Dissertation on

"Gesture Detection using Accelerometer and Gyroscope"

Submitted in partial fulfillment of the requirements for the award of the degree of

Bachelor of Technology
in
Computer Science & Engineering

UE18CS390B – Capstone Project Phase - 2

Submitted by:

Raghav Gupta	**PES2201800056**
Shashank Chaudhary	**PES2201800084**
Akshat Vedant	**PES2201800132**
Niladri Paul Choudhury	**PES2201800611**

Under the guidance of

Prof. Vandana M Ladwani
Assistant Professor
PES University

June - Nov 2021

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING
FACULTY OF ENGINEERING
PES UNIVERSITY
(Established under Karnataka Act No. 16 of 2013)
Electronic City, Hosur Road, Bengaluru – 560 100, Karnataka, India

PES UNIVERSITY

(Established under Karnataka Act No. 16 of 2013)
Electronic City, Hosur Road, Bengaluru – 560 100, Karnataka, India

FACULTY OF ENGINEERING

CERTIFICATE

This is to certify that the dissertation entitled

‘Gesture Detection using Accelerometer and Gyroscope’

is a bonafide work carried out by

Raghav Gupta	**PES2201800056**
Shashank Chaudhary	**PES2201800084**
Akshat Vedant	**PES2201800132**
Niladri Paul Choudhury	**PES2201800611**

In partial fulfillment for the completion of seventh-semester Capstone Project Phase - 2 (UE18CS390B) in the Program of Study -Bachelor of Technology in Computer Science and Engineering under rules and regulations of PES University, Bengaluru during the period June 2021 – Nov. 2021. It is certified that all corrections/suggestions indicated for internal assessment have been incorporated in the report. The dissertation has been approved as it satisfies the 7th-semester academic requirements regarding project work.

Signature	Signature	Signature
Prof. Vandana M Ladwani	Dr. Sandesh B J	Dr. B K Keshavan
Assistant Professor	Chairperson	Dean of Faculty

External Viva

Name of the Examiners **Signature with Date**

1. ______________________ ______________________

2. ______________________ ______________________

DECLARATION

We at this moment declare that the Capstone Project Phase - 2 entitled **“Gesture Detection using Accelerometer and Gyroscope”** has been carried out by us under the guidance of Prof. Vandana M Ladwani, Assistant Professor, and submitted in partial fulfillment of the course requirements for the award of the degree of **Bachelor of Technology** in **Computer Science and Engineering** of **PES University, Bengaluru** during the academic semester June – Nov. 2021. The matter embodied in this report has not been submitted to any other university or institution to award any degree.

PES2201800056	**Raghav Gupta**
PES2201800084	**Shashank Chaudhary**
PES2201800132	**Akshat Vedant**
PES2201800611	**Niladri Paul Choudhury**

ACKNOWLEDGEMENT

I want to express my gratitude to Prof. Vandana M Ladwani, Assistant Professor, Department of Computer Science and Engineering, PES University, for her continuous guidance, assistance, and encouragement throughout the development of this UE18CS390B -Capstone Project Phase – 2.

I am grateful to the Capstone Project Coordinator, Dr. Sarasvathi V, Associate Professor, for organizing, managing, and helping with the entire process.

I take this opportunity to thank Dr. Sandesh B J, Chairperson, Department of Computer Science and Engineering, PES University, for all the knowledge and support I have received from the department. I want to thank Dr. B.K. Keshavan, Dean of Faculty, PES University, for his help.

I am deeply grateful to Dr. M. R. Doreswamy, Chancellor, PES University, Prof. Jawahar Doreswamy, Pro-Chancellor – PES University, Dr. Suryaprasad J, Vice-Chancellor, PES University for providing to me various opportunities and enlightenment every step of the way. Finally, this project could not have been completed without the continual support and encouragement I have received from my family and friends.

ABSTRACT

Human hand gestures are a widely used real-time input modality for technologies that provide a human-machine interface. Computers mathematically analyze hand movements, and this type of technology can be used to exchange data or interact with the environment containing humans and computers or between them. Gesture-based interactions are one of the most comfortable and straightforward ways to communicate with the environment. The efficiency of vision-based approaches or the operation of such devices is highly dependent on lighting conditions and camera-facing angles. Sensor-based approaches thought are void of environmental conditions. However, they face the problem of portability—this study attempts to address this limitation by proposing a sensor-based input device, Arduino Nano 33 BLE. On the other hand, hand gestures have limits in conveying the complexity and diversity of human intentions. Each application program demands various user intentions; we created a hybrid input device to solve these constraints. The computer already recognizes the presently running application. When the currently running program changes, the same gesture provides a new function required by the new app. As a result, the number and complexity of basic hand movements can be significantly reduced. As we want our physical model to be a simple stick fitted with sensors, we employ Arduino Nano 33 BLE containing IMU, which has one miniature three-axis accelerometer and one miniature three-axis gyroscope, the data of which is rasterized and converted to a rasterized image. For continuous gesture identification, we employ a CNN machine learning model. Recognition accuracy is improved by considering gesture patterns, signal strength, and learning mode. Five-unit hand gestures in our system successfully provide most functionality given by numerous input devices. Our approach's characteristics are automatically modified by recognizing application programs or understanding user preferences. The technique exhibits good accuracy (90–96%) in various application programs, promising for building a summary. As users become more and more familiar with the method, accuracy rises to 100%.

TABLE OF CONTENTS

Chapter No.	Title	Page No.
1.	INTRODUCTION	10
2.	PROBLEM STATEMENT	12
3.	LITERATURE REVIEW	16
4.	PROJECT REQUIREMENTS SPECIFICATION	18
5.	STANDARD MACHINE LEARNING WORKFLOW FOR ARDUINO PROJECTS	30
6.	SYSTEM DESIGN (detailed)	32
7.	PROPOSED METHODOLOGY	34
8.	IMPLEMENTATION	38
9.	PSEUDOCODE	51
10.	RESULTS AND DISCUSSION	54
REFERENCES/BIBLIOGRAPHY		58

LIST OF FIGURES

Figure No.	Title	Page No.
2.1	A graph depicting accelerometer data for a jogging human	12
2.2	Data for the same person going downstairs is in this graph.	13
4.1	Arduino Nano 33 BLE Components	19
4.2	Arduino Nano 33 BLE Specifications	19
4.3	Parts of IMU	25
4.4	Axes of our Arduino Nano 33 BLE	26
4.5	Challenge – Human Interpretability	28
4.6	Challenge – Sensor Drift	29
4.7	Challenge – Deployment Sensitivity	29
5.1	Workflow of an Arduino Application	30
6.1	Earlier System Design	32
6.2	Revised and Final System Design	33
7.1	End-To-End Machine Learning Model	34
7.2	Angular Position	35
7.3	Rasterization Process	36
7.4	Flowchart when the system is the indifferent process: (a) While training; (b) While predicting gesture	37
7.5	Final Application Workflow	37
8.1	Application Implementation	38
8.2	Initialization of IMU	39

8.3 The Model Initialization. (a) Model Variable and Space Allocation (b) Model Interpreter variable allocation 40

8.4 Read the data of the Gyroscope and Accelerometer and estimate the drift of the gyroscope 41

8.5 Flatten the three-dimensional coordinates to two-dimensional coordinates and then rasterize that into an image 44

8.6 The process of Rasterization (a) flattens the three-dimensional coordinates to two dimensional and converting to an image (b) converting the image into an RGB image for building a CNN model for classification 45

8.7 Calling the TensorFlow Lite Micro Model for learning and classification 46

8.8 Reduction in size of the model to fit our need of Arduino Nano 33 BLE 47

8.9 The complex machine learning model workflow. It uses the CNN machine learning model for classification 48

8.10 The output of the gesture recognition 49

8.11 Five gestures drew with the help of a stick fitted with an accelerometer and gyroscope. (a) The axes of the IMU. (b)(c)(d)(e)(f) Different gestures recognized by the device. 50

9.1 Hand Gesture Recognition Algorithm Workflow 53

10.1 Establishing connection with the device and API 54

10.2 The input device is used for the application program 55

10.3 A series of experiments involving gestures as input to a multimedia movie player. Video playback and pause. 56

CHAPTER-1

INTRODUCTION

Computers improve the quality of our lives and change the way we live, making them an integral part of our daily lives. The natural human-computer interaction is becoming one of the most researched fields as the computer's presence in our lives grows. We use the simplest or most convenient ways to communicate our intentions or interact with the world in an environment like ours, where we are heavily reliant on computers.

One of the simplest and most popular ways to give commands is a remote-control panel. Just press the buttons. However, such interactions as pressing a button are not always practical or standard. Mainly where the user is elderly or visually impaired, or even specific people with a particular form of disability cannot differentiate between different buttons or click them. Gesture-based interactions are one of the most comfortable and straightforward ways to communicate with the world in such situations.

The hand gesture recognition technology is just a way for computers to interpret hand movements or a technique through which computers mathematically analyze hand movements. We can use this type of technology to exchange data or interact with the environment containing humans and computers or between them. We can use this technology in many different applications or appliances and various domains such as health care, education, etcetera.

There are two types of hand gesture recognition techniques VBR and SBR. VBR stands for Vision-based recognition, and SBR stands for Sensor-based recognition. There have been studies in gesture recognition, but most rely on computer vision. The efficiency of vision-based approaches or the operation of such devices is highly dependent on lighting conditions and camera-facing angles. It is inconvenient, and such limitations often limit the technology's usage in specific environments or for certain users.

Assume a person is watching a movie at home with the lights turned off. It would be challenging to increase the volume of a vision-based gesture recognition device under bad lighting conditions and with a camera-based system. It will also be very awkward and inconvenient always to face the camera while completing a gesture.

When the environment appears not to meet our needs, sensor-based gesture recognition will be helpful. With the growth and rapid development of MEMS technology, one can wear or carry one or more devices fitted with gesture-based sensors or accelerometers, gyroscopes, and other sensors. Mobile phones, such as Apple iPhones, or gaming controllers, such as the Nintendo Wiimote, are examples of these products. These mobile devices or wearables give us a new way to communicate with various applications, such as smart home appliances or mixed reality, for example.

The sensor-based gesture recognition system collects the data of different movements or time series of gesture motions. Many kinds of devices have been used in various studies to capture the acceleration data or data provided by an accelerometer. For example, there are TUB-Sensor Glove, a Tsukada's finger-based gesture input kit, and Mäntyjärv's sensor in a box.

CHAPTER-2

PROBLEM DEFINITION

Many applications in the Machine Learning area use data that is easily understandable by humans. We can understand speech and vision and use this information to create various applications. It is also not difficult for us to perceive visual or audio data and shape a mental picture of what is going on. On the other hand, many data types can give us difficulty understanding many details.

Machines and their sensors produce a massive amount of data or knowledge that our human senses cannot comprehend. Even if we represent the data in a human-readable format, it can be difficult for our minds or brains to understand its patterns or trends.

Let us consider two scenarios: one in which a person is jogging and the other in which the same person is walking downstairs. Below are the two graphs for the two systems. Sensors in cell phones capture the data, and the sensor used is an accelerometer.

The data for jogging is shown in Figure 2-1, while data for walking downstairs is in Figure 2-2.

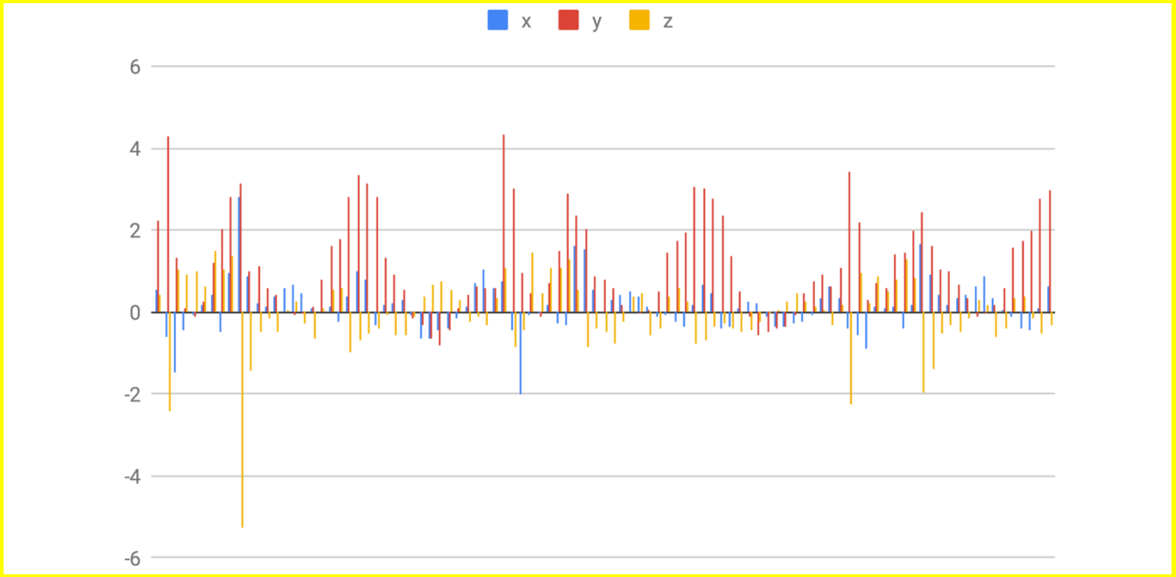


Figure 2-1 A graph depicting accelerometer data for a jogging human

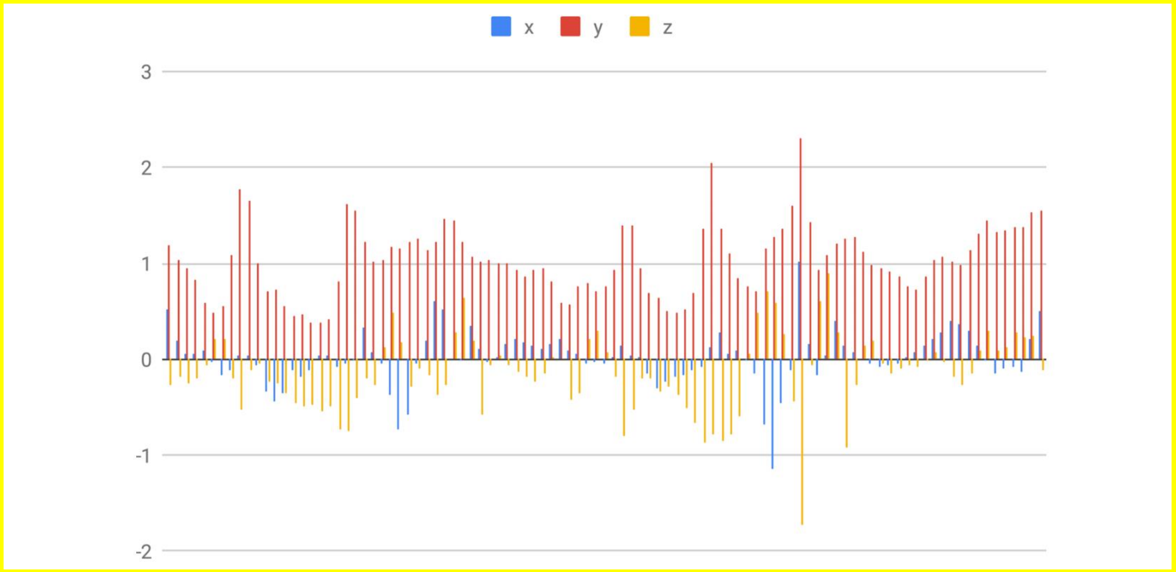


Figure 2-2 Data for the same person going downstairs is in this graph.

Even though the data reflects a routine operation, it is not easy to distinguish between two behaviors, as previously mentioned. Consider attempting to discern the state of a complex industrial machine with hundreds of sensors measuring various properties.

Handwritten algorithms that can make sense of this form of data can often be composed. A human gait expert can distinguish between jogging signs and communicate this understanding as a code feature. A heuristic is a function with various applications in various devices.

Heuristic requires three things, domain knowledge, programming, and mathematical expertise. They can be pretty valuable, but having domain knowledge and coding expertise makes them challenging to build.

Machine learning gives us a new and better approach. Train a model on labeled data, which can learn to differentiate one class or another. Training a deep learning model and embedding it in a microcontroller, we can create intelligent devices that can interact with the environment on a high level and tell us what is going on or can change the setting itself.

Here in this project, we tried to build a gesture recognition device. We have created a hardware model of a stick fitted with an accelerometer & gyroscope, which predicts the gesture made when waved in a 3-dimensional plane.

We first just used an accelerometer in the first attempt, and we tried to build a machine learning model. We tried to scale up our project in the second attempt by introducing a gyroscope.

We used an end-to-end machine learning approach in the first attempt to use an accelerometer. We directly give the raw data generated by an accelerometer to the machine learning model. The model itself does the feature extraction and builds itself, thereby becoming capable of classification. (Accelerometer-based Approach)

We scaled up our approach by introducing a gyroscope in the second attempt. As we introduced a gyroscope, it became hard to classify that gesture if we used an end-to-end machine learning model. So, to overcome this problem, we used a rasterization method for pre-processing and used not the raw data but more precisely processed data to train our classification model.
(Accelerometer & Gyroscope-based Approach)

We will discuss the collection of data and the building of applications in both types of scenarios.

We will conclude our project by building an API to interact with computer systems. Our device will predict the user's gestures and interact with the computer to perform several tasks.

CHAPTER-3

LITERATURE REVIEW

Paper 1

Gesture Recognition with a 3D Accelerometer

Jiahui, Gang, Daqing Zhang, Guande Qi, Shijian

Department of Computer Science Zhejiang University, Hangzhou - July 2009

The FDSVM (Frame-based Descriptor and multi-class SVM), DTW (Dynamic time warping), Naive Bayes (probabilistic classifier), C4.5 (decision tree), and HMM (Hidden Markov model) models are used in this paper to present an acceleration-based gesture recognition approach.

Paper 2

Accelerometer-Based Hand Gesture Recognition Using Artificial Neural Networks

Francisco Arce and José Mario

García Valdez Tijuana Institute of Technology, Tijuana México

Using Artificial Neural Networks in this paper, the team develop a hand gesture recognition algorithm (ANN). The group uses the three-axis accelerometer found in the Wiimote controller to test this method. The group uses the controller to create a wide variety of hand gesture datasets for geometric shapes and letters.

Paper 3

Recognition of Hand Gesture Sequences by Accelerometers and Gyroscopes

Yen-Cheng Chu, Yun-Jie Jhang, Tsung-Ming Tai, and Wen-Jyi Hwang

The paper introduces us to various novel neural network (NN) algorithms that we can use in multiple sensor-based hand gesture recognition systems. From the data provided by accelerometers and gyroscopes, hand gesture recognition algorithms may accurately identify a sequence of gestures or hand gestures.

PairNet (a Convolutional Neural Network (CNN)) is the parent novel neural network (NN) algorithm used, and it can perform simple pairing operations with low computational complexities. Residual PairNet, PairNet with Inception, and Residual PairNet with Inception are three types of novel neural network (NN) or feedforward neural network (NN) used in the paper.

According to the experiment, the PairNet is superior or more effective than the primary convolutional neural network (CNN) and its equivalent, the recurrent neural network (RNN). The Residual PairNet, PairNet with Inception, and Residual PairNet with Inception boost or increase the classification rate or classification hit rate more than the primary paint. In simple words, it shortens the time it takes to recognize hand gestures or improves the efficiency of hand gesture recognition.

CHAPTER-4

PROJECT REQUIREMENTS SPECIFICATION

1. Project Objectives

We have to build a gesture recognition device. We have created a hardware model of a stick fitted with an accelerometer & gyroscope, which predicts the gesture made when waved in a 3-dimensional plane. The IMU, which consists of an accelerometer and a gyroscope, will provide data regarding the stick's velocity across space as the user is moving it. The output from the IMU will be the input for our machine learning model. We have attempted to build the application from two different approaches. One uses an accelerometer and another a combination of accelerometer & gyroscope. Once our model is finished training and ready to do classification, we have to build an API to interact with computer devices. The gesture which the model will predict will be output by the microcontroller. The microcontroller will output information to its serial port with each motion, through which we will operate a connected computer.

2. Hardware Used - Arduino Nano 33 BLE

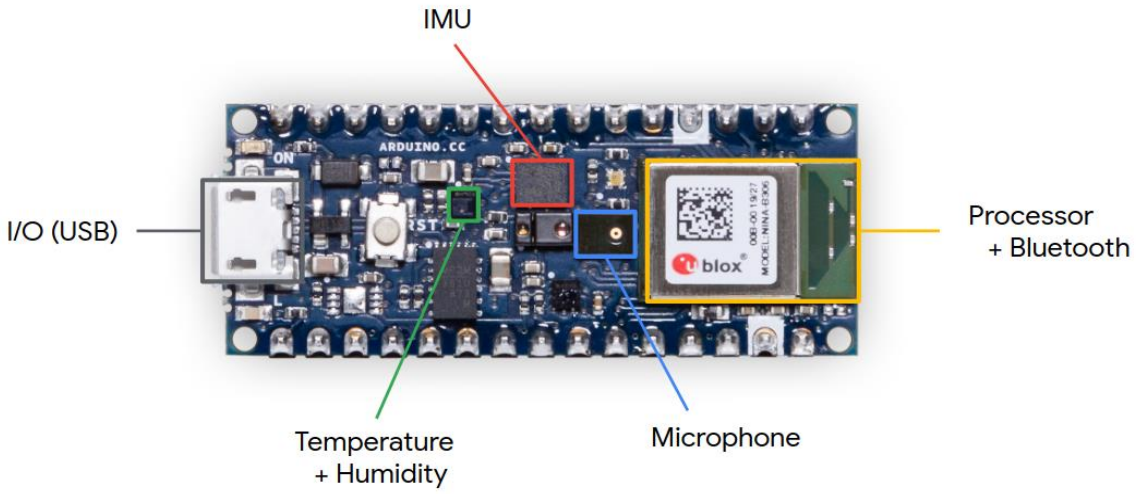
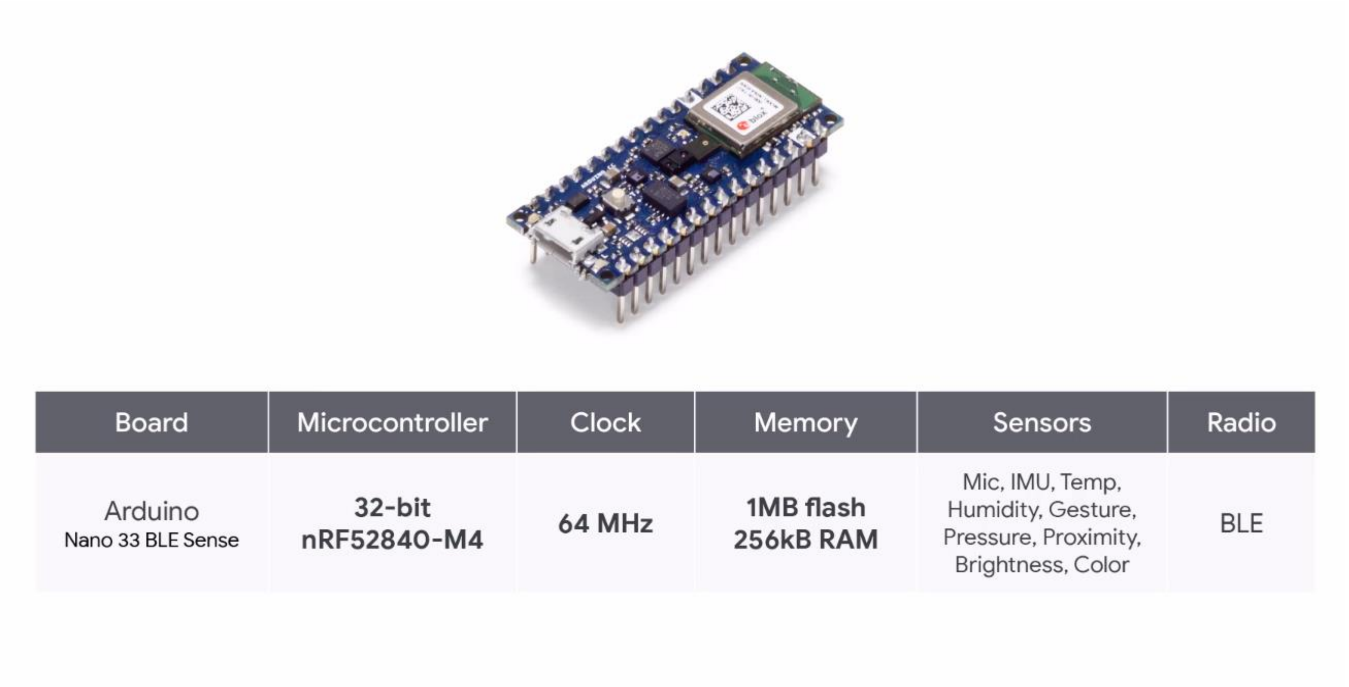


Figure 4.1 Arduino Nano 33 BLE Components



Board	Microcontroller	Clock	Memory	Sensors	Radio
Arduino Nano 33 BLE Sense	32-bit nRF52840-M4	64 MHz	1MB flash 256kB RAM	Mic, IMU, Temp, Humidity, Gesture, Pressure, Proximity, Brightness, Color	BLE

Figure 4.2 Arduino Nano 33 BLE Specifications

The most challenging part of our gesture detection tool is that it must be movable in 3D space, i.e., it must be light. For this, we will use the Arduino Nano 33 BLE board, which is a very light embedded device. It is one of the most suitable embedded devices for the project, with a weight of 5 grams, a length of 45mm, and a width of 18mm.

The exciting thing about this board is that, despite its small size, it contains many of the symbolic elements of an embedded system, all on one printed circuit board (PCB): a collection of surface-mount sensors, a form of the actuator in the onboard programmable RGB LED, and an integrated MCU-BLE module, where BLE is an acronym for a branch of the Bluetooth standard called Bluetooth Low Energy.

These simple constraints govern where and how to deploy such systems; board size and form have significant consequences for use in the field. The quantity and physical size, or package, of the components that live on the board clearly, define board size. An embedded system must have an MCU chip, power supply (often a lithium polymer battery), and power circuitry. The nRF52840 MCU on the Nano 33 BLE Sense is in the U-Blox NINA-B306 MCU-BLE module, which measures 10mm by 15 mm and includes a trace antenna. The MPM3610 step-down converter used to down-regulate the 5V given to the board via USB is 3mm by 5 mm in size, and small SMD passives surround it. Meanwhile, the complete panel measures 18 by 45 mm.

The cost of a development board scales with its MCU's computational capabilities and comprehensive feature set, which is unsurprising. When choosing an MCU for an application, keep in mind that MCUs are frequently used to fulfill specified tasks, and the complexity they will face is typically predetermined or constrained. We chose the Arduino Nano 33 BLE for its versatility, which has a large number of onboard sensors, breakout pins for IO, and a BLE module for projects that require a wireless connection, as well as adequately representative computation parameters for resource-constrained hardware.

3. Software Used - TensorFlow Lite Micro

TensorFlow has surpassed other prominent deep learning frameworks such as PyTorch and Keras to become the most popular. Google's TensorFlow has a Python interface and highly optimized C++ code at its foundation, making it easy to program, fast, and efficient. The library has a strong developer community and has become industry standards for most machine learning applications.

Despite this, TensorFlow is not appropriate for all situations. The typical TensorFlow library is approximately 400 MB in size, and even a modest model (e.g., 200 MB) might consume a significant amount of random access memory (> 1 GB). Running simple models on lightweight platforms is nearly impossible because of the enormous storage and memory requirements.

Recognizing this problem, Google created TensorFlow Lite, also known as TensorFlow Mobile. The TFLite binary is about 1 MB in size, far less than the original library, allowing mobile devices such as smartphones or Arduino to run deep learning models. The size reduction is possible because of eliminating functionality that is generally unnecessary for mobile deployment.

Even while this is an improvement, we still have a problem: TFLite is not ideal for every case. Many major deep learning applications are developed at the microcontroller level, with far fewer resources than mobile devices, typically having less than 1 MB of storage and 256 KB of RAM. Deployment of TFLite models cannot be on microcontrollers because of the above conditions; hence an alternative approach was required.

Then there is the TF Lite Micro. TF Lite Micro compresses the TensorFlow library to the limit, removing everything but the most basic features. In reality, the library's core runtime is only 16 KB, which is several orders of magnitude lower than TFLite. Even the tiniest microcontrollers, such as an Arduino Nano, can have a deep learning model running on them because of this lightweight frame's minimal memory footprint.

However, the reduction of TF Lite Micro is not without its drawbacks. There are some new and distinct obstacles when building models for the TF Lite Micro. For example, removing charting and debugging functionality has made troubleshooting model issues complex. Furthermore, because many microcontrollers lack floating-point units or only support 8-bit arithmetic, the model weights and activations must be quantized appropriately on the microcontroller system. Model training requires near-machine precision to execute gradient descent; therefore, On-device training is impossible. As a result, before being translated to the microcontroller, TF Lite Micro models must first be trained on a device with more computational capacity, adding an extra step to the machine learning workflow.

4. Constraints - Memory

Due to the limited resources available on microcontroller units (MCUs), we must understand the many forms of on-device memory and how they interact and operate. This knowledge will be essential for optimizing machine learning models for a specific hardware architecture.

What exactly is memory? Computer memory stores the information as binary values, often known as bits, with one or zero. However, keeping a single bit is insufficient information; hence numerous bits are frequently bundled into 8-bit values known as bytes. A byte may hold 256 different values, representing everything from alphabet letters to neural network weights.

Microcontrollers use bytes to read from and write to memory which behaves like a memory address. This memory address is a hexadecimal value that, like a postal address, tells the microcontroller where the memory it is seeking is. Microcontrollers have a finite amount of memory, limiting the complexity of programs and computations that they can do. When consistent power is not there, different types of memory behave differently. They may take longer to access or lose data, resulting in the employment of many kinds of memory for various reasons.

RAM and Flash Memory. The two most important forms of memory are flash memory and random-access memory (RAM). Our Arduino Nano 33 BLE comes with 1 MB of flash memory and 256 KB of static RAM.

Data retention is possible because flash memory is non-volatile, even switching the system's power off. We save our model weights and program code in this memory, which we transfer when we "flash" our model onto the device. Keeping to flash memory is a slow procedure that gradually degrades the memory over time. As a result, flash memory is essentially employed as read-only memory and is only overwritten during reprogramming, not when storing intermediate results during program execution, to reduce this degradation.

However, because RAM is volatile, data is lost when the device's power goes off. As a result, temporarily storing variables like input and output buffers and intermediate tensors RAM is advantageous. RAM is substantially faster than flash memory at reading and writing data, making it ideal for use as the primary memory during program execution. There are static RAM (SRAM) and dynamic RAM (DRAM).

Dynamic RAM (DRAM) functions, using a single transistor and capacitor to store a bit. However, the capacitor quickly loses charge and must be refreshed regularly to avoid the recorded information from being lost. Static RAM (SRAM) stores each bit using six transistors and can keep the data stored without recharging it. There is no time or energy spent doing a DRAM refresh between reading and writing operations. SRAM is more expensive than DRAM due to the additional transistors required. However, it is faster and uses less power. Because of these differences, DRAM is a good contender for primary memory in current computers, while SRAM is better for caches. We have access to SRAM in our microcontroller and use it as our main memory during program execution.

While using TF Lite Micro, we must remember that the model weights and code must not exceed the available MCU flash memory (i.e., flash memoryless the 2 KB bootloader).

Also, the size of the input buffer's intermediate tensors must not exceed the device's available SRAM. These are strict limits, and we must be cautious when working with huge models that may surpass them. In such circumstances, a more resource-rich MCU platform or a simpler model may be helpful.

5. Anatomy of an IMU - Accelerometer & Gyroscope

Figure 4.3 Parts of IMU

Sensors like accelerometers, gyroscopes, and magnetometers communicate information about a device's movement from an Inertial Measurement Unit (IMU) system. These sensors provide a thorough account of a device's instantaneous motion, orientation, and acceleration.

These IMU's are helpful in navigation and stabilization in vehicles and airplanes as part of electronic feedback control systems. It is what we will use for our project application. Arduino Nano 33 BLE comes with a 9-axis IMU.

Machine learning may classify different actions connected with a device using data supplied by IMU. For example, let us assume we were attaching an IMU to a single body part, such as an arm. It may be feasible to distinguish different movements such as waving, shaking hands, or typing on a keyboard merely from time-series data. The IMU is especially beneficial for gesture-activated embedded machine learning applications, such as lighting up the phone screen when we pick it up, checking the time on a smartwatch, or even fall detection.

At least two of the following sensors make up an IMU: an accelerometer, a gyroscope, and a magnetometer. In theory, these devices can provide single- or multi-axis measurements. However, most offer multi-axis measures to provide accurate information about a device's location and orientation in three-dimensional space. Our Arduino Nano BLE, which comes with a built-in 9-axis IMU, emphasizes the relevance of the IMU for sensing applications.

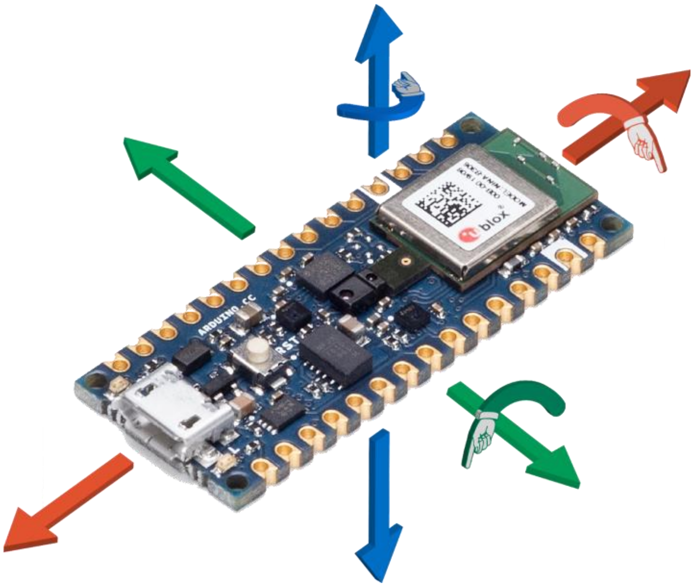


Figure 4.4. Axes of our Arduino Nano 33 BLE

The three sensors that constitute an IMU afford the following capabilities:

Accelerometer. Measures changes in velocity (acceleration), position (coordinates), and absolute orientation. The accelerometer is a mechanism found in tablets and smartphones that guarantees the image on the screen stays upright regardless of the device's orientation. The accelerometer delivers information about the X-, Y-, and Z- directions in both linear and rotational orientations by itself. We can visualize an onboard accelerometer as a micromechanical damped mass-spring system, where the mass-spring system's compression or extension can be mathematically related to an object's acceleration.

Gyroscope. Measures changes in orientation (rotation) and rotational velocity. Many consumer devices, such as gaming controllers, contain microelectromechanical gyroscopes, sometimes known as gyro meters. The rotational X- (roll), Y- (pitch), and Z- (yaw) directions are measured using a gyroscope.

Using an accelerometer only has various flaws, including the contamination of accelerometer results due to linear acceleration altering the Earth's gravity vector.
A 3-axis gyroscope can be added to the system to compensate for these flaws, resulting in a "gyro-stabilized" system. The gyroscope complements the original design by independent measurement of instantaneous rotation speed. The gyroscope uses the principle of conservation of angular momentum, in which a spinning wheel or disc rotates at a fast rate. In contrast, the spin axis rotates freely and can take on any direction. The spinning wheel's inertia permits it to remain in the same order throughout tilting or rotation, allowing rotational data extraction.

6. Challenges of IMU

- **Interpretability:** The notion that the sensor data can make sense of what that data represents. We can see images and hear audio, but when we look at some of the sensor data from IMU. We do not have any easy way to interpret that kind of data—moreover, it is challenging to build an application around it quickly.

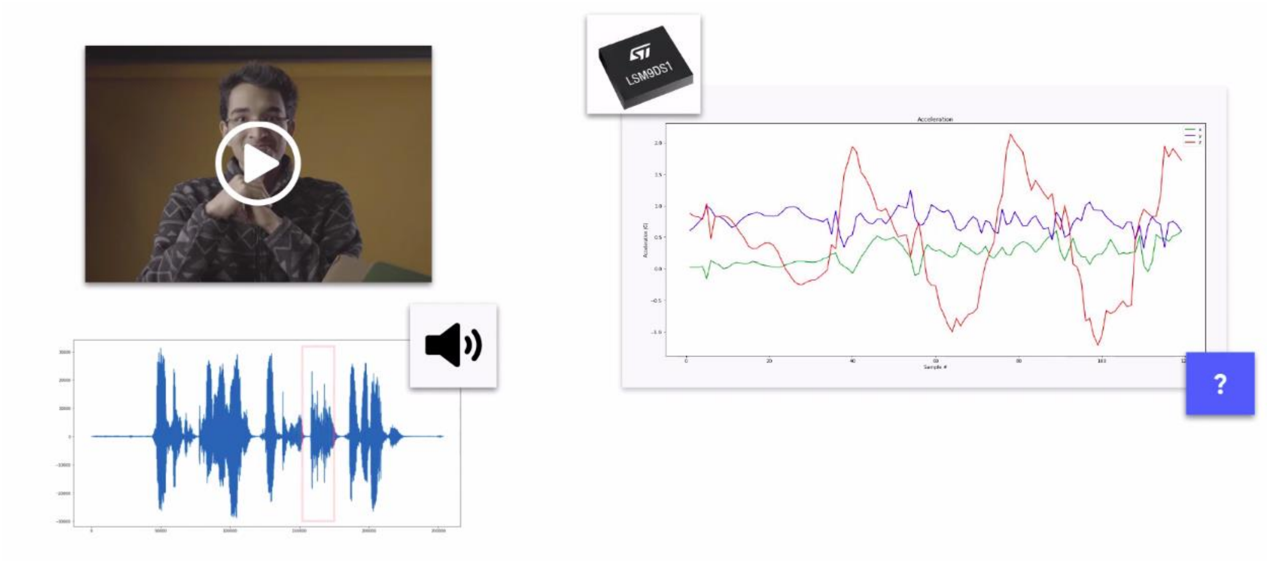


Figure 4.5 Challenge – Human Interpretability

- **Sensor Drift**: The sensors are not quite ideal. In other words, sensors tend to behave differently over time. A perfect sensor would have the same behavior on day 1,000 as it does, for instance, on day zero. Well, it turns out that many of these tiny miniature sensors tend to have problems like drift. For example, Gyroscopes are expressly subject to bias instabilities. That means that their initial zero reading ends up drifting over time. Moreover, that can be problematic if we have made assumptions about how to engineer the application based on day zero. Commercially, the accelerometers are sensitive to vibration, and any non-gravity accelerations can also affect them.

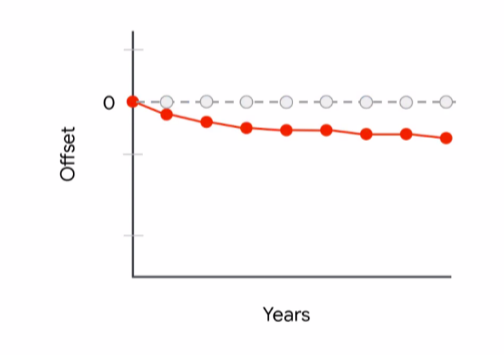
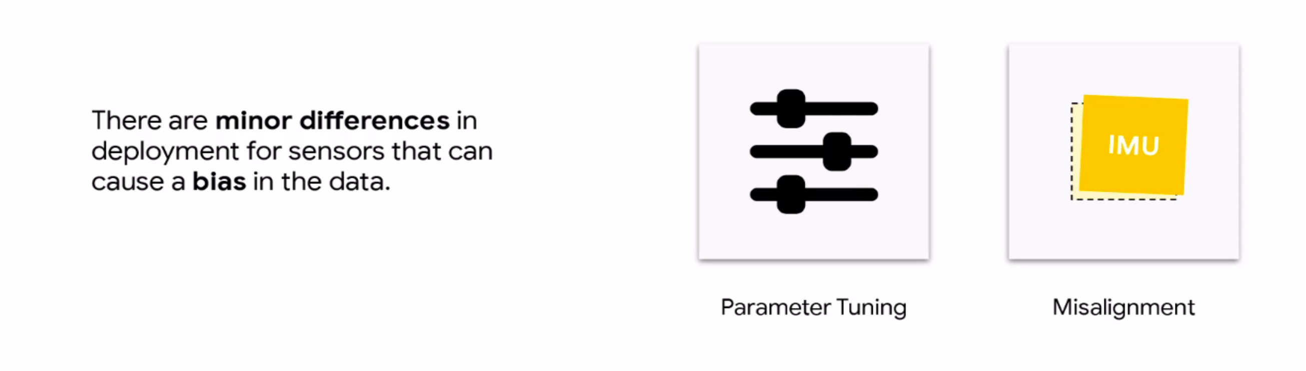


Figure 4.6 Challenge – Sensor Drift

- **Deployment Sensitivity:** Often, we have to calibrate or fine-tune some of these sensors. There tend to be these minor differences that can significantly affect the application outcome. These differences often manifest in biases in the data that we have to compensate for in the pre-processing stage.



There are **minor differences** in deployment for sensors that can cause a **bias** in the data.

Figure 4.7 Challenge – Deployment Sensitivity

CHAPTER-5

STANDARD MACHINE LEARNING WORKFLOW FOR ARDUINO PROJECTS

The first steps of the workflow are similar to a standard machine learning workflow: we collect and process data, then use the TensorFlow framework to create and train a model for a specific machine learning task. The model will most likely require an iterative design to fulfill performance requirements. After that, we use TFLite to optimize our model (e.g., quantizing model weights into a helpful format) and then convert it to a valid format, usually a hex dump file (i.e., a binary file with its contents represented by hexadecimal values). Finally, we flash this hex dump file to the microcontroller's flash memory, which allows it to execute inferences on the embedded device.

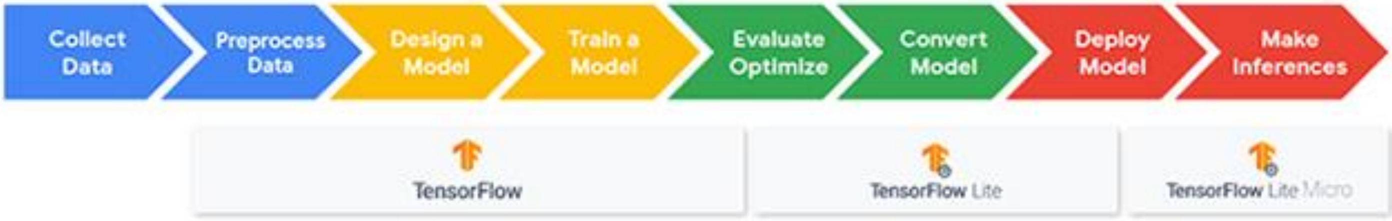


Figure 5.1 Workflow of an Arduino Application

Step 1: Data collection: The most challenging component of a machine-learning-based microcontroller application is generally gathering the data. For the dataset to be valid, datasets must be vast and specific.

Step 2 - Data Pre-processing: Using our NN for efficient inference, we may need to extract features from the input signal for classification. First, we must convert any analog input signals acquired from our sensors into digital signals. We may then need to perform additional modifications.

Step 3 - Model Design: We need a compact model to put onto our microcontroller. We looked at the trade-offs of such models and how tiny they should be.

Step 4 - Training: We will use typical training procedures to train our model to create our dataset. Because we often work with small gradients throughout the training process, training demands extremely exact computation, excluding us from completing our embedded device. As a result, we train our models with TensorFlow on a machine with a lot of processing power.

Step 5 - Evaluation: We then put the absolute accuracy to the test. Initial testing is frequently inadequate, necessitating iterative design to satisfy performance goals and end-user accuracy requirements. These application-specific performance criteria span from inference speed to model accuracy.

Step 6 - Conversion: We will compress and port our working model to our microcontroller device at this point. We begin by suitably quantizing our model weights and activations to be represented by 8-bit or fixed-point arithmetic, and then we convert the model to a hex dump file. This stage is simple to implement, but it is critical to get it right. This stage is crucial for our model to work when deployed on the microcontroller.

Step 7 - Write Deployment Code: To deploy our model to the Arduino Nano with TFMicro, we need all of the preparation code in C++ once we have our model ready. The legend includes the TFMicro runtime and code instructing the Arduino to respond to the machine learning model's data.

Step 8 - Deploy the Full Application: The final step is to put the model into action. This stage entails loading the binary file into the program and flashing the file to the microcontroller via the Arduino IDE.

CHAPTER-6

SYSTEM DESIGN

Our project's system design has been through a series of evolution.
Initially, in our project, we decided to use an accelerometer. We had decided to use raw accelerometer data to build our model for the classification of gestures. As we had decided to use the raw data, we had made an end-to-end model and incorporated it into our application.

Its design is as follows:

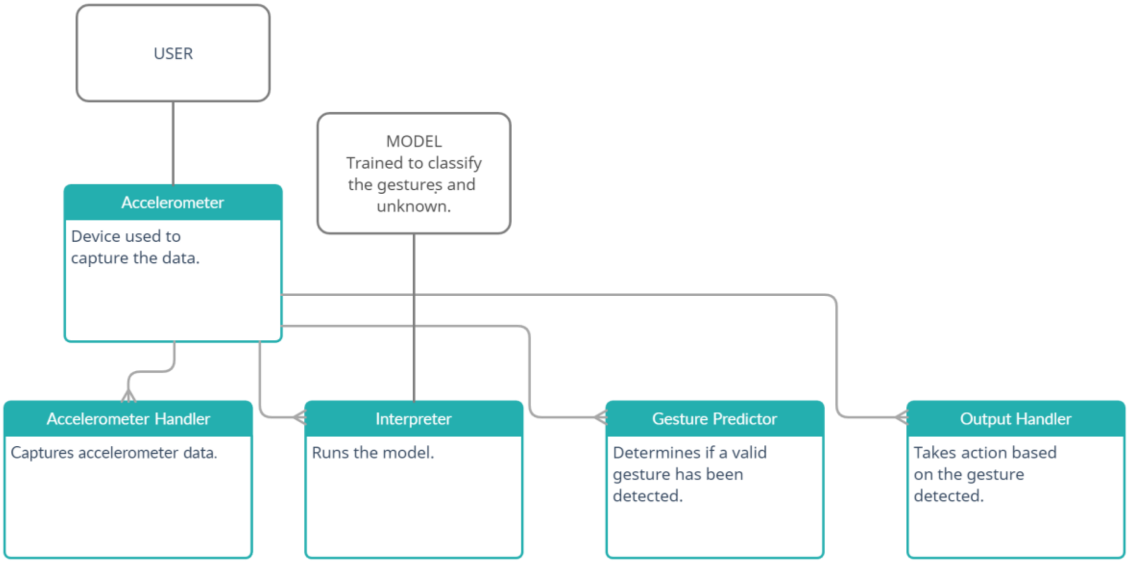

USER
MODEL
Trained to classify the gestures and unknown.
Accelerometer
Device used to capture the data.
Accelerometer Handler
Captures accelerometer data.
Interpreter
Runs the model.
Gesture Predictor
Determines if a valid gesture has been detected.
Output Handler
Takes action based on the gesture detected.

Figure 6.1 Earlier System Design

Using an accelerometer gives us a tiny amount of information about the microcontroller moving in the 3-dimensional space. We scaled up our project by introducing one more component of the IMU called Gyroscope. Gyroscope comes in with its own set of problems and challenges.

The Gyroscope helped us build a more robust model than when we were only using an accelerometer. However, the Gyroscope gives us the coordinates on a three-dimensional spherical coordinate system, and it becomes even more challenging to build a model around it.

Eventually, because of such reasons, we have to change our approach from an end-to-end machine learning project for such reasons. To use some of the well-known pre-processing techniques used to handle cases regarding coordinates in three dimensions called rasterization.

The change in the system design and the final system design is as follows:

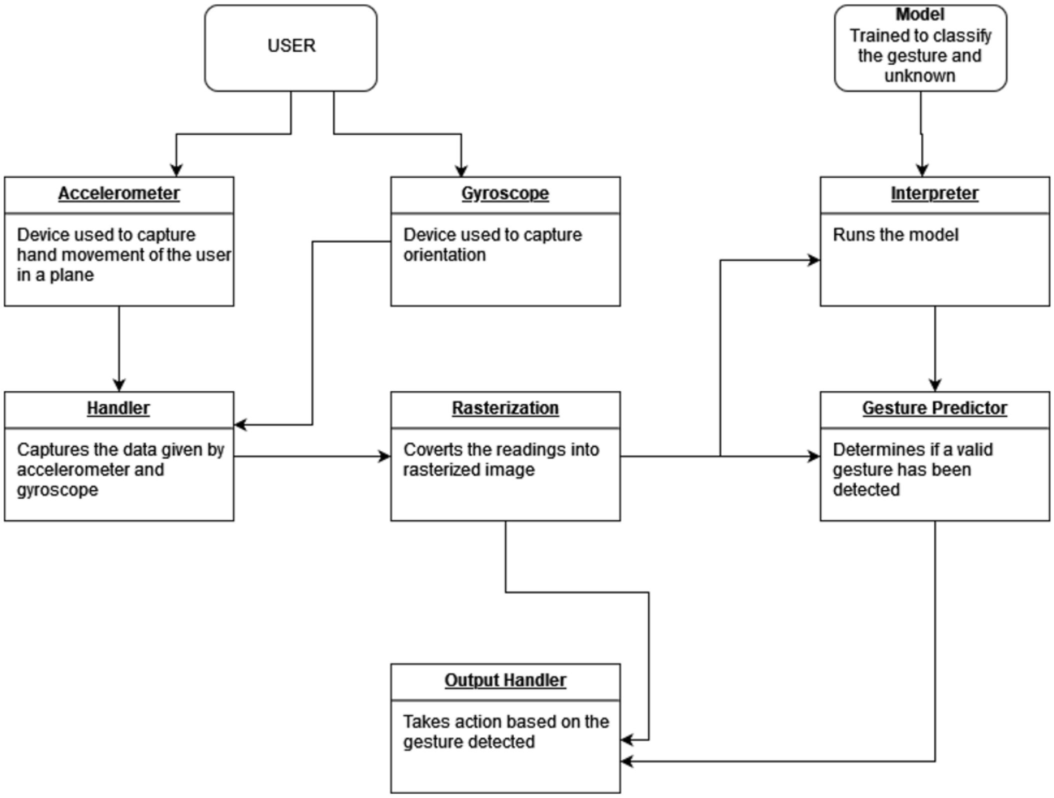

USER
Model
Trained to classify the gesture and unknown
Accelerometer
Device used to capture hand movement of the user in a plane
Gyroscope
Device used to capture orientation
Interpreter
Runs the model
Handler
Captures the data given by accelerometer and gyroscope
Rasterization
Coverts the readings into rasterized image
Gesture Predictor
Determines if a valid gesture has been detected
Output Handler
Takes action based on the gesture detected

Figure 6.2 Revised and Final System Design

CHAPTER-7

PROPOSED METHODOLOGY

In the first approach, the idea is to make an end-to-end machine learning project capable of predicting or detecting the gesture we made using the microcontroller. Such an end-to-end model would require a massive amount of data. It would need a relatively large model to get good accuracy, especially in complicated gestures.

The model built will be trained on and use raw accelerometer data.

Our approach would be pretty straightforward. IMU has an accelerometer inside it. In other words, the IMU can tell when there are sudden changes in a specific direction of the IMU. We can use this to say if the stick was a swing in one particular order with a certain level of acceleration. If that acceleration exceeds a certain threshold, we would say that the device is moving in that direction. We apply the threshold things on a sequence. We build a series of those specific actions we are making and then see if those sequences match some predetermined categories that we might have.

Result: There are many nuances in actually getting this to work robustly. For the model to work perfectly, it is challenging because this ends up being very sensitive to the speed and shape of the gesture. Even slight differences here could lead to significant issues.

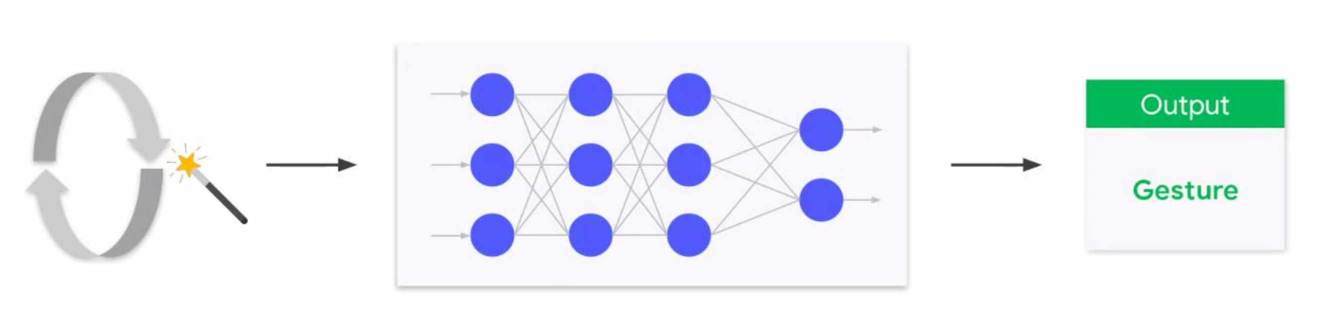


Figure 7.1 End-To-End Machine Learning Model

Accelerometer gives us a tiny amount of information about the microcontroller moving in the 3-dimensional space. We scaled up our project by introducing one more component of the IMU called Gyroscope.

The Gyroscope gives us the angular position of the microcontroller wrt to a fixed point. The angular position gives us a much higher fidelity in the recording we try to make. An improvement would be to use the angular values from the Gyroscope to map the IMU's position onto the spherical coordinate system. Using a gyroscope also helps mitigate the difference in gesture speed and direction since we only estimate the shape of the actual gesture we are making.

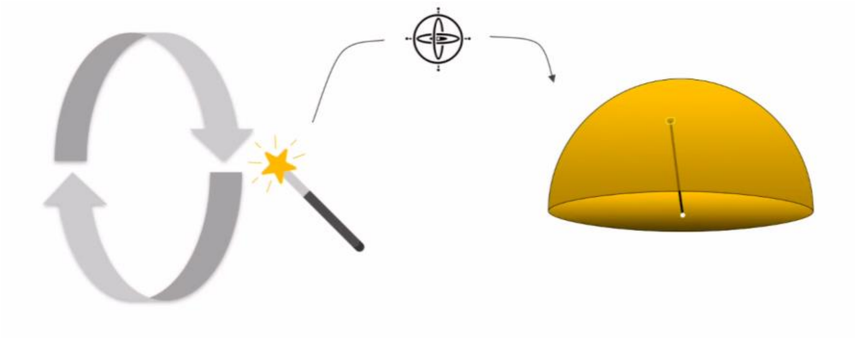
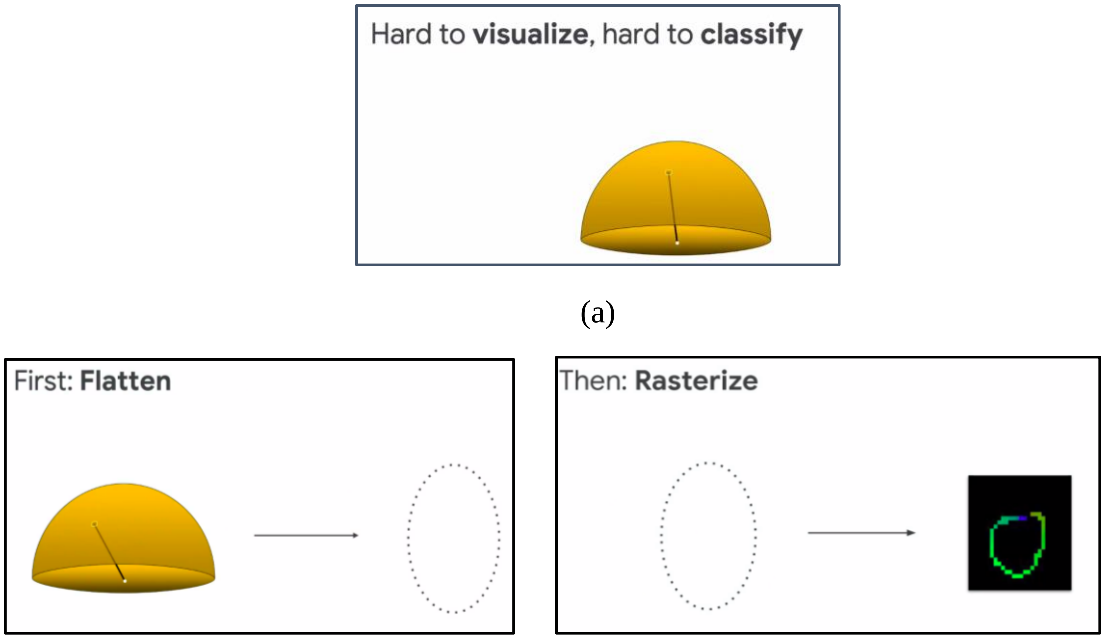


Figure 7.2 Angular Position

As stated above in Anatomy of IMU, an also gyroscope helps us make a gyro-stabilized system.

However, there is a problem that it is hard to visualize; therefore, it is hard to classify that gesture. To imagine it, we turn that angular rotation in the gyroscope into a two-dimensional plot of the gesture. Converting the sensor data into a two-dimensional motion action allows us to visualize the data more efficiently. Now we can take those points that we have collected and then turn them into a 2D image through rasterization.



(a)

(b)

(c)

Figure 7.3 Rasterization Process

Images with defined sizes are more standard and are easier for feature extraction. We will give these images to the neural network for training, which will provide us with a model to classify future gestures and the predicted gesture we made. In the future, when we make a particular gesture, we can do the rasterization again, and then we will give it to the model which will classify it.

At the time of Gesture prediction:

Take the information

↓

Do the flattening

↓

Use the machine learning method for classifications.

(a)

At the time of Gesture prediction:

Take the information

↓

Do the flattening

↓

Use the machine learning method for classifications.

(b)

Figure 7.4 Flowchart when the system is the indifferent process: (a) While training; (b) While predicting gesture

The entire pipeline would start with vectorization of the IMU data from the input side, then rasterize and flatten into a two-dimensional image. We would then give to the neural networks for training. And then future inputs, we would rasterize, and then the model will classify the gesture as an output.

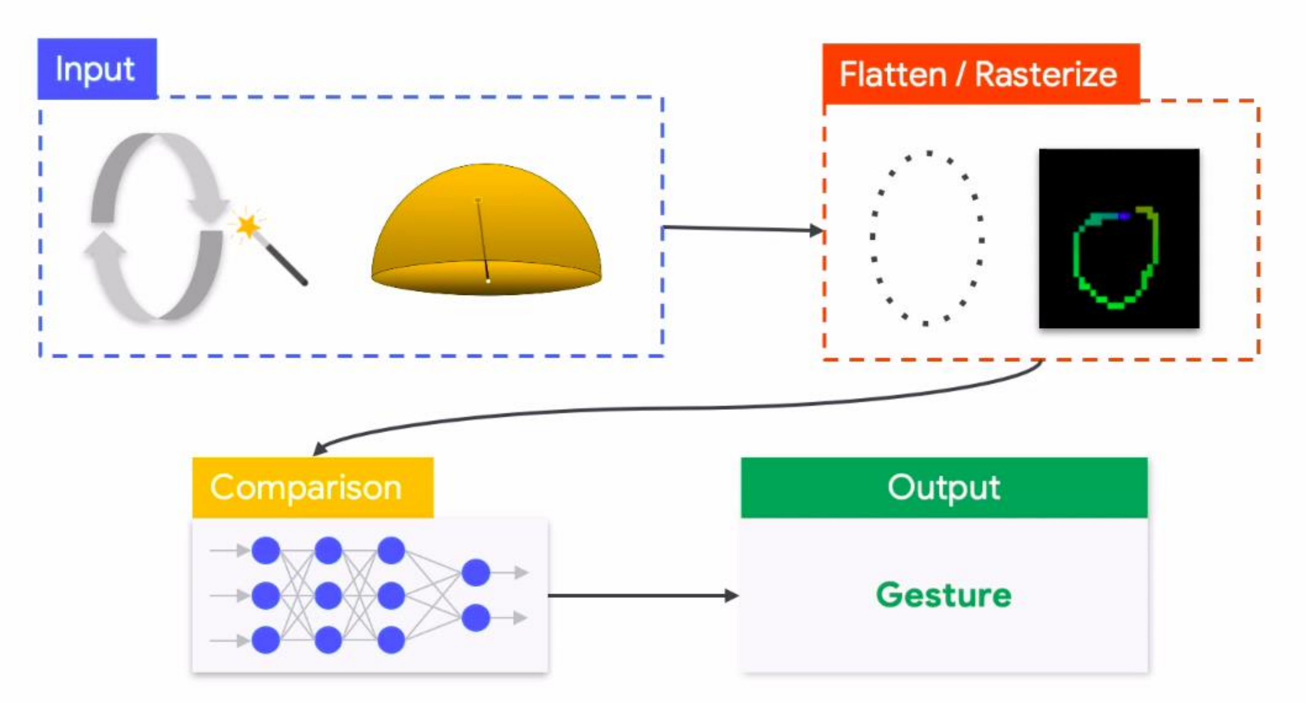
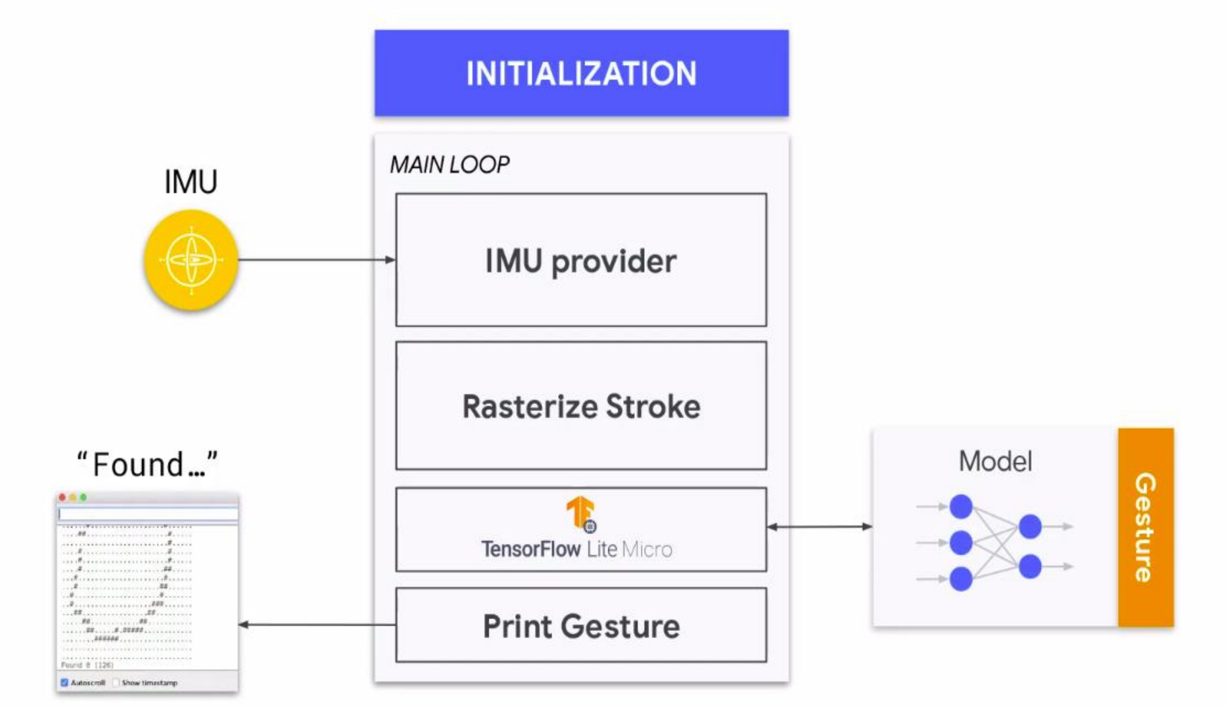


Figure 7.5 Final Application Workflow

CHAPTER-8

IMPLEMENTATION

There are two primary components, which are the initialization phase followed by the main loop.



```
Found 8 (126)
```

Figure 8.1 Application Implementation

- **Initialization**

The initialization phase's job is primarily to set up the IMU and then set up all the resources needed to run the TensorFlow lite macro model.

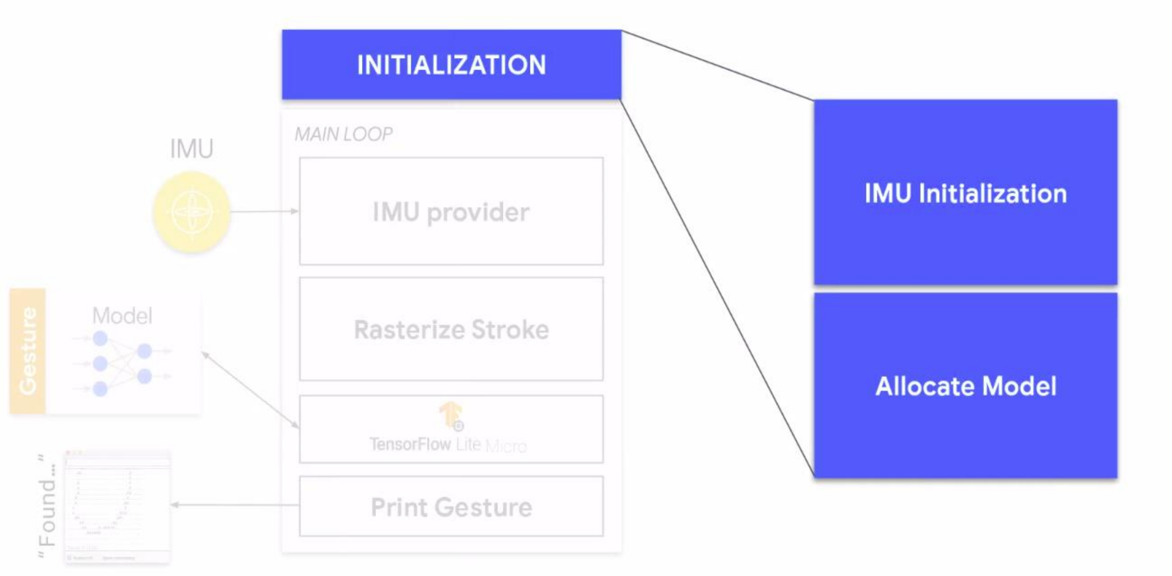
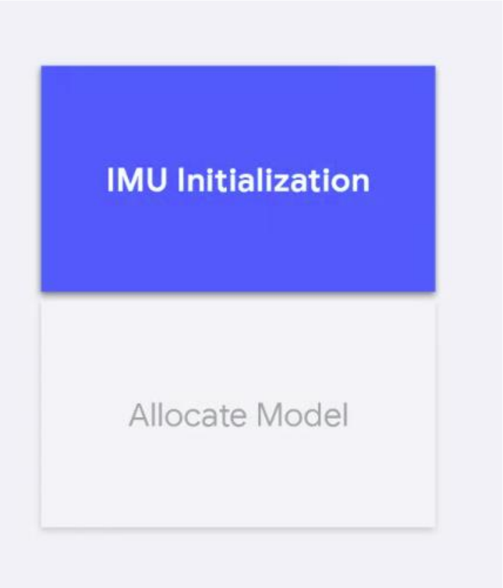
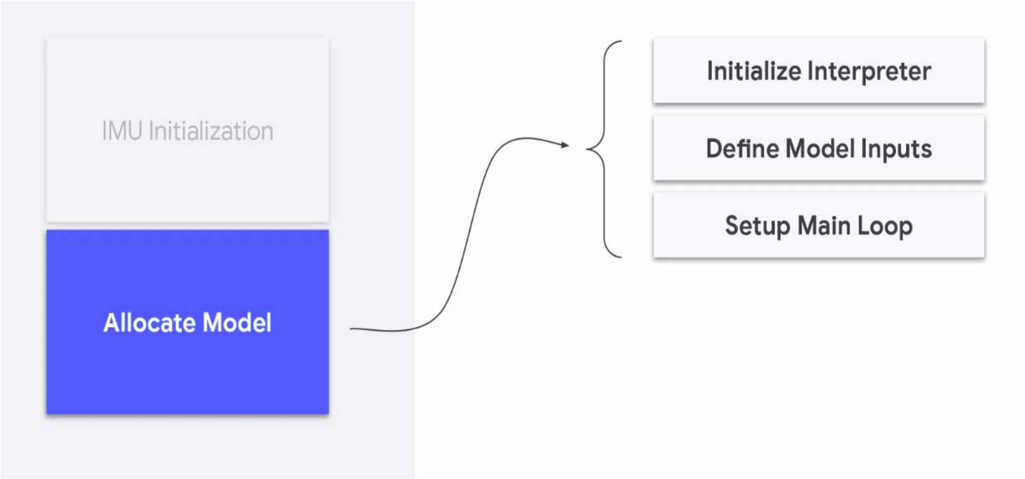


Figure 8.2 Initialization of IMU

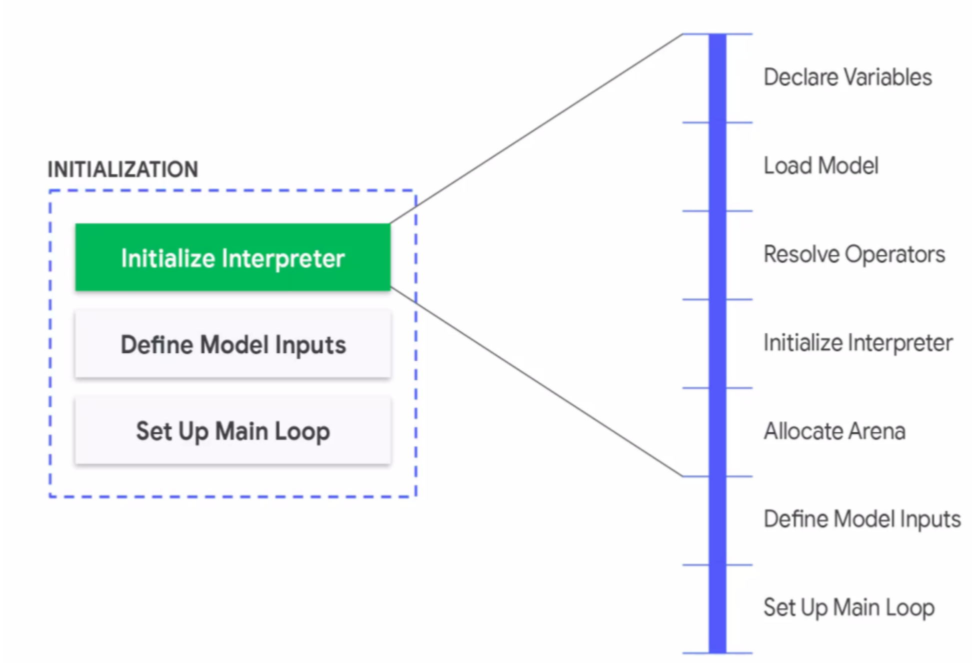


```
if (! IMU.begin()) {
   Serial.println("Failed to initialized IMU!");
   while (1);
 }
 SetupIMU();
 void SetupIMU() {

 IMU.setContinuousMode();

 acceleration_sample_rate = IMU.accelerationSampleRate();
 gyroscope_sample_rate = IMU.gyroscopeSampleRate();
 }
```

The first step of the initialization phase is the IMU initialization, which is done using this **setup IMU** routine. When you go into the setup IMU routine, you will find device-specific calls that tap into the IMU functions that the library provides.



(a)



(b)

Figure 8.3. The Model Initialization. (a) Model Variable and Space Allocation (b) Model Interpreter variable allocation

The second component is setting up all the resources needed to run the model. This might be the pointer to the model, the initialization of the interpreter using the Tensor arena, the model, the observer, etc.

- **IMU Provider - Pre Processing**

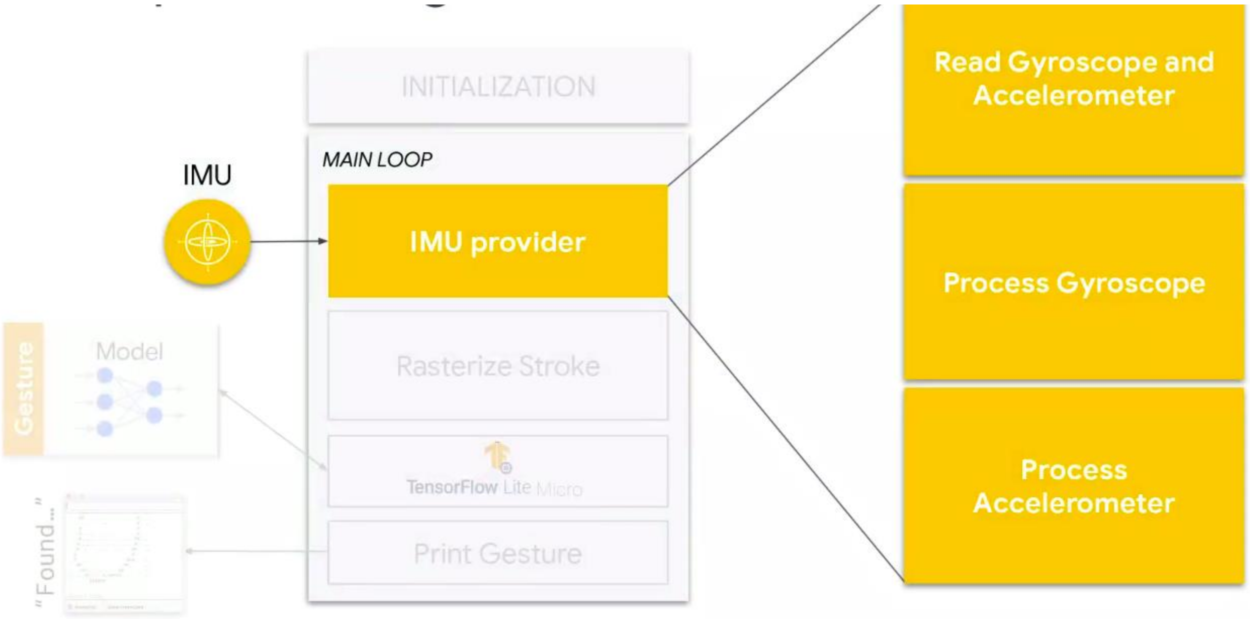
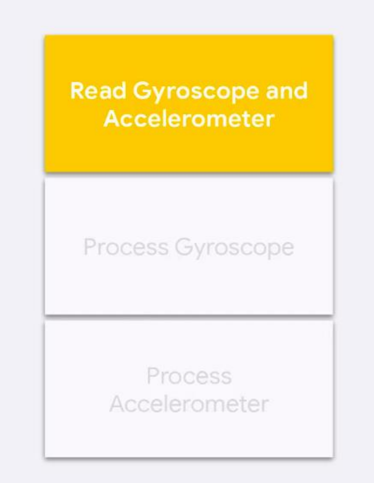
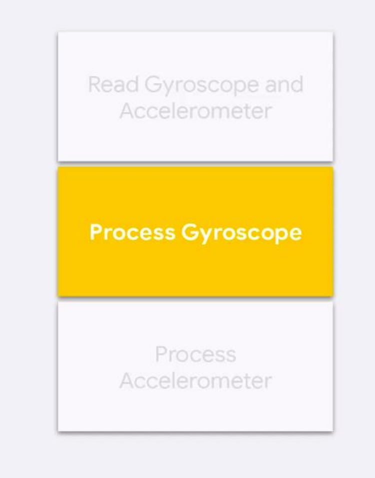


Figure 8.4. Read the data of the Gyroscope and Accelerometer and estimate the drift of the gyroscope

Its job is to get data from the gyroscope and the accelerometer and then process it. There are function calls that are readily available that will allow us to read the data from the gyroscope and the accelerometer. So as long as there are data available from the IMU, we're going to go ahead and process that data.

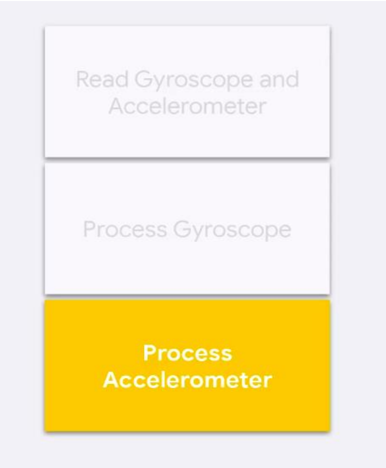


```
const bool data_available = IMU.accelerationAvailable() ||
IMU.gyroscopeAvailable();
  if (! data_available) {
    return;
  }
  int accelerometer_samples_read;
  int gyroscope_samples_read;
  ReadAccelerometerAndGyroscope(&accelerometer_samples_read,
&gyroscope_samples_read);
```



```
bool done_just_triggered = false;
  if (gyroscope_samples_read > 0) {
    EstimateGyroscopeDrift(current_gyroscope_drift);
    UpdateOrientation(gyroscope_samples_read, current_gravity,
current_gyroscope_drift);
    UpdateStroke(gyroscope_samples_read, &done_just_triggered);
    if (central && central.connected()) {
      strokeCharacteristic.writeValue(stroke_struct_buffer,
stroke_struct_byte_count);
    }
  }
```

The gyroscope ends up having a little bit of drift, so we have to compensate for that drift. And that's what this function **estimating gyroscope drift** is doing. When we determine that the IMU is not moving using the accelerometer, we can then calculate the importance of the gyroscope and then account for it subsequently. Next, we want to integrate the gyroscope's incremental angular changes that are coming in overtime because that will give us a part of the gesture in the spherical coordinate system, and that's what this function **updates orientation** doing. It's trying to capture that part that's coming in continuously. Next, we effectively want to project it into a two-dimensional plane inside this physical system. Well, that's because it's much easier for us to understand a 2D gesture than a complex 3D gesture. And therefore, **update stroke** is going to do that flat mapping.



```
if (accelerometer_samples_read > 0) {
   EstimateGravityDirection(current_gravity);
   UpdateVelocity(accelerometer_samples_read, current_gravity);
  }
```

Then comes the process of actually processing the accelerometer data. We want to estimate the direction of the gravity to control for the role of the sensor in the gyroscope readings. Everyone is going to be holding the stick in a specific angular momentum. So this means that you have to be able to neutralize or normalize for that effect. For example, you might be holding the stick with your right hand or holding the post with your left hand. However, the gesture that you’re performing is the same thing. Either way, we've got the same number written, so we get to compensate for that. And the way we do that is by effectively trying to figure out the role of the gyroscopes reading. And then, we update the velocity to know when the sensor is still, and we can correct the sensor drift.

- **Rasterize Stroke - Pre Processing**

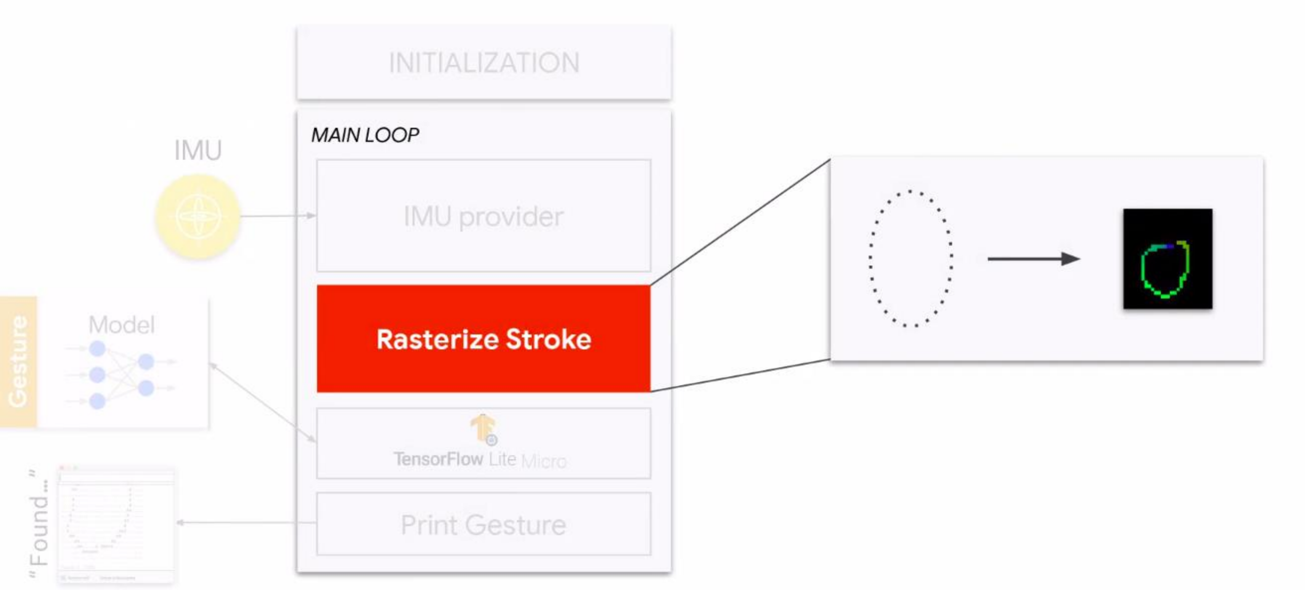
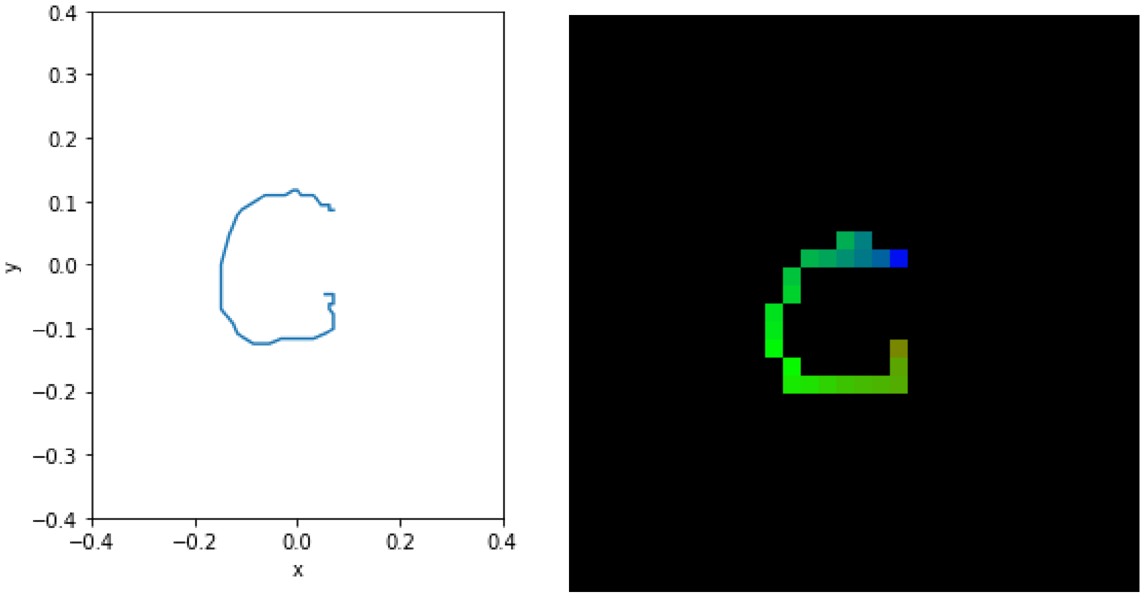


Figure 8.5. Flatten the three-dimensional coordinates to two-dimensional coordinates and then rasterize that into an image.

```
RasterizeStroke(stroke_points, *stroke_transmit_length, 0.6f, 0.6f,
raster_width, raster_height, raster_buffer);
```

The step after doing that, which is effectively capturing the data, is to rasterize that stroke then. We pre-process this because it's easy to feed an image into a convolutional neural network. And there is a function that helps us do that: rasterized stroke.



(a) (b)

Figure 8.6. The process of Rasterization (a) flattens the three-dimensional coordinates to two dimensional and converting to an image (b) converting the image into an RGB image for building a CNN model for classification.

- **Model**

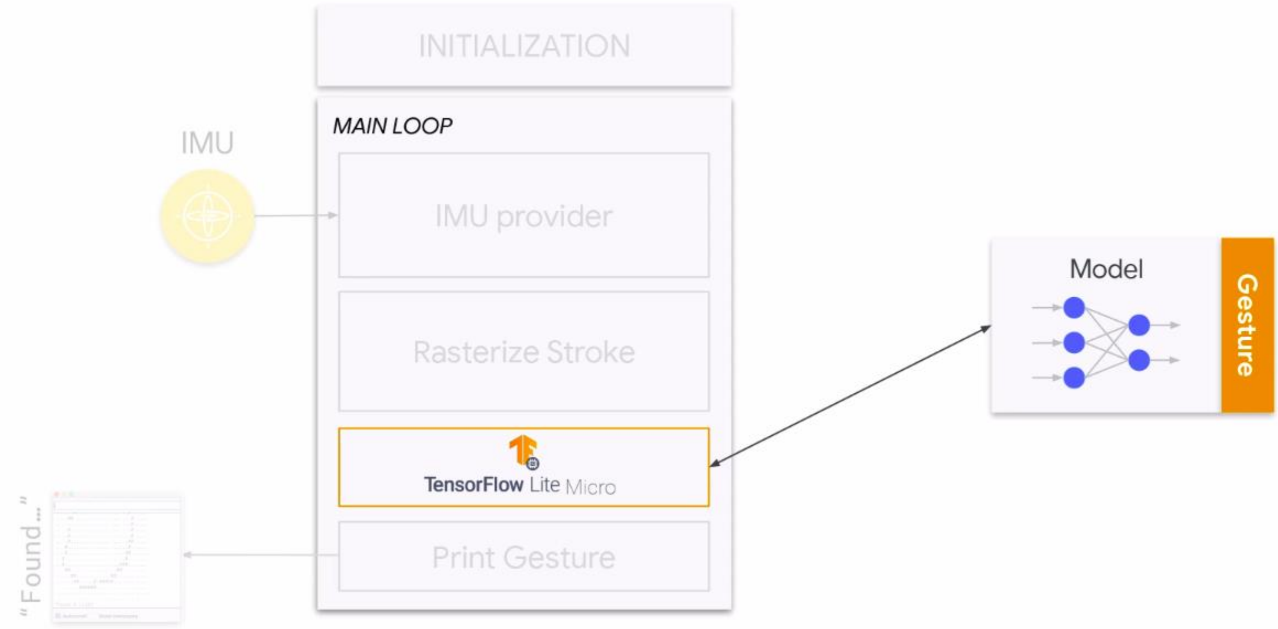
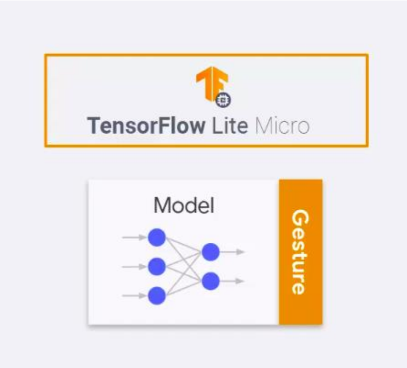


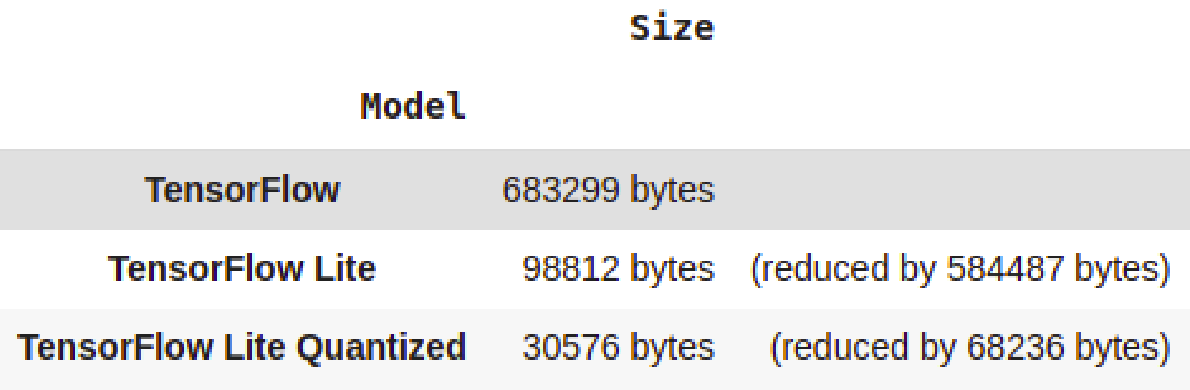
Figure 8.7 Calling the TensorFlow Lite Micro Model for learning and classification



```
TfLiteTensor* model_input = interpreter->input(0);
  for (int i = 0; i < raster_byte_count; ++i) {
    model_input->data.int8[i] = raster_buffer[i];
  }
  TfLiteStatus invoke_status = interpreter->Invoke();
```

After pre-processing, the next part is to hand that rasterized image directly to our convolutional neural net. In this case, we will pass in an RGB image, a red, green, and blue image. So there are three challenges to the idea that we're giving into the net. And that input is then going to be run using a convolutional neural network, which will predict the gesture. To invoke the model, we have to set up the input buffers.

Also, due to having memory constraints, we have to quantize our model.



	Size	
Model		
TensorFlow	683299 bytes	
TensorFlow Lite	98812 bytes	(reduced by 584487 bytes)
TensorFlow Lite Quantized	30576 bytes	(reduced by 68236 bytes)

Figure 8.8 Reduction in size of the model to fit our need of Arduino Nano 33 BLE

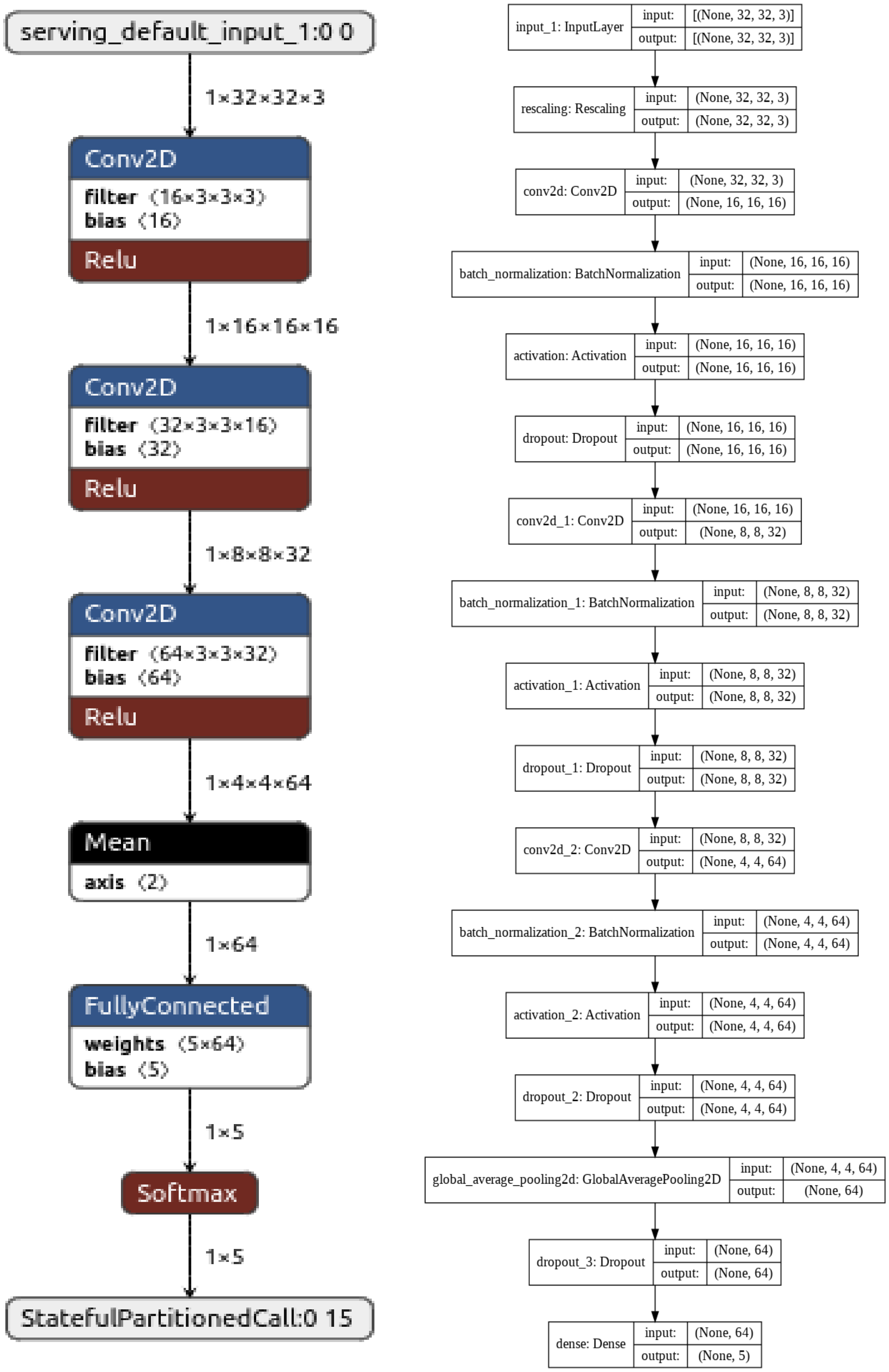
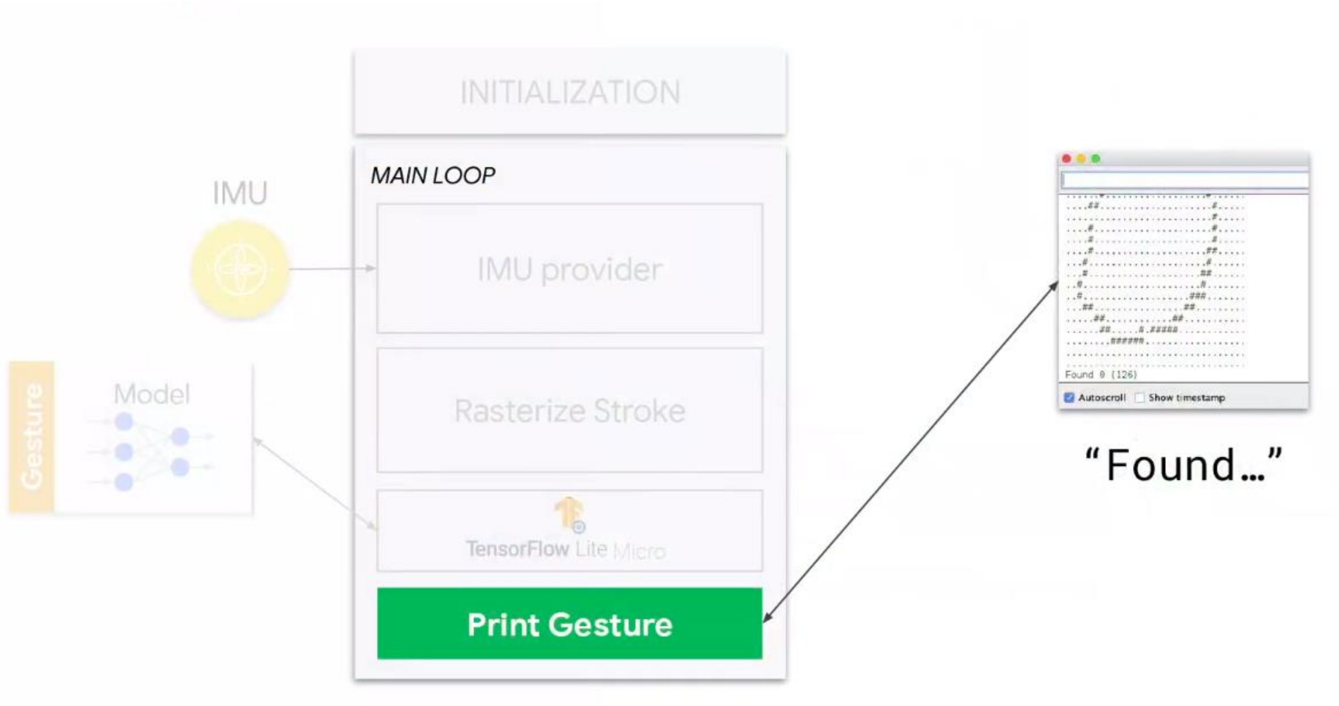

serving_default_input_1:0 0
1×32×32×3
Conv2D
filter ⟨16×3×3×3⟩
bias ⟨16⟩
Relu
1×16×16×16
Conv2D
filter ⟨32×3×3×16⟩
bias ⟨32⟩
Relu
1×8×8×32
Conv2D
filter ⟨64×3×3×32⟩
bias ⟨64⟩
Relu
1×4×4×64
Mean
axis ⟨2⟩
1×64
FullyConnected
weights ⟨5×64⟩
bias ⟨5⟩
1×5
Softmax
1×5
StatefulPartitionedCall:0 15
input_1: InputLayer input: [(None, 32, 32, 3)] output: [(None, 32, 32, 3)]
rescaling: Rescaling input: (None, 32, 32, 3) output: (None, 32, 32, 3)
conv2d: Conv2D input: (None, 32, 32, 3) output: (None, 16, 16, 16)
batch_normalization: BatchNormalization input: (None, 16, 16, 16) output: (None, 16, 16, 16)
activation: Activation input: (None, 16, 16, 16) output: (None, 16, 16, 16)
dropout: Dropout input: (None, 16, 16, 16) output: (None, 16, 16, 16)
conv2d_1: Conv2D input: (None, 16, 16, 16) output: (None, 8, 8, 32)
batch_normalization_1: BatchNormalization input: (None, 8, 8, 32) output: (None, 8, 8, 32)
activation_1: Activation input: (None, 8, 8, 32) output: (None, 8, 8, 32)
dropout_1: Dropout input: (None, 8, 8, 32) output: (None, 8, 8, 32)
conv2d_2: Conv2D input: (None, 8, 8, 32) output: (None, 4, 4, 64)
batch_normalization_2: BatchNormalization input: (None, 4, 4, 64) output: (None, 4, 4, 64)
activation_2: Activation input: (None, 4, 4, 64) output: (None, 4, 4, 64)
dropout_2: Dropout input: (None, 4, 4, 64) output: (None, 4, 4, 64)
global_average_pooling2d: GlobalAveragePooling2D input: (None, 4, 4, 64) output: (None, 64)
dropout_3: Dropout input: (None, 64) output: (None, 64)
dense: Dense input: (None, 64) output: (None, 5)

Figure 8.9. The complex machine learning model workflow. It uses the CNN machine learning model for classification.

- **Output**



```
Found 0 (126)
```

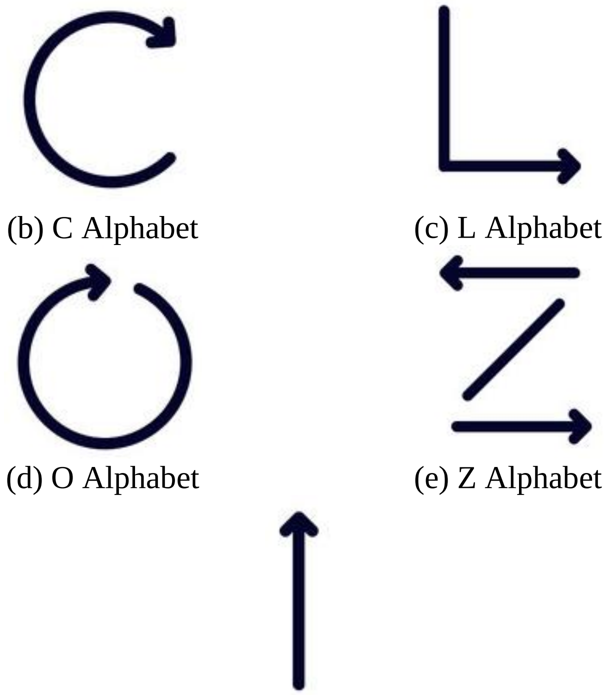
☑ Autoscroll □ Show timestamp

Figure 8.10 The output of the gesture recognition

```
TfLiteTensor* output = interpreter->output(0);
    int8_t max_score;
    int max_index;
    for (int i = 0; i < label_count; ++i) {
      const int8_t score = output->data.int8[i];
      if ((i == 0) || (score > max_score)) {
        max_score = score;
        max_index = i;
      }
    }
    TF_LITE_REPORT_ERROR(error_reporter, "Found %s (%d)",
labels[max_index], max_score);
  }
}
```

We get the output from the neural network to see what it has determined as an actual gesture, and in terms of processing the result, we print the work to the screen.

- **Hand Gestures Recognized**



(b) C Alphabet

(c) L Alphabet

(d) O Alphabet

(e) Z Alphabet

(f) I Alphabet

Figure 8.11. Five gestures drew with the help of a stick fitted with an accelerometer and gyroscope. (a) The axes of the IMU. (b)(c)(d)(e)(f) Different gestures recognized by the device.

CHAPTER-9

PSEUDOCODE

- **Algorithm and Pseudocode**

The Gesture Recognition application accomplishes a reasonably complicated task by carefully crafting a 2D image from 3D IMU data. The dataflow is as follows:

1. The accelerometer data is read and passed to the EstimateGravityDirection (), where it is used to determine the orientation of the Arduino concerning the ground.
2. The Accelerometer data is passed to UpdateVelocity (), where it is used to calculate the velocity of the Arduino.
3. The direction of gravity is passed to UpdateVelocity () and is used to cancel out the acceleration due to gravity from the accelerometer data.
4. The velocity is then passed to EstimateGyroscopeDrift (), determining if the Arduino is stationary or moving.
5. The gyroscope data is passed to EstimateGyroscopeDrift (), where it is used to calculate the sensor drift of the gyroscope if the Arduino is not moving (velocity is 0).
6. The gyroscope data is passed to UpdateOrientation(), where it is integrated to determine the angular orientation of the Arduino.
7. The gyroscope drift is also passed to UpdateOrientation() and subtracted from the gyroscope reading to cancel the essence.
8. The 3D angular orientation is passed to UpdateStroke() transformed into 2d positional coordinates. UpdateStroke() also handles whether the current gesture has ended or a new motion has been started by analyzing the length of the gesture and testing whether the orientation data is still changing.
9. The direction of gravity is also passed to UpdateStroke() to determine the roll orientation of the Arduino.

10. The 2d positional coordinates are passed to RasterizeStroke(), which takes the 2D coordinates and draws lines between them on a 2D image. The color of the lines shifts from red to green to blue to indicate the direction of motion during the gesture.
11. The 2D image of the gesture is converted to ASCII art and printed on the serial monitor.
12. The 2D image of the gesture is passed to the model.
13. The model predicts the label of the gesture, and the title is printed to the serial monitor. Figure 9.1 depicts the above algorithm workflow.

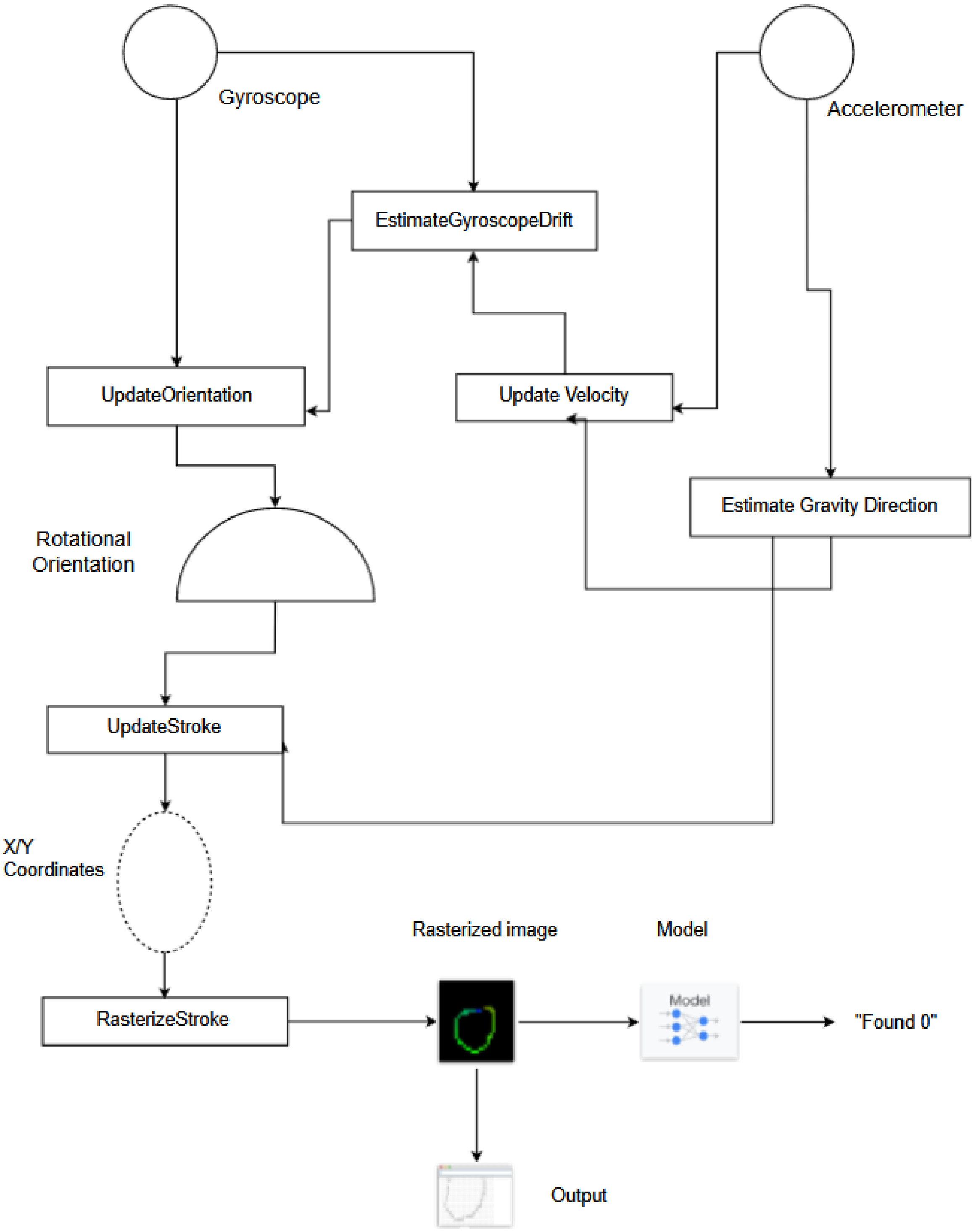


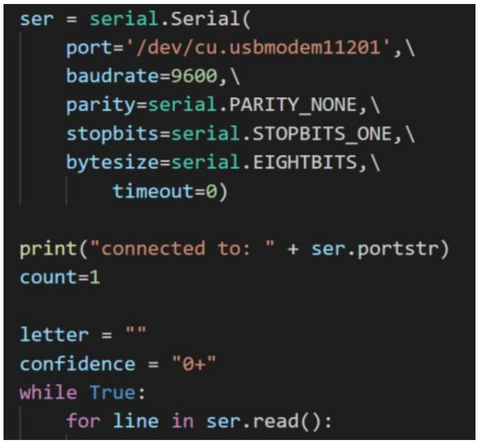
Figure 9.1. Hand Gesture Recognition Algorithm Workflow

CHAPTER-10

RESULTS AND DISCUSSION

After successfully implementing the above workflow, we have a running project to identify a required gesture. We have built an API to interact with the computer or any connected device to complete the project.

The idea behind the API is that once we have the sensor data and we have rasterized it, we will give the formed image to our machine learning model, which will classify it. We must remember that the rasterization and classification are happening on the Arduino Nano 33 BLE. So, after the model classifies the gesture, the Arduino will return the result to the serial port it is connected to.



```
ser = serial.Serial(
    port='/dev/cu.usbmodem11201',\
    baudrate=9600,\
    parity=serial.PARITY_NONE,\
    stopbits=serial.STOPBITS_ONE,\
    bytesize=serial.EIGHTBITS,\
        timeout=0)

print("connected to: " + ser.portstr)
count=1

letter = ""
confidence = "0+"
while True:
    for line in ser.read():
```

Figure 10.1 Establishing connection with the device and API

Once we have uploaded our project on the Arduino Nano, until and unless we cut off the power of Arduino, the code will be kept on running. This means that there is a continuous classification going, and its result is going to the serial port on which the Arduino is connected until we cut off the power.

Several critical approaches were previously discussed, such as rasterization, the model utilized for high-accuracy classification, and the learning mode enabling user-customized interaction. These strategies created a varied HMI input device with easy, real-time, accurate, user-friendly, and multi-functional capabilities. These benefits were demonstrated in follow-up investigations.

Figure 15 depicts our experimental setup. The sensor system consisted of a microcontroller unit, inertial sensor IMU, and an inertial sensor system. An accelerometer, a gyroscope, and a magnetometer were all included in the inertial sensor, but only the three-axis accelerometer and three-axis gyroscope were used in this investigation. The sampling frequency was set to 25 Hz. The sensor system is constructed by mounting the microcontroller on a stick and interacting via USB. Using the BLE (Bluetooth) module, we may increase the usability.

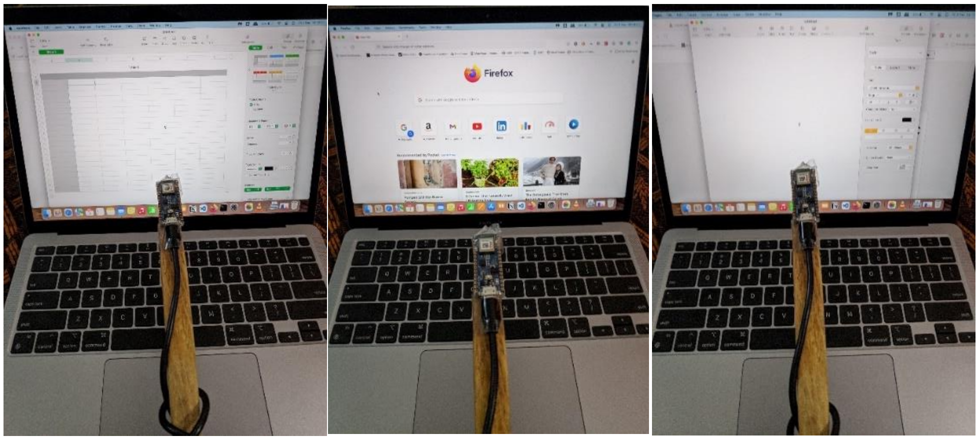


Figure 10.2. The input device is used for the application program.

- **Verification**

The constructed input device is used on multiple application programs controlled by the input device to test the suggested concept. The program's purpose was to transition between various programs and a media player for playing video and media files.

Each experiment was carried out in a particular order. First, we tested the input and gesture recognition device's connectivity and operation. We execute the target program and assess the functions once the device is connected correctly. The five hand gestures in Figure 14 and, if necessary, simple variations of five movements match the activities. After the initial setup, the first volunteer in the experiment activated the device and then performed a scenario involving a series of hand gestures that fulfilled all of the fundamental operations.

- **Verification of Media Player**

The opening of the media player is the initial feature. The current playing file can be played and paused with the second function. The third option is to mute the movie, increase or decrease the volume. Figure 16 shows an experimental video file playback sequence. The following are the play, pause, and volume controls.

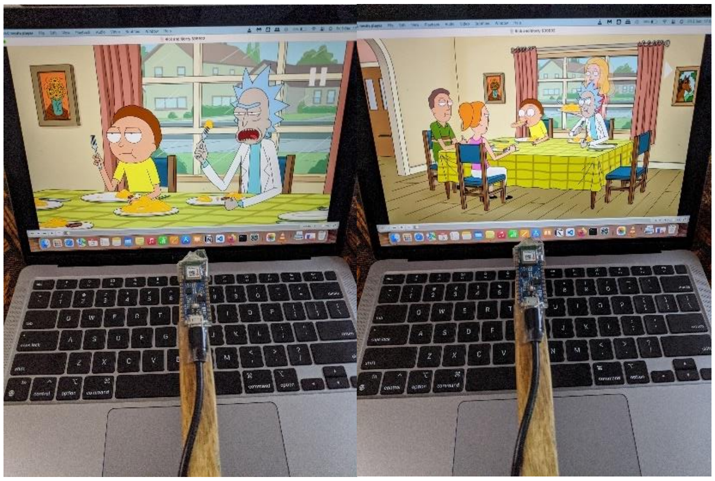


Figure 10.3. A series of experiments involving gestures as input to a multimedia movie player. Video playback and pause.

- **Conclusions**

This paper proposes a sensor-based gesture recognition system that can be used as an input device for the HMI system. Five gestures are being used for multiple different applications. For other types of running on the device, the same device behaves differently at some point. If used to open an application for another application, it could mute the device, pause it, or play it. The project's primary emphasis is on the project's portability and fast and reliable recognition. For portability, we have used Arduino Nano 33 BLE. We use the rasterization process for fast and reliable recognition, which converts the three-dimensional spherical coordinates into two-dimensional coordinates and then rasterizes the image. This image is easy to build a highly accurate and robust model, which ensures our gesture recognition is perfect.

Appendix A: References

1. Gesture Recognition with a 3D Accelerometer Jiahui Wu, Gang Pan, Daqing Zhang, Guande Qi, and Shijian Li Department of Computer Science Zhejiang University, Hangzhou - July 2009
2. Accelerometer-Based Hand Gesture Recognition Using Artificial Neural Networks Francisco Arce and José Mario García Valdez Tijuana Institute of Technology, Tijuana México
3. Gyroscope-Based Continuous Human Hand Gesture Recognition for Multi-Modal Wearable Input Device for Human-Machine Interaction Hobeom Han and Sang Won Yoon Department of Automotive Engineering, Hanyang University, Seoul 04763, Korea
4. Sensor-Based Continuous Hand Gesture Recognition by Long Short-Term Memory Tsung-Ming Tai, Yun-Jie Jhang, Zhen-Wei Liao, Kai-Chung Teng, and Wen-Jyi Hwang Department of Computer Science and Information Engineering, National Taiwan Normal University, Taipei 117, Taiwan
5. Recognition of Hand Gesture Sequences by Accelerometers and Gyroscopes Yen-Cheng Chu, Yun-Jie Jhang, Tsung-Ming Tai and Wen-Jyi Hwang Department of Computer Science and Information Engineering, National Taiwan Normal University, Taipei 117, Taiwan
6. Pavlovic, V.I.; Sharma, R.; Huang, T.S. Visual interpretation of hand gestures for human-computer interaction: a review. IEEE Trans. Pattern Anal. Mach. Intell. 1997, 19, 677–695. [CrossRef]
7. Zhang, Z. Microsoft Kinect Sensor and Its Effect. IEEE Multimed. 2012, 19, 4–10. [CrossRef]
8. Cavalieri, L.; Mengoni, M.; Ceccacci, S.; Germani, M. A Methodology to Introduce Gesture-Based Interaction into Existing Consumer Product. In Proceedings of the International Conference on Human-Computer Interaction, Toronto, ON, Canada, 17–22 July 2016; pp. 25–36.
9. Yang, X.; Sun, X.; Zhou, D.; Li, Y.; Liu, H. Towards wearable A-mode ultrasound sensing for real-time finger motion recognition. IEEE Trans. Neural Syst. Rehabil. Eng. 2018, 26, 1199–1208. [CrossRef]

10. King, K.; Yoon, S.W.; Perkins, N.; Najafi, K. Wireless MEMS inertial sensor system for golf swing dynamics. Sens. Actuators A Phys. 2008, 141, 619–630. [CrossRef]
11. Luo, X.; Wu, X.; Chen, L.; Zhao, Y.; Zhang, L.; Li, G.; Hou, W. Synergistic Myoelectrical Activities of Forearm Muscles Improving Robust Recognition of Multi-Fingered Gestures. Sensors 2019, 19, 610. [CrossRef]
12. Lee, B.G.; Lee, S.M. Smart Wearable Hand Device for Sign Language Interpretation System with Sensors Fusion. IEEE Sens. J. 2018, 18, 1224–1232. [CrossRef]
13. Liu, X.; Sacks, J.; Zhang, M.; Richardson, A.G.; Lucas, T.H.; Van der Spiegel, J. The Virtual Trackpad: An Electromyography-Based, Wireless, Real-Time, Low-Power, Embedded Hand-Gesture-Recognition System Using an Event-Driven Artificial Neural Network. IEEE Trans. Circuits Syst. II Express Briefs 2017, 64, 1257–1261. [CrossRef]
14. Jiang, S.; Lv, B.; Guo, W.; Zhang, C.; Wang, H.; Sheng, X.; Shull, P.B. Feasibility of Wrist-Worn, Real-Time Hand, and Surface Gesture Recognition via sEMG and IMU Sensing. IEEE Trans. Ind. Inform. 2018, 14, 3376–3385. [CrossRef]
15. Pomboza-Junez, G.; Holgado-Terraza, J.A.; Medina-Medina, N. Toward the gestural interface: a comparative analysis between touch user interfaces versus gesture-based user interfaces on mobile devices. Univers. Access Inf. Soc. 2019, 18, 107–126. [CrossRef]
16. Lopes, J.; Simão, M.; Mendes, N.; Safeea, M.; Afonso, J.; Neto, P. Hand/arm gesture segmentation by motion using IMU and EMG sensing. Procedia Manuf. 2017, 11, 107–113. [CrossRef] Sensors 2019, 19, 2562 18 of 18
17. Kartsch, V.; Benatti, S.; Mancini, M.; Magno, M.; Benini, L. Smart Wearable Wristband for EMG based Gesture Recognition Powered by Solar Energy Harvester. In Proceedings of the 2018 IEEE International Symposium on Circuits and Systems (ISCAS), Florence, Italy, 27–30 May 2018; pp. 1–5.
18. Kundu, A.S.; Mazumder, O.; Lenka, P.K.; Bhaumik, S. Hand gesture recognition based omnidirectional wheelchair control using IMU and EMG sensors. J. Intell. Robot. Syst. 2017, 91, 1–13. [CrossRef]
19. Tavakoli, M.; Benussi, C.; Lopes, P.A.; Osorio, L.B.; de Almeida, A.T. Robust hand gesture recognition with a double channel surface EMG wearable armband and SVM classifier. Biomed. Signal Process. Control 2018, 46, 121–130. [CrossRef]

20. Xie, R.; Cao, J. Accelerometer-Based Hand Gesture Recognition by Neural Network and Similarity Matching. IEEE Sens. J. 2016, 16, 4537–4545. [CrossRef]
21. Deselaers, T.; Keysers, D.; Hosang, J.; Rowley, H.A. GyroPen: Gyroscopes for Pen-Input with Mobile Phones. IEEE Trans. Hum.-Mach. Syst. 2015, 45, 263–271. [CrossRef]
22. Abbasi-Kesbi, R.; Nikfarjam, A. A Miniature Sensor System for Precise Hand Position Monitoring. IEEE Sens. J. 2018, 18, 2577–2584. [CrossRef]
23. Wu, Y.; Chen, K.; Fu, C. Natural Gesture Modeling and Recognition Approach Based on Joint Movements and Arm Orientations. IEEE Sens. J. 2016, 16, 7753–7761. [CrossRef]
24. Kortier, H.G.; Sluiter, V.I.; Roetenberg, D.; Veltink, P.H. Assessment of hand kinematics using inertial and magnetic sensors. J. Neuroeng. Rehabil. 2014, 11, 70. [CrossRef]
25. Jackowski, A.; Gebhard, M.; Thietje, R. Head Motion, and Head Gesture-Based Robot Control: A Usability Study. IEEE Trans. Neural Syst. Rehabil. Eng. 2018, 26, 161–170. [CrossRef] [PubMed]
26. Zhou, Q.; Zhang, H.; Lari, Z.; Liu, Z.; El-Sheimy, N. Design and Implementation of Foot-Mounted Inertial Sensor-Based Wearable Electronic Device for Game Play Application. Sensors 2016, 16, 1752. [CrossRef] [PubMed]
27. Yazdi, N.; Ayazi, F.; Najafi, K. Micromachined inertial sensors. Proc. IEEE 1998, 86, 1640–1659. [CrossRef]
28. Yoon, S.W.; Lee, S.; Najafi, K. vibration-induced errors in MEMS tuning fork gyroscopes. Sens. Actuators A Phys. 2012, 180, 32–44. [CrossRef]
29. Xu, R.; Zhou, S.; Li, W.J. MEMS Accelerometer Based Nonspecific-User Hand Gesture Recognition. IEEE Sens. J. 2012, 12, 1166–1173. [CrossRef]
30. Arsenault, D.; Whitehead, A.D. Gesture recognition using Markov Systems and wearable wireless inertial sensors. IEEE Trans. Consum. Electron. 2015, 61, 429–437. [CrossRef]
31. Gupta, H.P.; Chudgar, H.S.; Mukherjee, S.; Dutta, T.; Sharma, K. A Continuous Hand Gestures Recognition Technique for Human-Machine Interaction Using Accelerometer and Gyroscope Sensors. IEEE Sens. J. 2016, 16, 6425–6432. [CrossRef]