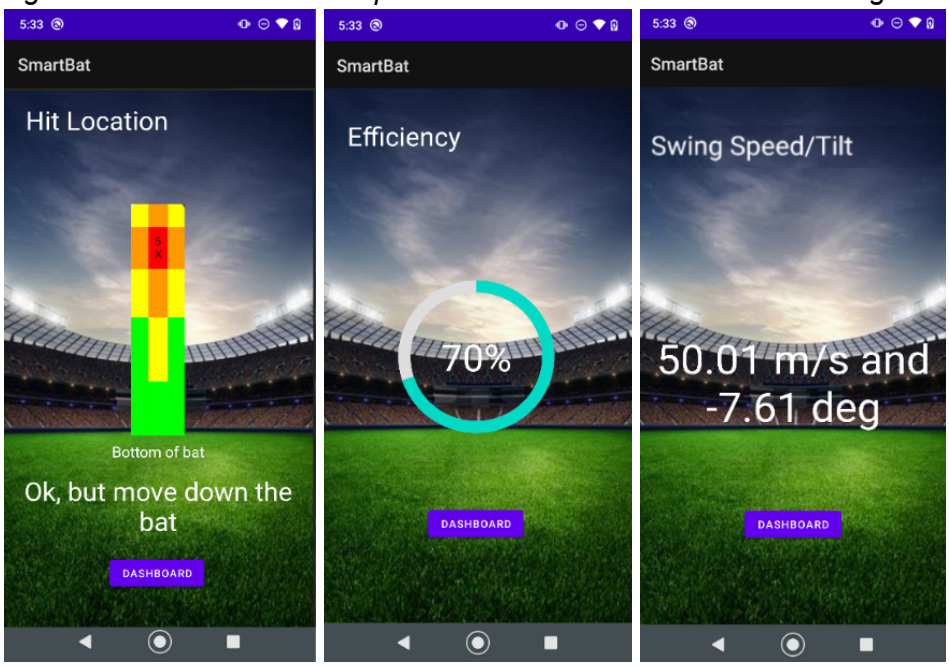


Figure 1d: Output shown in App

Hit Location 14 Data

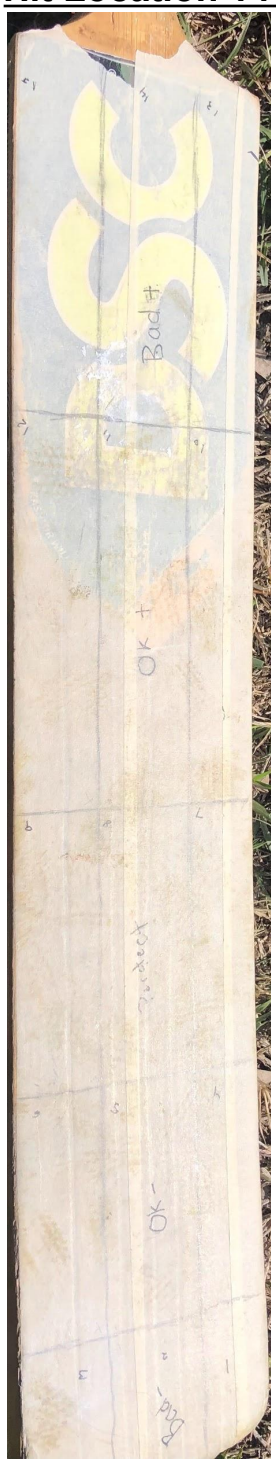
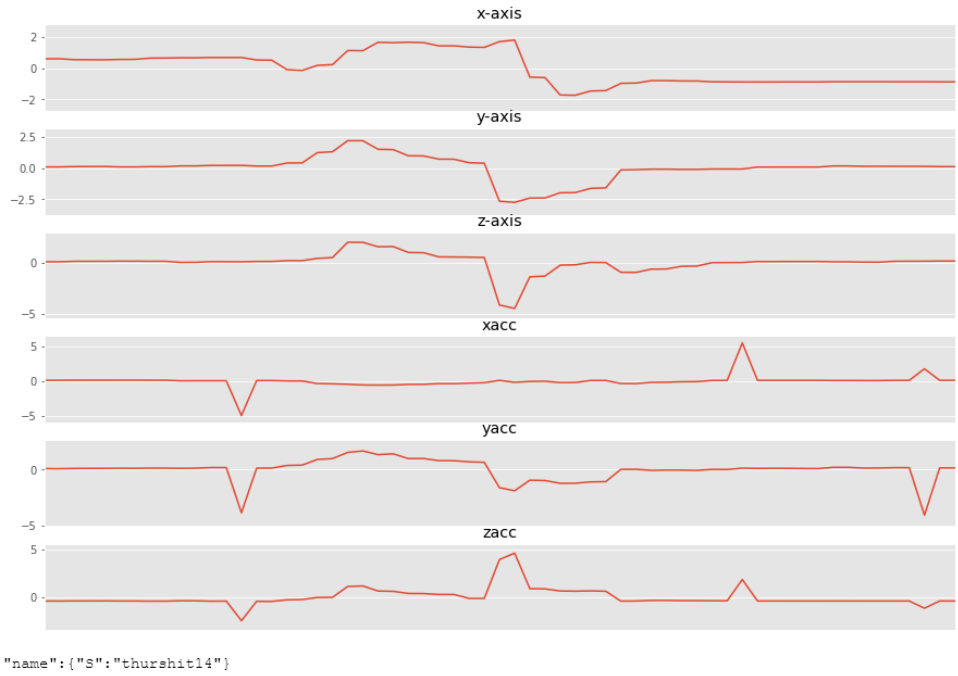


Figure 2a: Marked Physical Collision on Hit Location 14



"name":{"S":"thurshit14"}

Figure 2b: Hit Data Plots After Preprocessing of Machine Learning

{'region': 5, 'speedoverall': 89.96263917038004, 'speedimpactx': 0
458639434195035, 'angley': -0.5952277331718819, 'anglez': 0.0}
ubuntu@ip-172-31-26-208:~$

Figure 2c: Screenshot of Output Hit Location from Machine Learning

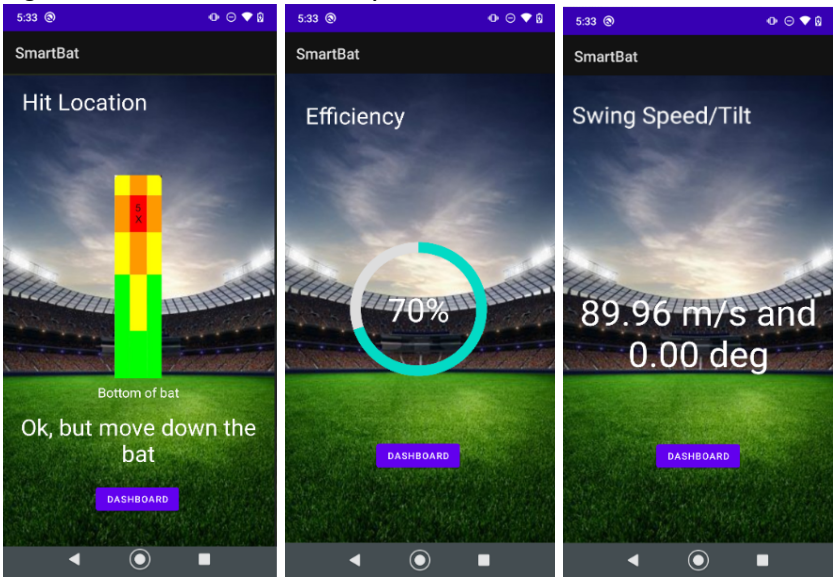


Figure 2d: Output shown in App

Hit Location 13 Data

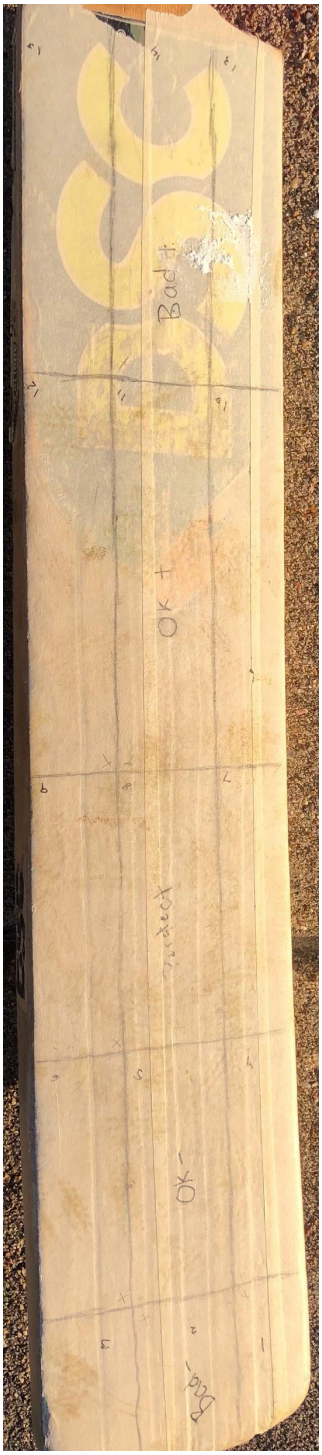


Figure 3a: Marked Physical Collision on Hit Location 13

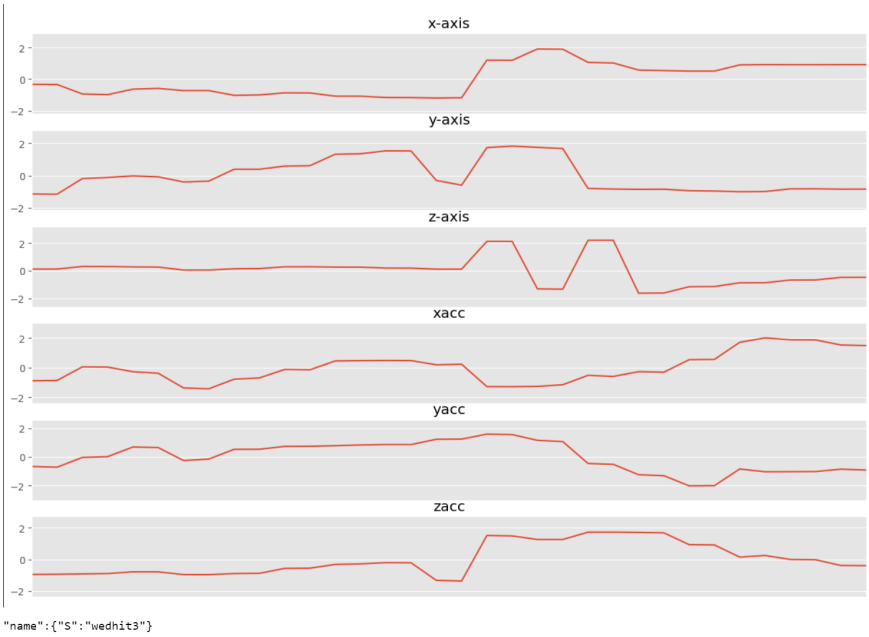


Figure 3b: Hit Data Plots After Preprocessing of Machine Learning

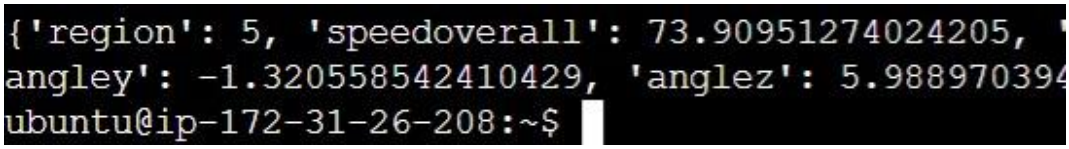


Figure 3c: Screenshot of Output Hit Location from Machine Learning

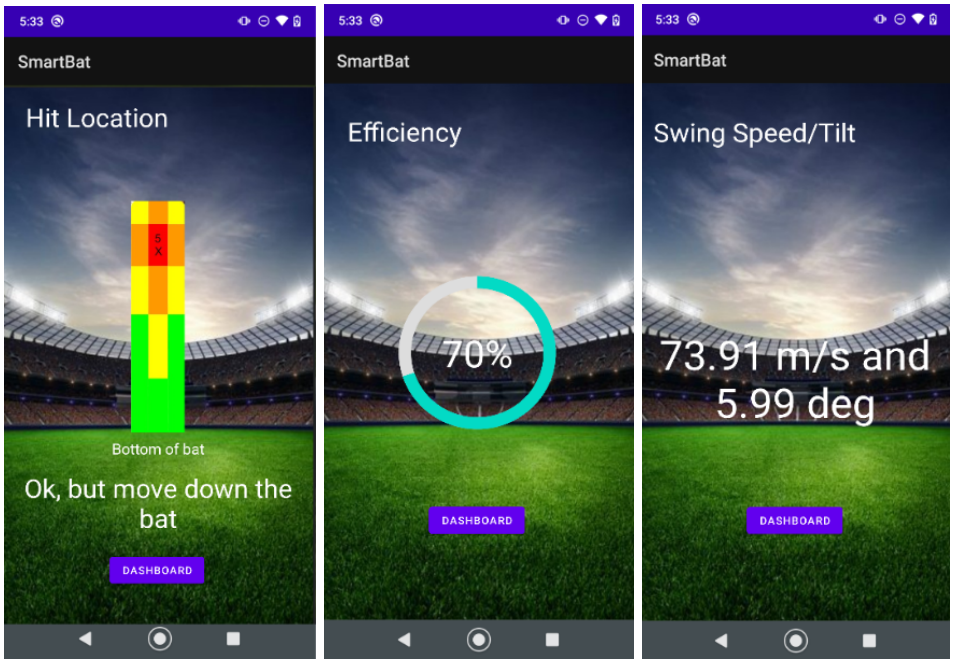


Figure 3d: Output shown in App

Hit Location 12 Data

Figure 4a: Marked Physical Collision on Hit Location 12

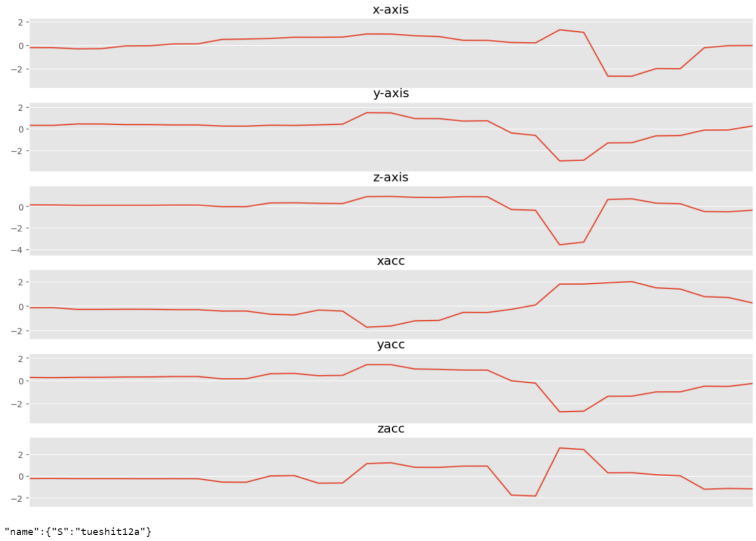


Figure 4b: Hit Data Plots After Preprocessing of Machine Learning

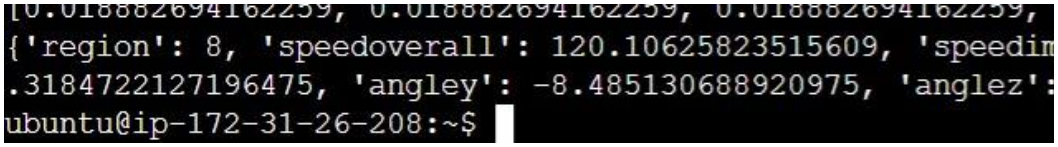


Figure 4c: Screenshot of Output Hit Location from Machine Learning

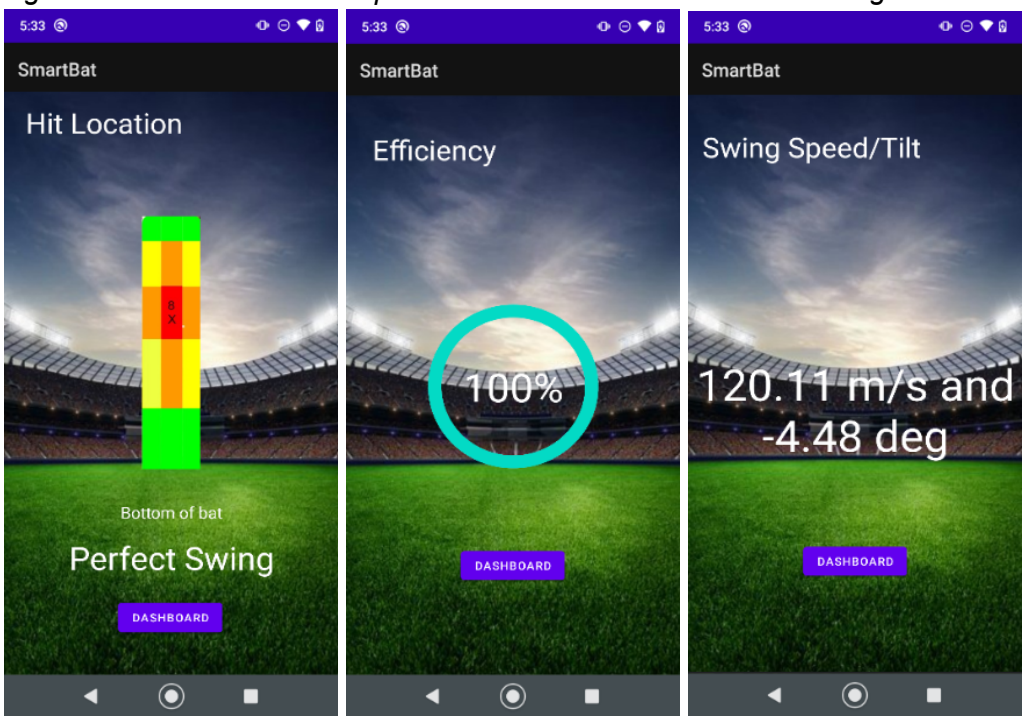


Figure 4d: Output shown in App

Hit Location 11 Data

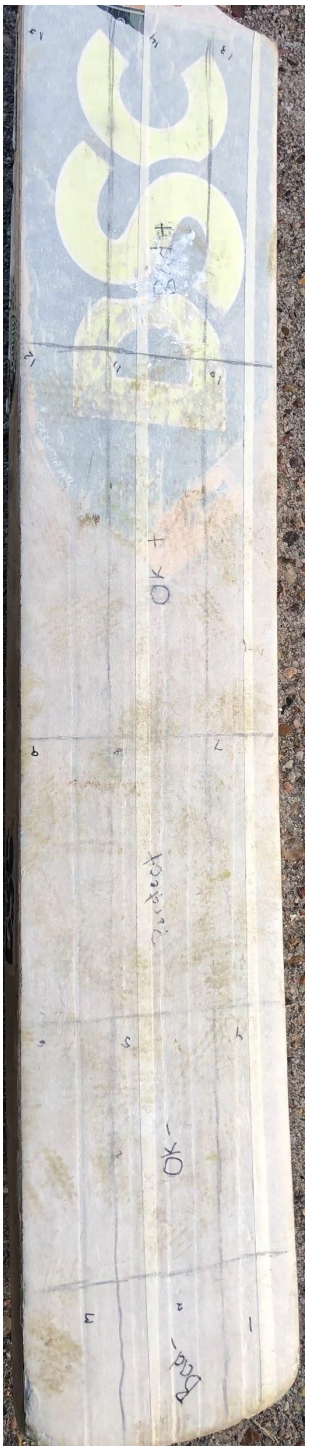


Figure 5a: Marked Physical Collision on Hit Location 11

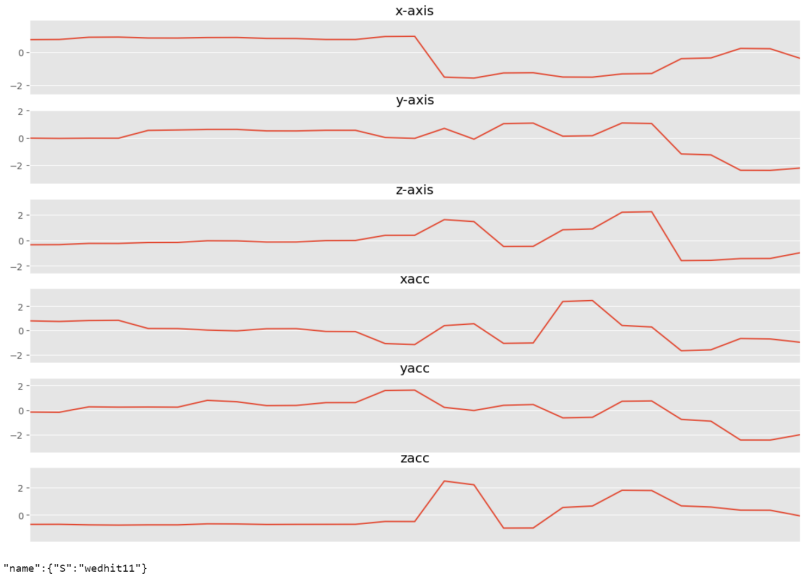


Figure 5b: Hit Data Plots After Preprocessing of Machine Learning

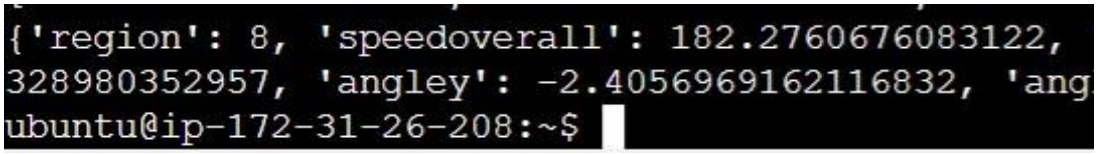


Figure 5c: Screenshot of Output Hit Location from Machine Learning

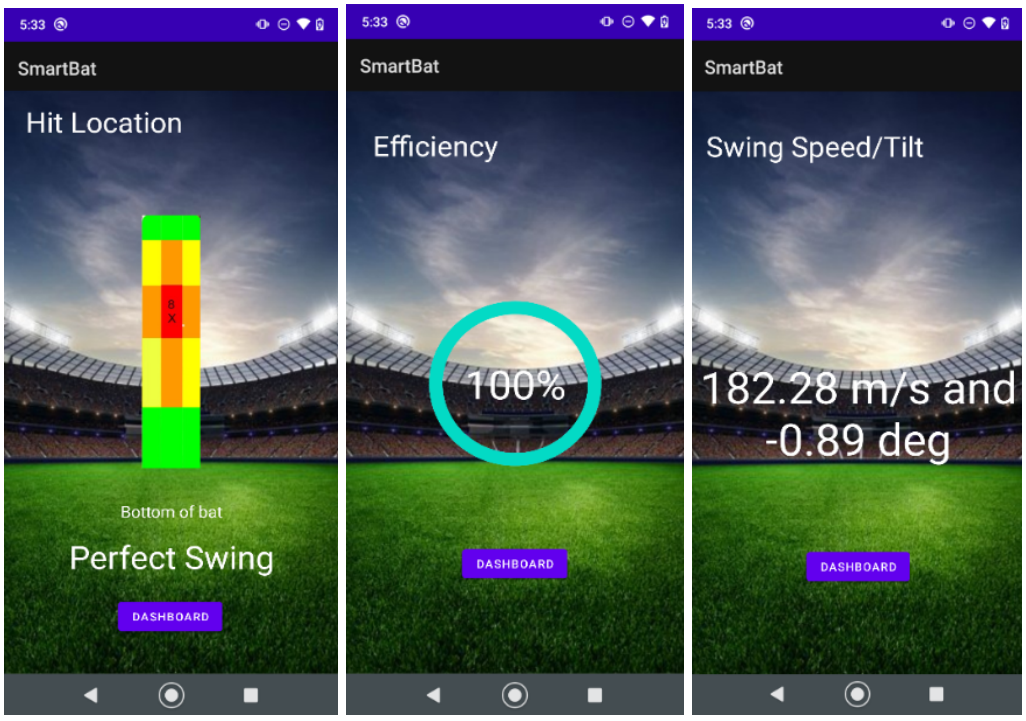


Figure 5d: Output shown in App

Hit Location 10 Data

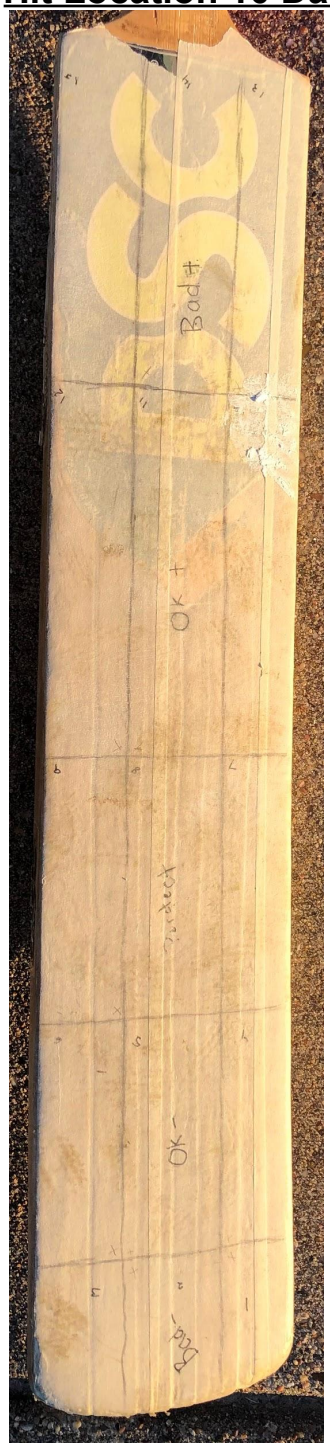


Figure 6a: Marked Physical Collision on Hit Location 10

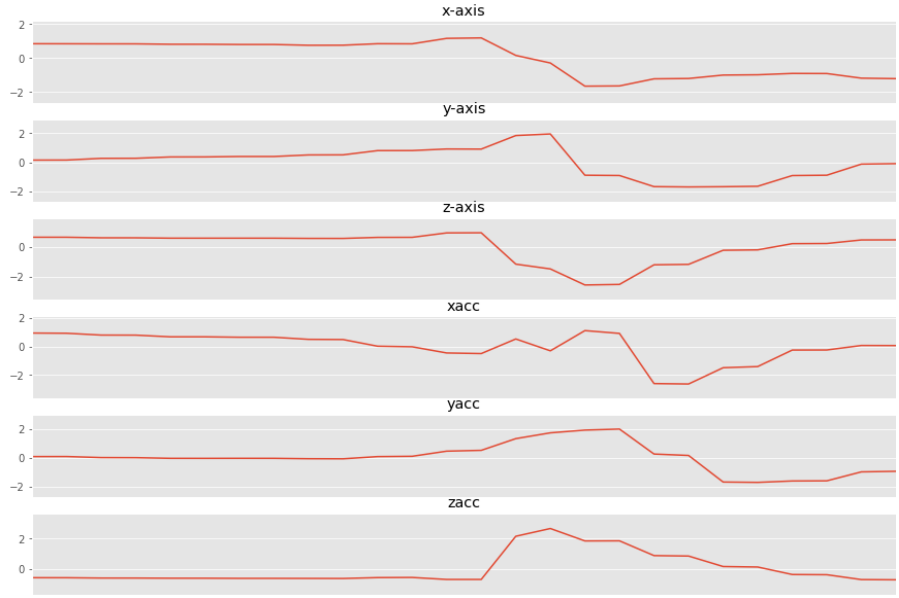


Figure 6b: Hit Data Plots After Preprocessing of Machine Learning

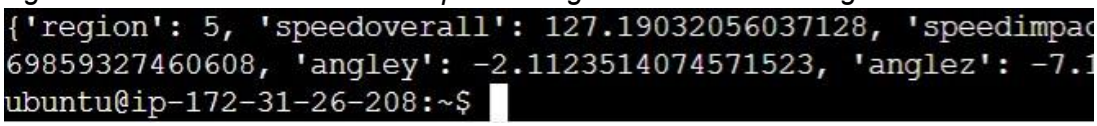


Figure 6c: Screenshot of Output Hit Location from Machine Learning

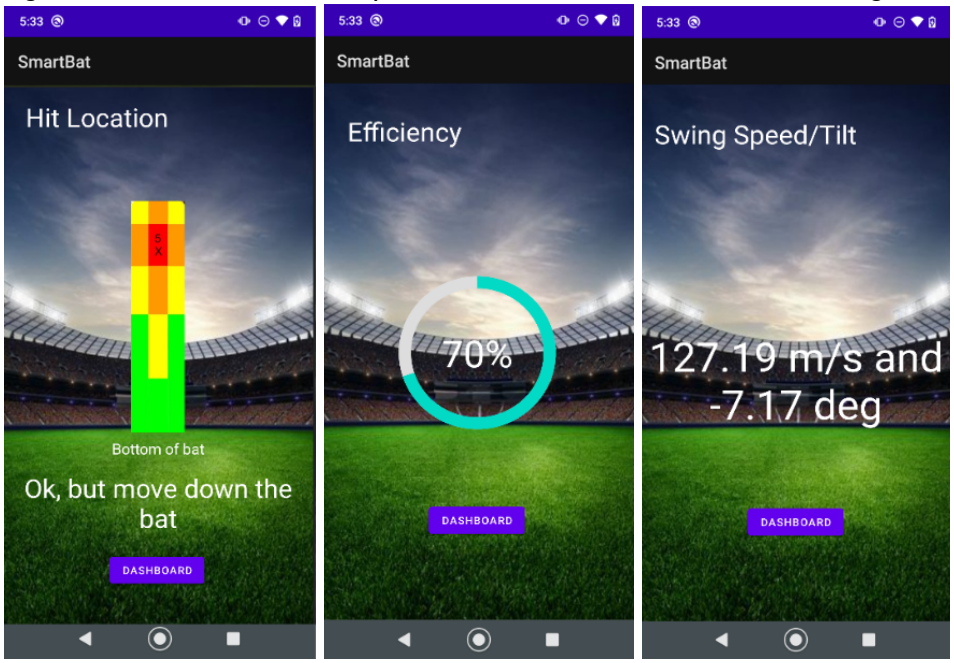


Figure 6d: Output shown in App

Hit Location 9 Data

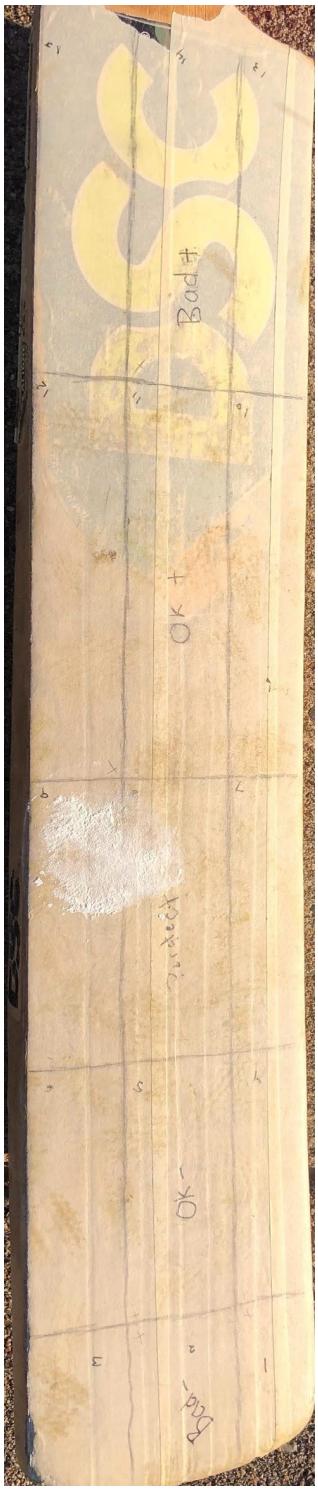
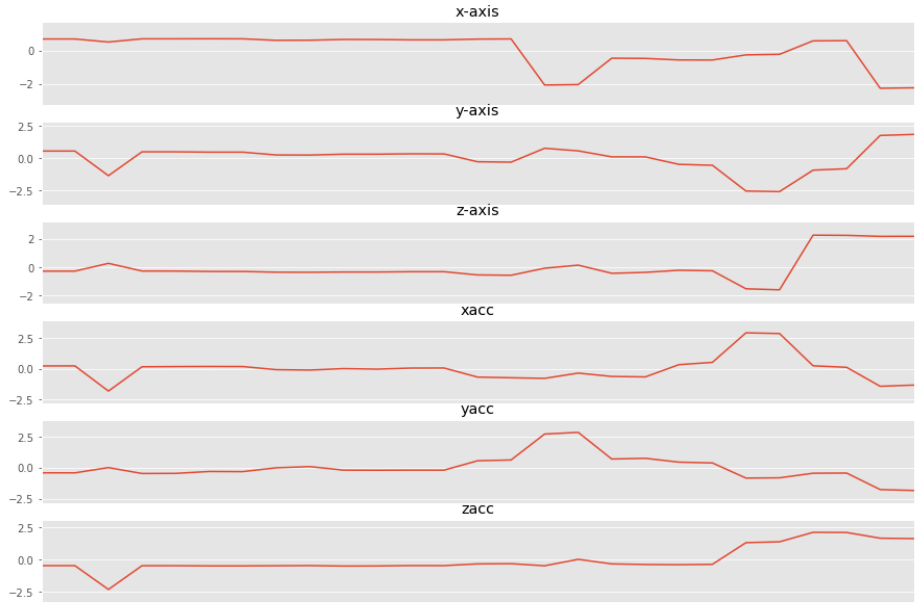


Figure 7a: Marked Physical Collision on Hit Location 9



"name":{"S":"testb"}

Figure 7b: Hit Data Plots After Preprocessing of Machine Learning

{'region': 5, 'speedoverall': 119.31773775093124, 'speedimpactx': 2.99762
4870757939777945, 'angley': -5.034752402260575, 'anglez': 3.6220936005870
ubuntu@ip-172-31-26-208:~$

Figure 7c: Screenshot of Output Hit Location from Machine Learning

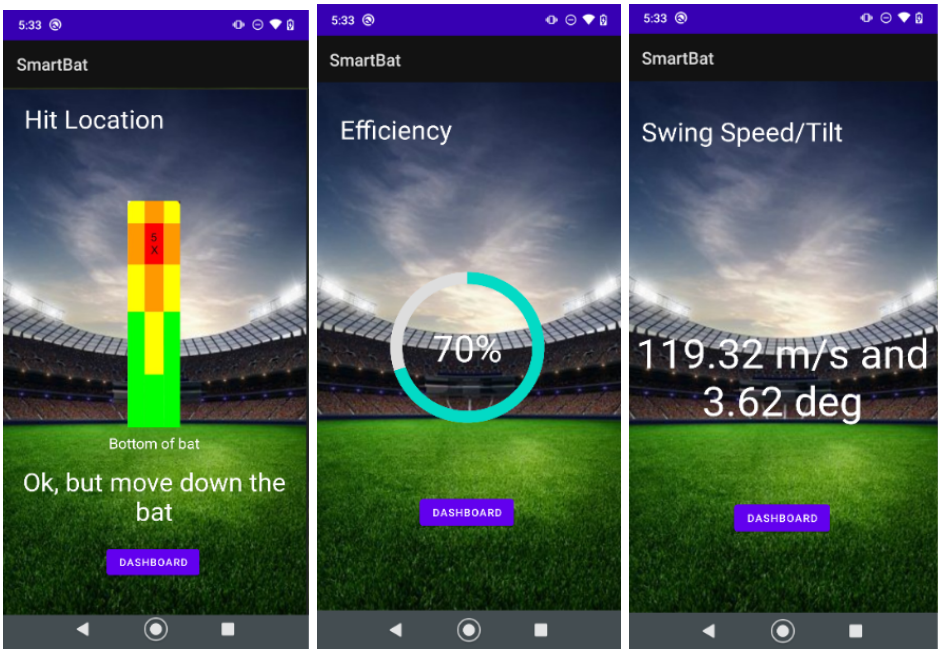


Figure 7d: Output shown in App

Hit Location 8 Data

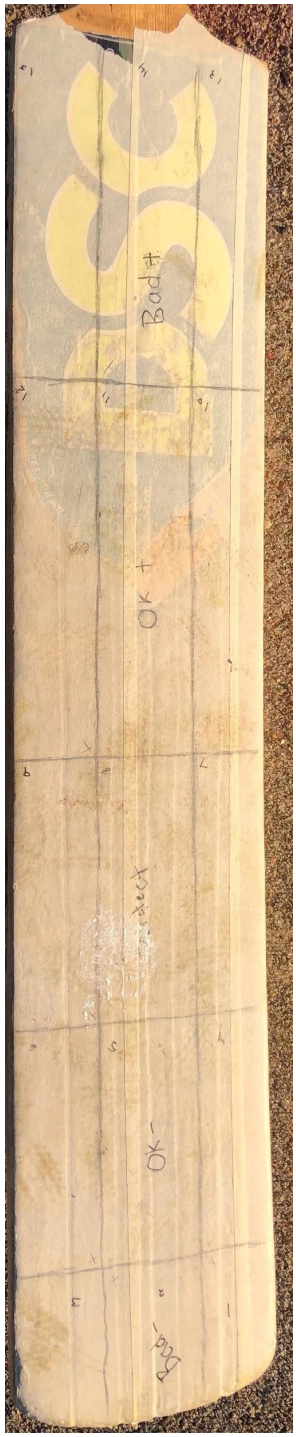


Figure 8a: Marked Physical Collision on Hit Location 8

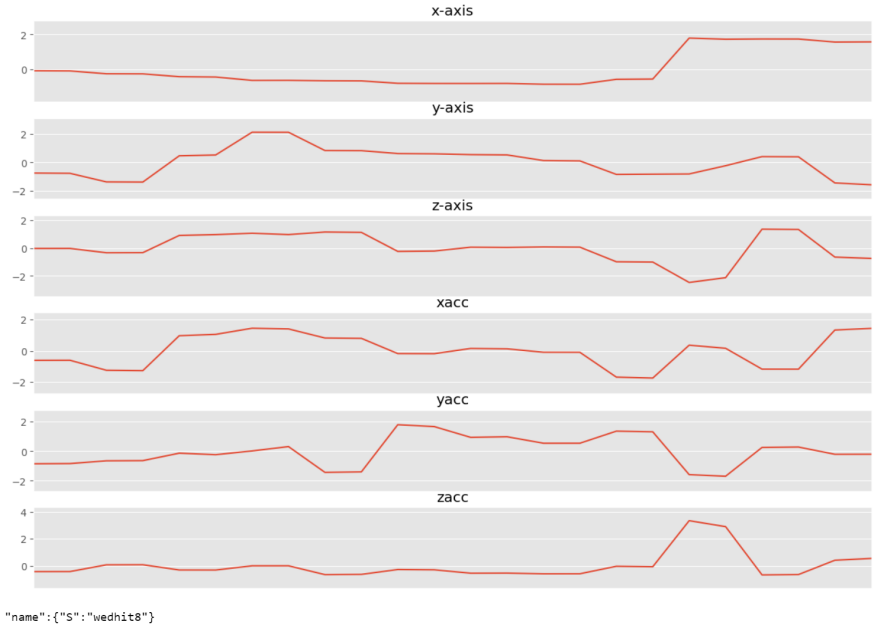


Figure 8b: Hit Data Plots After Preprocessing of Machine Learning

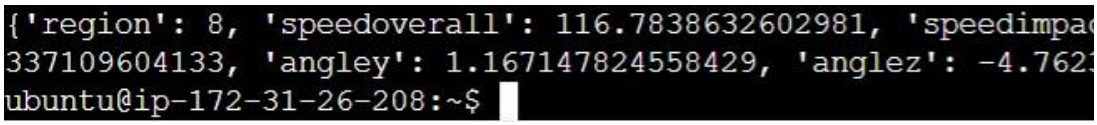


Figure 8c: Screenshot of Output Hit Location from Machine Learning

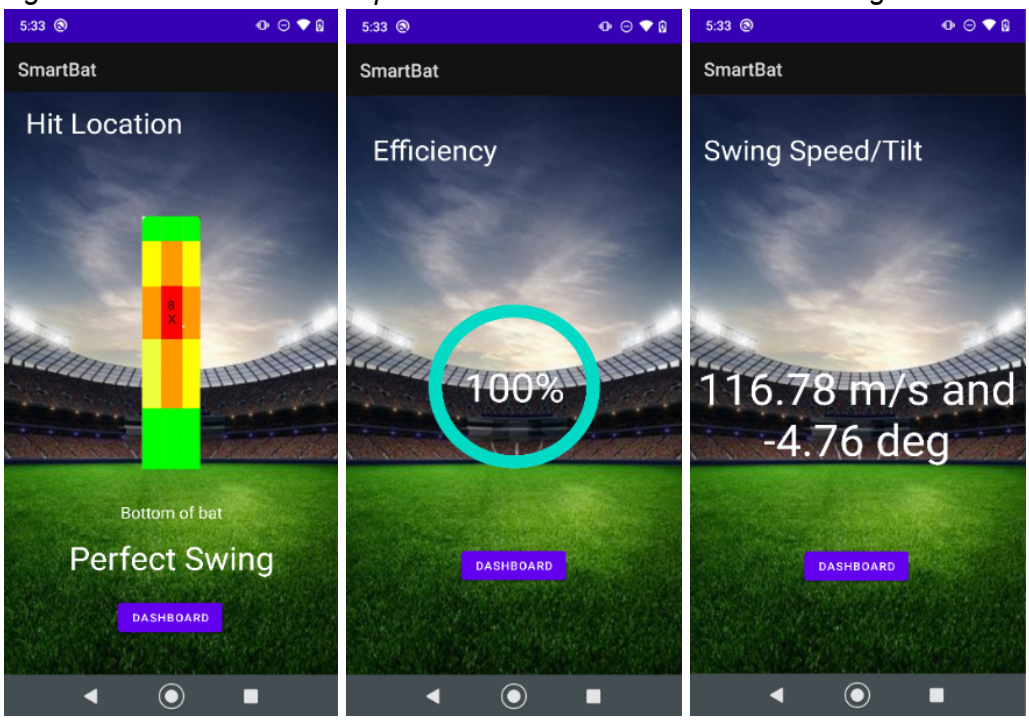


Figure 8d: Output shown in App

Hit Location 7 Data

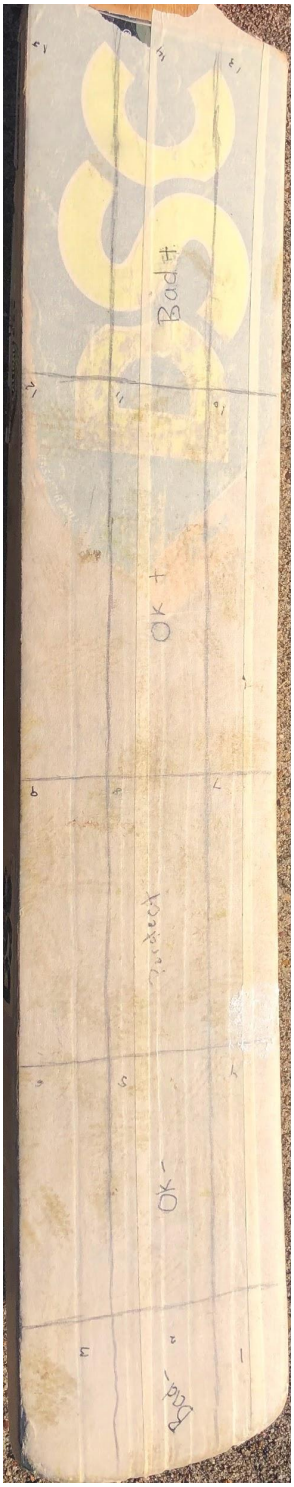


Figure 9a: Marked Physical Collision on Hit Location 7

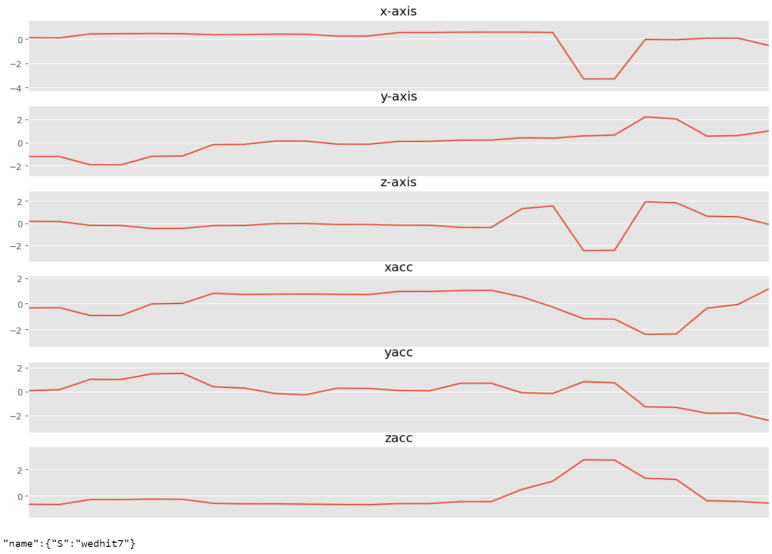


Figure 9b: Hit Data Plots After Preprocessing of Machine Learning

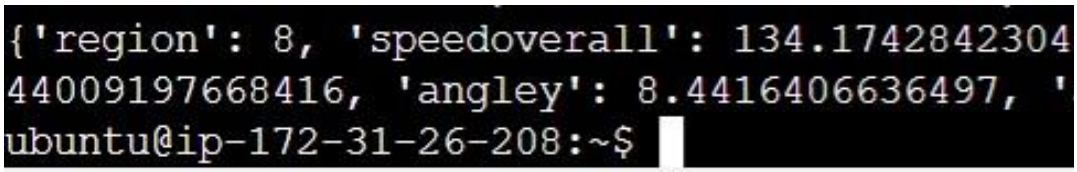


Figure 9c: Screenshot of Output Hit Location from Machine Learning

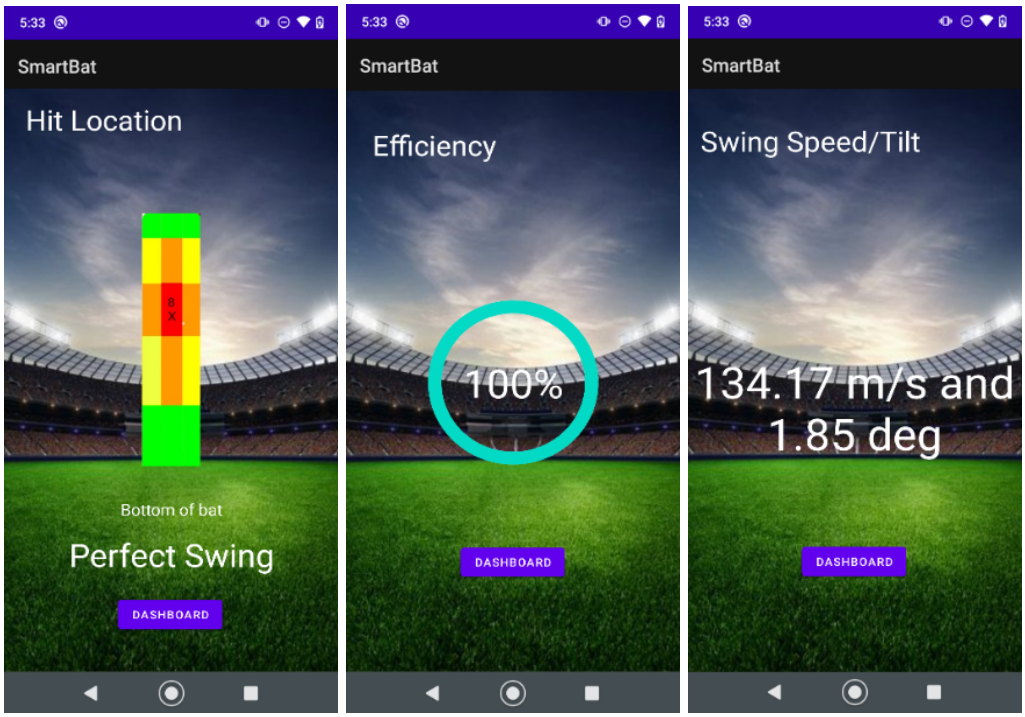


Figure 9d: Output shown in App

Hit Location 6 Data

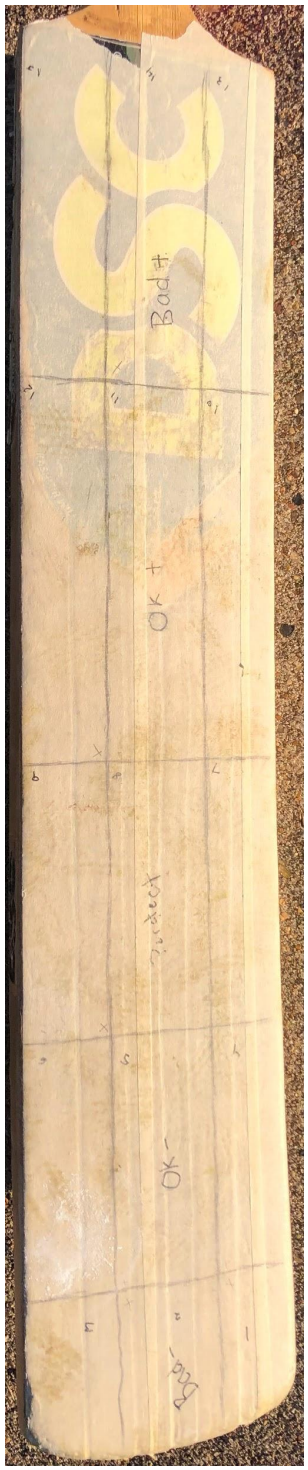
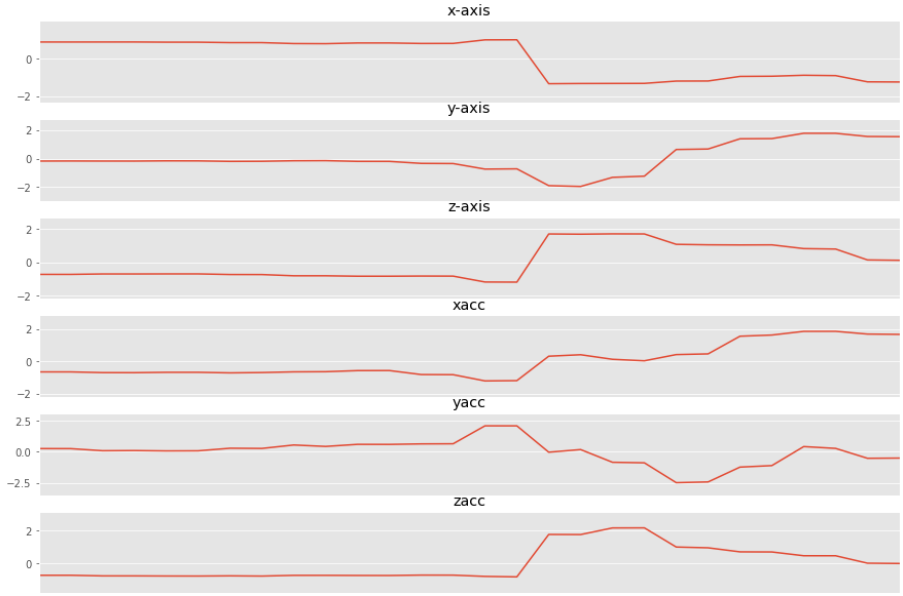


Figure 10a: Marked Physical Collision on Hit Location 6



"name":{"S":"thurshit6"}

Figure 10b: Hit Data Plots After Preprocessing of Machine Learning

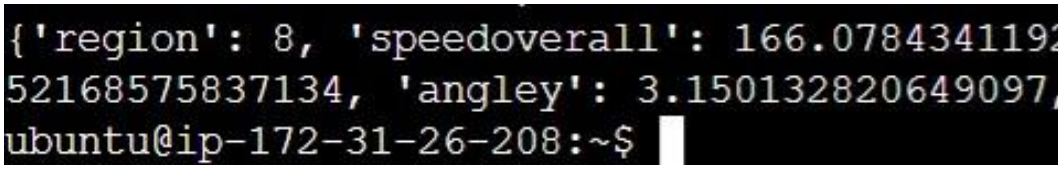


Figure 10c: Screenshot of Output Hit Location from Machine Learning

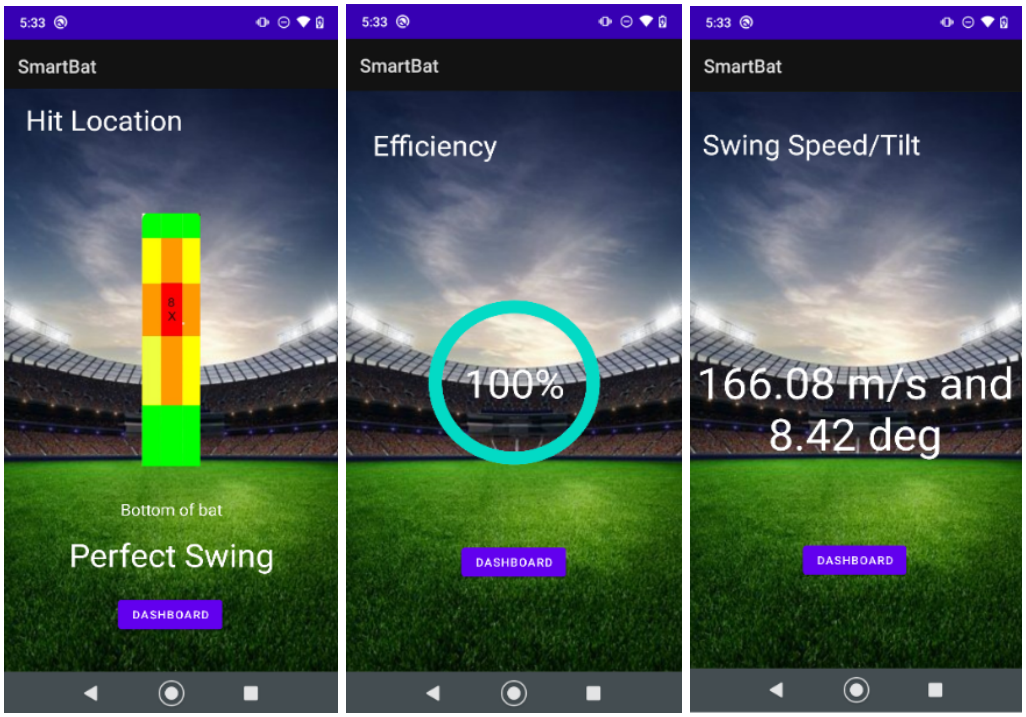


Figure 10d: Output shown in App

Hit Location 5 Data

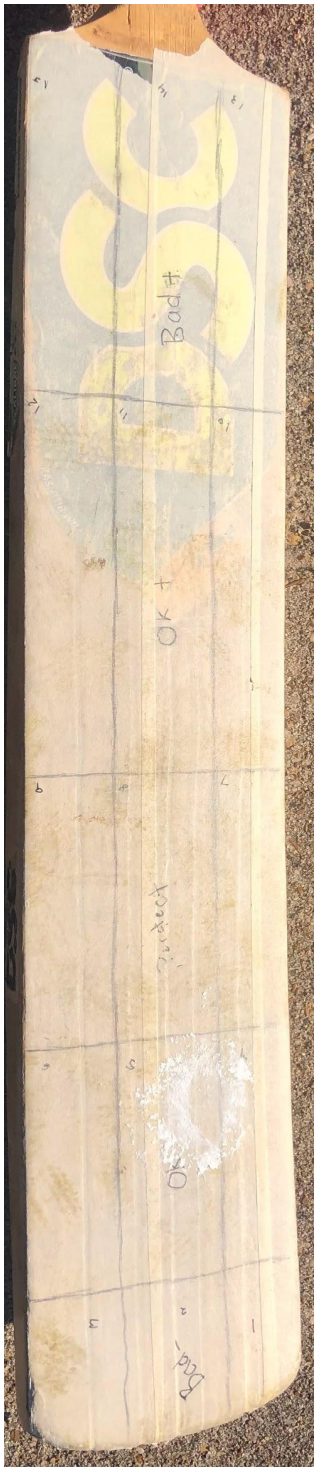


Figure 11a: Marked Physical Collision on Hit Location 5

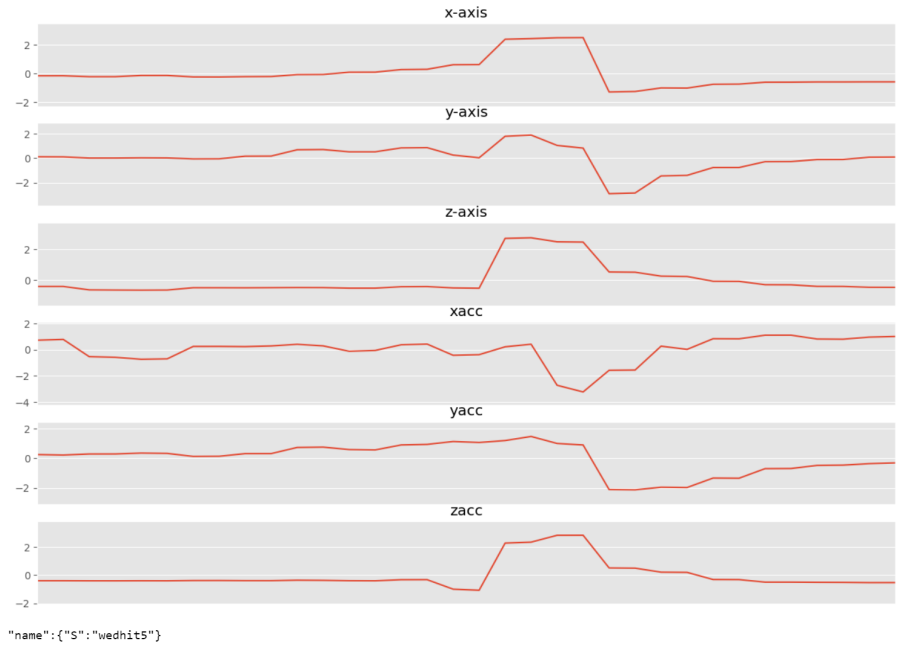


Figure 11b: Hit Data Plots After Preprocessing of Machine Learning

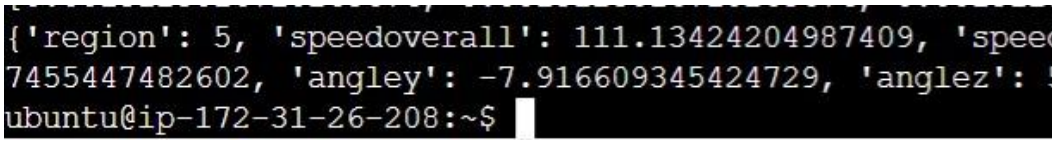


Figure 11c: Screenshot of Output Hit Location from Machine Learning

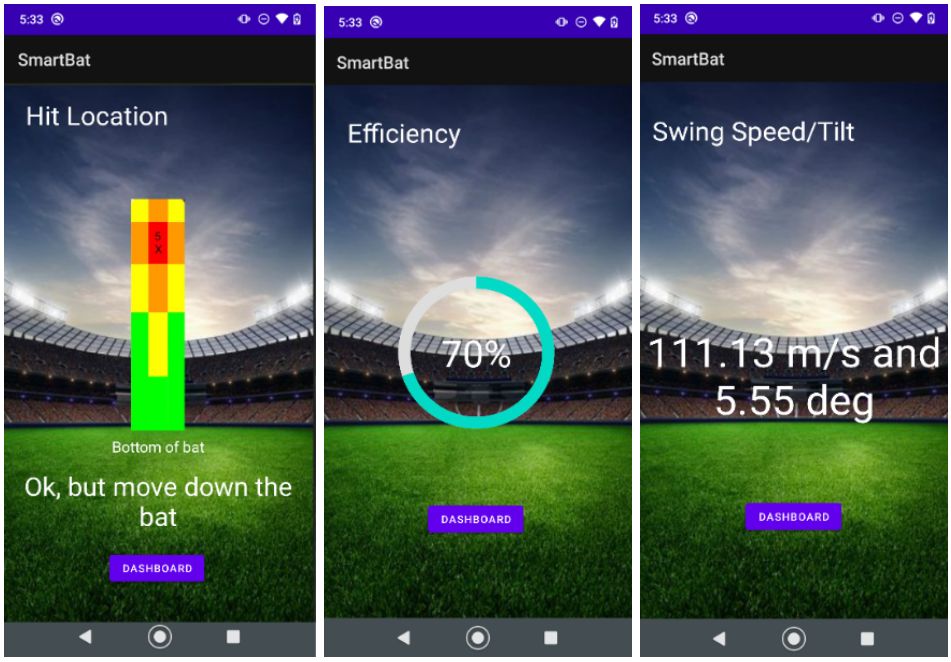


Figure 11d: Output shown in App

Hit Location 4 Data

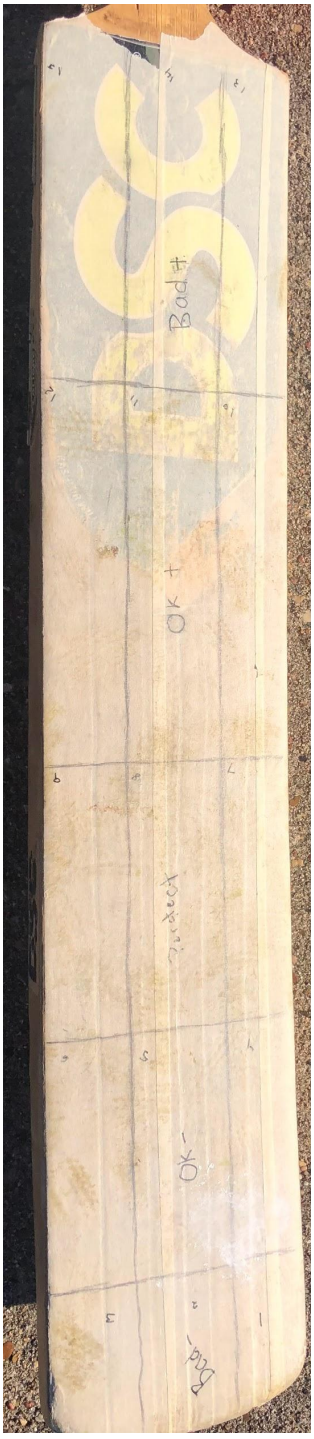


Figure 12a: Marked Physical Collision on Hit Location 4

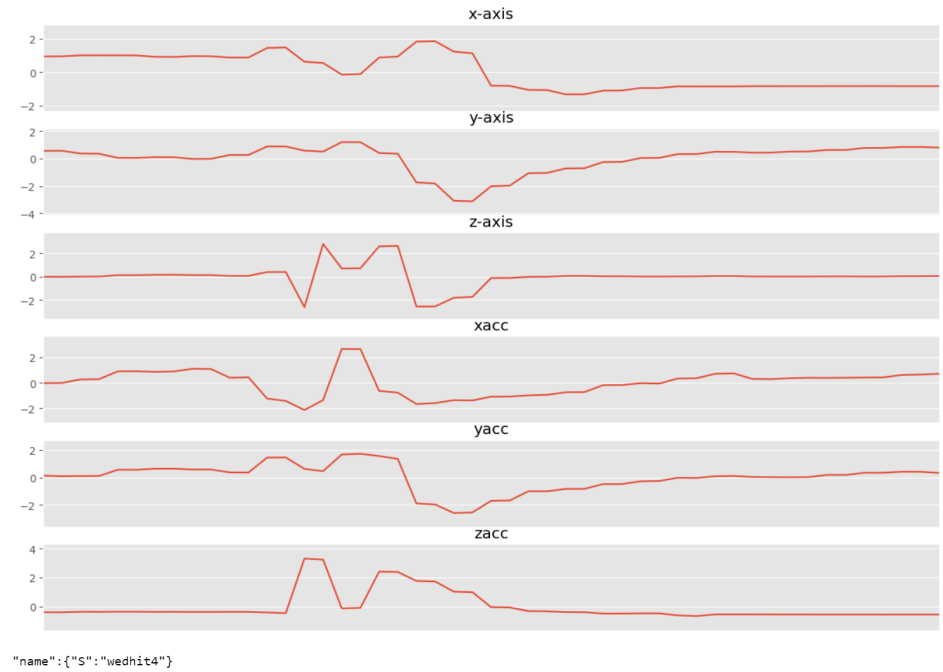


Figure 12b: Hit Data Plots After Preprocessing of Machine Learning

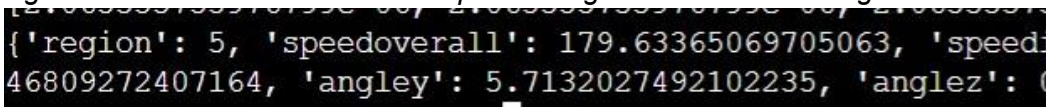


Figure 12c: Screenshot of Output Hit Location from Machine Learning

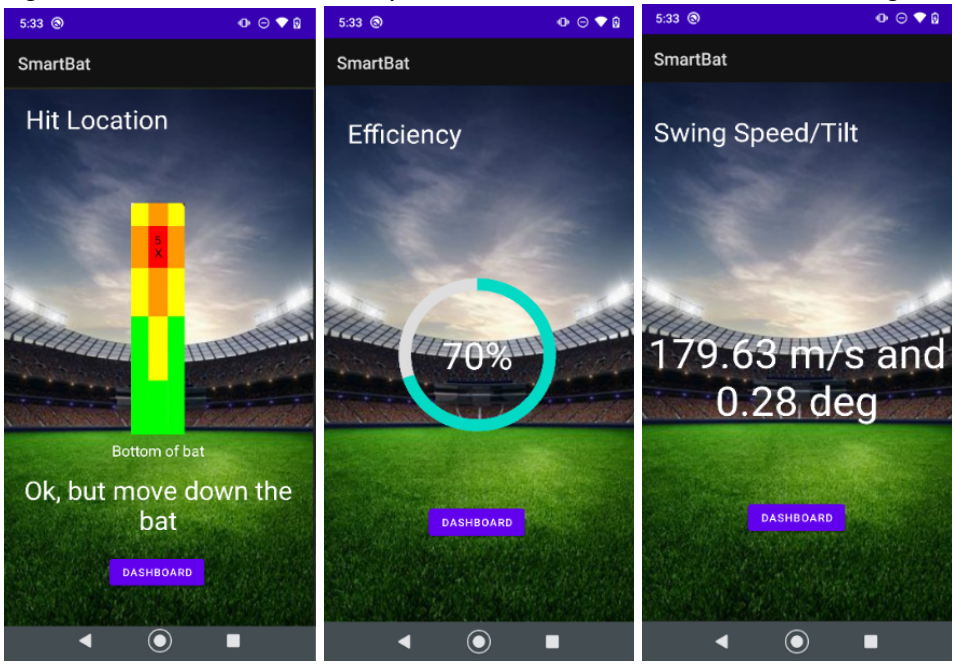


Figure 12d: Output shown in App

Hit Location 3 Data

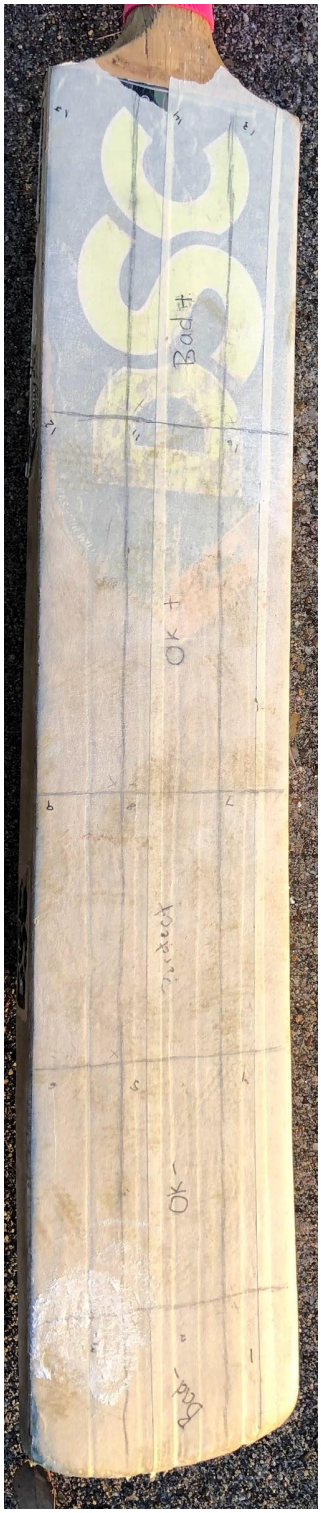


Figure 13a: Marked Physical Collision on Hit Location 3

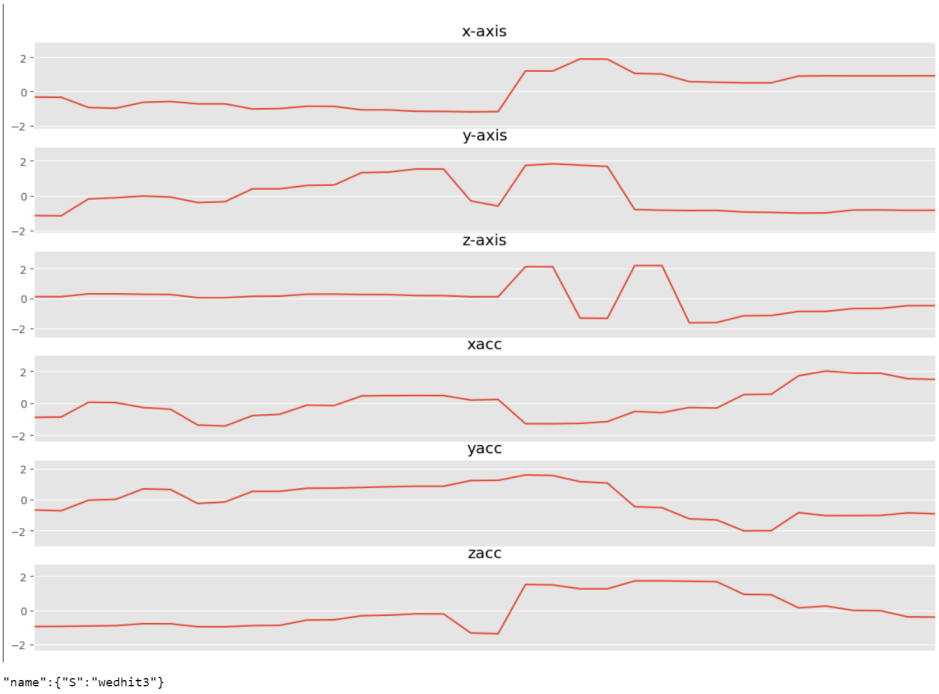


Figure 13b: Hit Data Plots After Preprocessing of Machine Learning

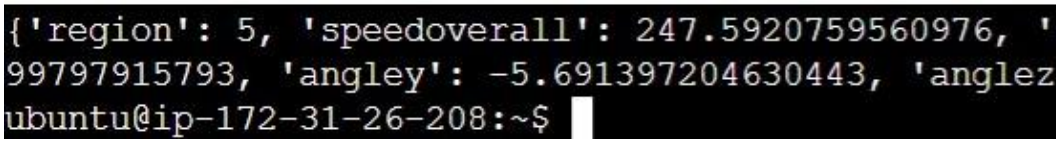


Figure 13c: Screenshot of Output Hit Location from Machine Learning

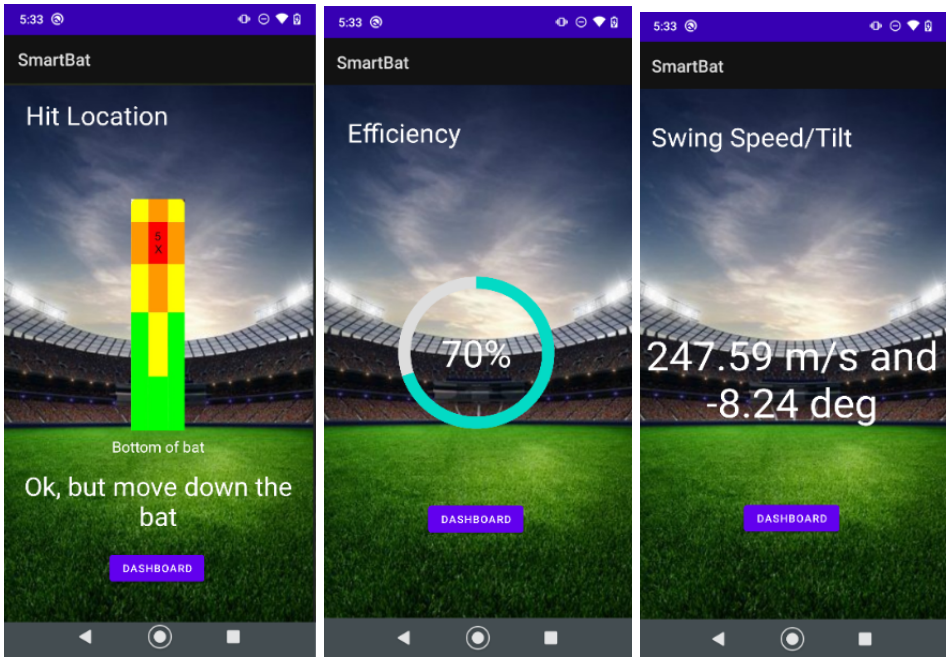


Figure 13d: Output shown in App

Hit Location 2 Data

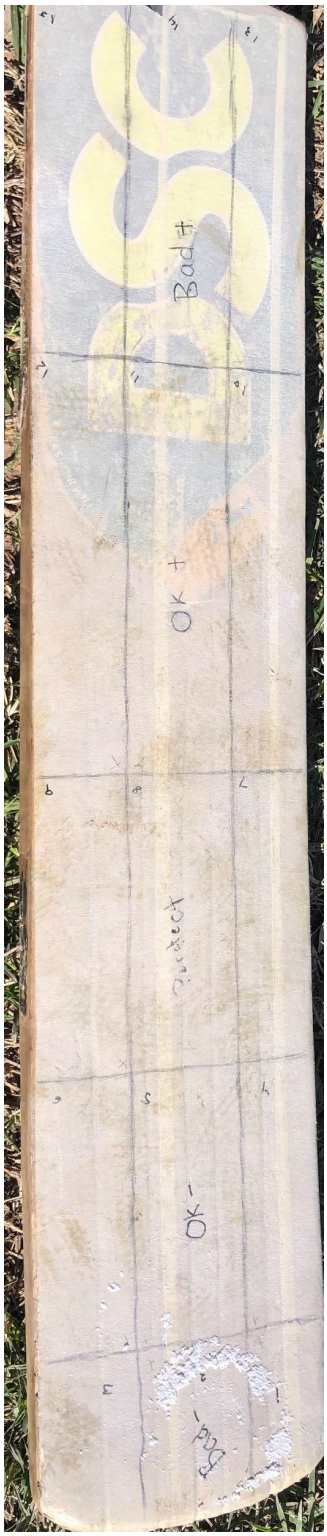


Figure 14a: Marked Physical Collision on Hit Location 2

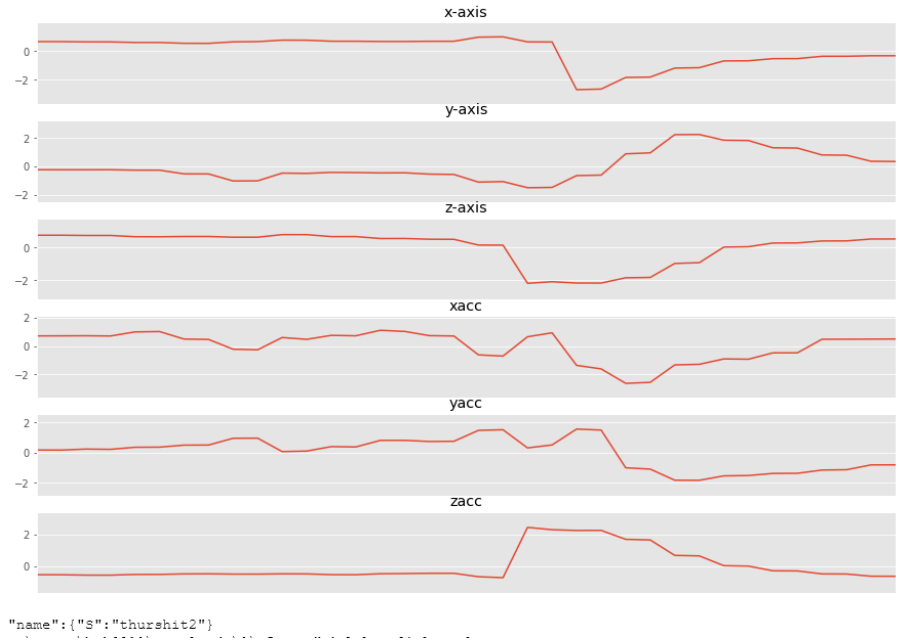


Figure 14b: Hit Data Plots After Preprocessing of Machine Learning

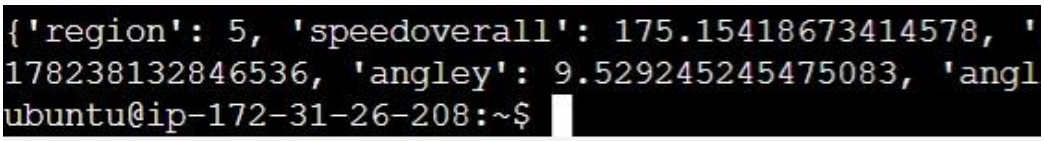


Figure 14c: Screenshot of Output Hit Location from Machine Learning

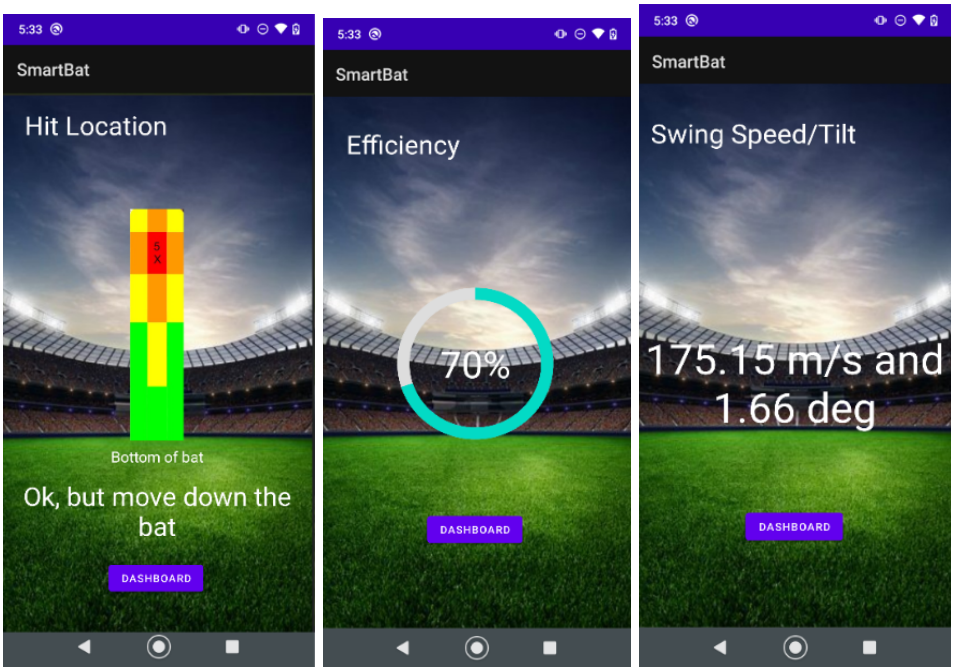


Figure 14d: Output shown in App

Hit Location 1 Data

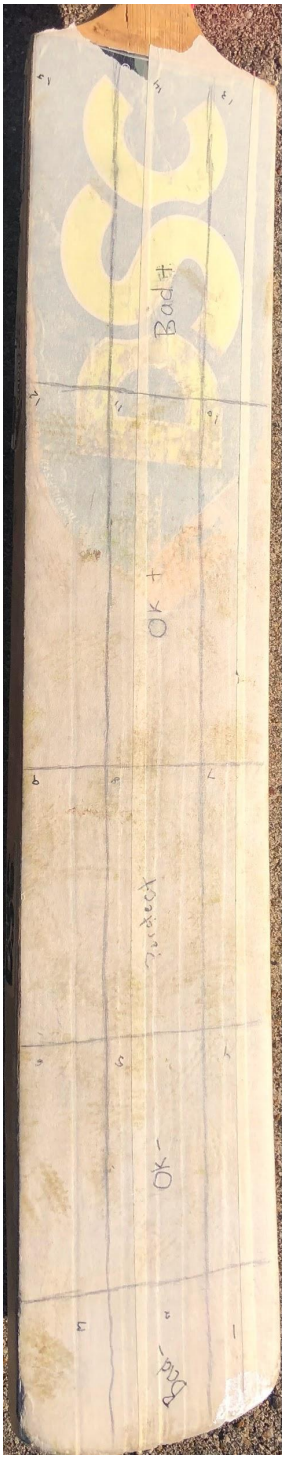


Figure 15a: Marked Physical Collision on Hit Location 1

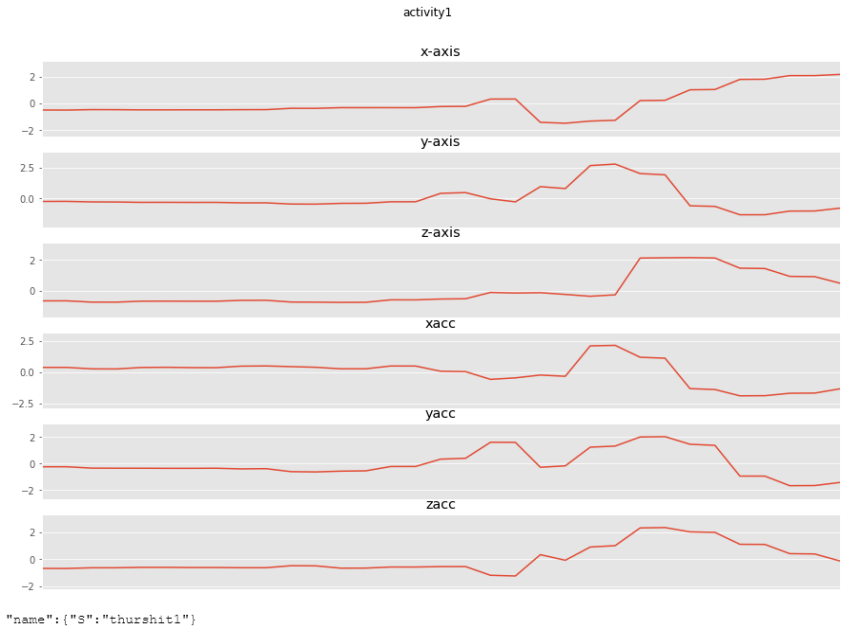


Figure 15b: Hit Data Plots After Preprocessing of Machine Learning

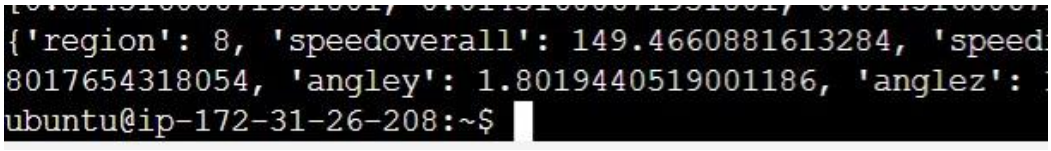


Figure 15c: Screenshot of Output Hit Location from Machine Learning

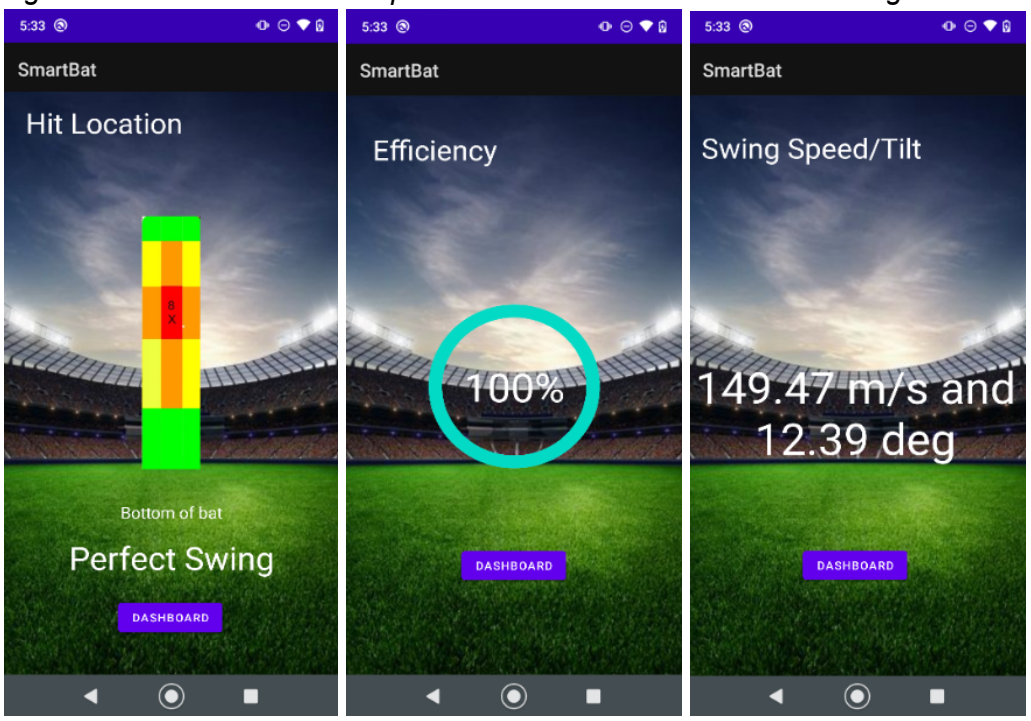


Figure 15d: Output shown in App

System validation was done by simply using the device as a normal user would, using chalk on the ball to mark where on the bat the ball struck, and comparing that location to the output of the system, i.e. the hit location found by the machine learning algorithm. By looking at all the figures above, one can see that the output region calculated by the machine learning model is not correct for most regions and is always either location 5 or 8. This is because of a lack of training data for the machine learning model, the largest group of data acquired were 5 and 8 as they are the most commonly hit regions on a cricket bat. Throughout the prediction of our model on test set data, and further enchanted by validations and the tests we did, we found that an actual hit on region 5 and region 8 can be predicted correctly. So, we think by increasing the training set the overall accuracy of all other regions would increase. Although accuracy is important, our sponsor for this project has said that high accuracy is not the main goal of the project as much as getting a working “proof of concept”. That is being able to go from collecting data on the MCU to being processed by machine learning to being output on the phone app. And this validates that the device created is able to at least deliver the impact and rotational data from a given swing to a machine learning algorithm in the cloud, and deliver an output to a user friendly app with some accuracy.

1.4. System Conclusion

Overall, our fully integrated system was able to meet all but one requirement placed by our sponsor. Our sponsor set a requirement for the duration from gathering swing data to viewing the calculated results from the ML to take no longer than one minute. However, due to some complications with integration of the ML with the app, exporting data from our AWS Amplify GraphQL API to EC2 instance that runs the ML model takes approximately 8 minutes. This exporting process is out of our control since the latency of exporting is controlled by AWS. Once the ML model has completed processing our data, the user is able to view their swing results at their discretion by pressing the designated buttons. In the future we would have wanted for this exporting process to be done automatically to substantially reduce our total run time. Aside from run time, our system was successful in meeting the weight and dimension requirements. These requirements were put in place to allow our device to be as least intrusive to the user during the swing. If the device is too heavy or too big it would significantly affect how the user swings the cricket bat. Our device also met all bluetooth requirements which include a connection range of at least 100 ft and the MCU is controlled by the app via bluetooth. We were able to successfully establish a connection range of at least 220 ft and communicated with the MCU via bluetooth with a press of a button. The app is able to start and stop data receiving, calibrate the device, and determine battery life of the system via trigger signals sent to the MCU through bluetooth. Lastly, our sponsor set a requirement for the battery life to last at least 2 hours to allow the user to complete a full practice session. Our device is able to last about 6 hours on a full battery. Once the user has concluded their practice session, the user can recharge the device via a simple USB connection. If the battery was fully discharged, charge time to full battery will take approximately 7 hours.