University of Applied Sciences
HOCHSCHULE EMDEN·LEER

Improvements

Magic Wand using Arduino Nano

Author: Srikanth Nanda
Matriculation No.: 7026002

Course of Studies : Business Intelligence and Data Analytics

First examiner: Prof. Dr. Elmar Wings
Submission date: January 31, 2025

University of Applied Sciences Emden/Leer · Faculty of Technology ·
Mechanical Engineering Department
Constantiaplatz 4 · 26723 Emden · http://www.hs-emden-leer.de

Contents

List of Figures

List of Listings

Acronyms

CNN Convolutional Neural Network

FOG Fiber Optic Gyro

IDE Integrated Development Environment

IMU Inertial Measurement Unit

IoT Internet of Things

KDD Knowledge Discovery in Databases

LED Light Emitting Diode

MEMS Microelectromechanical Systems

RGB Rot-Grün-Blau

RTOS Real Time Operating System

TFLite TensorFlow-Lite

TinyML Tiny Machine Learning

Part I.

Introduction

1. Introduction

The development of Machine Learning and the evolution of Internet of Things (IoT) have progressed hand-in-hand within these several years. The IoT Idea has permeated various aspects of our lives, such as healthcare, agriculture, intelligent cities, etc.[**Had:2020**]. Also, lots of IoT applications should be able to process data to respond instantly. Handling such requirements by cloud computing is then not suitable[**shi:2016**].

Tiny Machine Learning (TinyML), therefore, is a rapid enhancement in machine learning area due to its low-latency, low power and bandwidth model inference at edge devices[**sakr:2020**]. TinyML primarily employs small form factor devices, for instance, Real Time Operating System (RTOS) based microcontrollers, to run machine learning models and algorithms[**anh:2009**]. Also, one of the most striking features of TinyML is its small size and, in some cases, its ability to function on battery power for years[**Aba:2023**]. This technology heralds a new era of edge services and applications that are not reliant on cloud processing but rather thrive on distributed edge inference and autonomous reasoning.

Arduino is one of the critical components of this project. It allows for the control of board behavior via instructions relayed to the board's microcontroller. Such instructions are provided using the Arduino Integrated Development Environment (IDE), based on Processing and the Arduino programming language. Arduino's original purpose was to serve as a simple prototyping tool for students without prior knowledge of electronics or programming. However, as the platform developed, Arduino boards began to diversify, expanding their scope from simple 8-bit boards to devices designed for IoT applications, wearable technology, 3D printing, and embedded environments[**kushner:2011**]. This adaptability has enabled Arduino to meet new challenges and needs effectively. The Arduino IDE and its Programming Language allow for a high extent of customization to meet the user's requirements, making it a favored development platform for rapid prototyping and idea validation[**kushner:2011**].

This paper delves into using the 3.3V board Arduino Nano 33 BLE Sense. The hardware features three-axis accelerometers, indicating its ability to detect motion and calculate acceleration in three dimensions[**Ard:2021**]. The objective of this report is to construct a "Magic Wand" and integrate it with the board Arduino Nano 33 BLE Sense. The board is programmed to recognize predefined gestures, indicating movement direction for gesture recognition.

The Light Emitting Diode (LED) can be programmed to flash in response to hand movement commands in Arduino. When the wand is moved in a specified direction, the board interprets the gesture and converts it into a visual output. To create the magic wand, the board Arduino Nano 33 BLE Sense is attached to the end of a stick, and the accelerometer's data is fed into a deep-learning model to ascertain whether a recognized gesture has been made. The Arduino board's integral multi-dimensional detector is used to collect gestures, which are vital for understanding complex data. A model can be trained to understand and embed this complex data into the board Arduino Nano 33 BLE Sense.

Additionally, TensorFlow-Lite (TFLite) is used to implement a Neural Network model to recognize gestures using an accelerometer. Triumphant gesture casting results in visualising the corresponding gesture on the screen and the Arduino board's LED lighting up in response to the human input.

A heuristic function is integral to this application. It encodes gesture knowledge into

code, requiring programming skills, mathematical knowledge, and domain expertise. The data collected from the accelerometer undergoes mathematical conversions to produce the desired output or the intended gestures. Instead of creating a heuristic method from scratch, users can select an appropriate model architecture, gather and label a dataset, and iteratively build a model through training and evaluation. The model uses the accelerometer data as input without pre-processing, performs inference, and analyses the results. Three movements have been trained: wing (w), ring (o), and slope (/). The direction of motion for the gestures is shown in the figures 1.1 1.2 1.3. If the input is valid, it generates a visual output of the gestures on a terminal and responds to each spell by lighting an LED. There is also an output to denote an "unknown" gesture if any movement is not recognized.These "unknown" gestures are the motions which are not belong to the wing, ring and slope movements.

Figure 1.1.: **Wing Gesture [War:2020]**

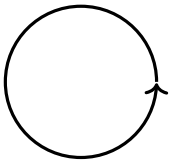


Figure 1.2.: **Ring Gesture [War:2020]**

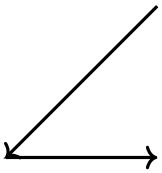


Figure 1.3.: **Slope Gesture [War:2020]**

The planning process encountered several challenges, including:

- The size of the data model should be controlled at a reasonable level due to the limitation of memory in board Arduino Nano 33 BLE Sense [**shi:2016**]
- Creating a Machine Learning data model for the four gestures, especially the "unknown" gesture
- The size of the data model should be controlled at a reasonable level due to the limitation of memory in board Arduino Nano 33 BLE Sense [**Ard:2021**]
- Training the data model to get a more accurate result [**shi:2016**]
- Avoiding extended waved gestures that could lead the accelerometer to record false readings and generate incorrect results.

1.1. Problem's Description

Deploying deep learning (DL) models on edge devices like microcontroller units (MCUs) is a significant challenge due to their limited resources. Most DL algorithms are

designed for high-performance systems such as GPUs or high-performance computing clusters, which makes them incompatible with the memory, computational, and energy constraints of MCUs. This poses a critical issue when adapting DL models for applications requiring real-time processing, such as industrial gauge inspection, where models must be optimized without losing significant accuracy or efficiency.

1.2. Challenges

- **Resource Constraints:**
  Limited memory, computational power, and energy efficiency in MCUs restrict the deployment of traditional DL models. High-performance computer vision models exceed the capacity of edge devices.[**Ard:2021**]
- **Accuracy vs. Efficiency Trade-off:**
  Reducing model size and computational complexity often results in performance degradation, particularly for computer vision tasks.[**shi:2016**]
- **Framework Limitations:**
  Existing tools like TensorFlow Lite and AIfES offer optimization options but are not fully adapted to handle complex computer vision requirements on constrained hardware.[**Kristian:2020**]
- **Real-world Application Needs:**
  Industrial applications, such as gauge inspection, require fast and accurate real-time image processing, which remains challenging to achieve on MCUs with existing models.[**Kristian:2020**]

1.3. Solutions

- **Optimization Techniques:**
  - **Quantization:** Convert weights and activations from 32-bit floating-point to lower-precision formats (e.g., 8-bit integers) to reduce memory and computation demands.
  - **Pruning:** Eliminate redundant parameters in neural networks to reduce complexity without significantly affecting accuracy.
  - **Knowledge Distillation:** Use large, pre-trained models to guide the training of smaller, more efficient models.[**Maciel:2023**]
- **Specialized Frameworks and Tools:**
  - **TensorFlow Lite (TFLite):** Supports model optimization techniques like quantization and pruning and enables efficient execution on embedded devices.[**Kristian:2020**]
  - **AIfES:** Tailored for Arduino-based systems, allowing compact model creation and adaptation for 8-64 bit systems.
  - **TinyNeuralNetwork:** Simplifies model compression and supports seamless integration between TensorFlow Lite and PyTorch.
  - **CMSIS-NN:** Provides optimized neural network kernels specifically designed for ARM Cortex-M processors.[**Kristian:2020**]
- **Real-world Validation:**
  Implement and test optimized DL pipelines for tasks such as industrial gauge inspection, leveraging quantized and pruned models to meet performance and resource constraints. Use converted lightweight models for real-time processing tasks, ensuring that accuracy is maintained within acceptable thresholds.[**Kristian:2020**]

1.4. Report Structure

Following the introduction, our report progresses to a comprehensive exploration of domain knowledge in second section. This section focuses on the nuances of data collection (**??**), management, and its pivotal role, alongside an in-depth analysis of both the hardware (Arduino Nano 33 BLE Sense) in 3 and the utilized software suite (Arduino IDE, TensorFlow, Python, etc.) in **??** Subsequently, we delve into the utilization of Convolutional Neural Network (CNN) in the realm of data mining in **??**, elucidating their components and their pertinence to our project. The report then navigates through section 3, the intricacies of the Knowledge Discovery in Databases (KDD) Process (**??**), covering its development (**??**), deployment (**??**), and the materials involved (5). In section 4 conclusion, the report culminates by summarizing the principal discoveries and proposes avenues for future exploration and development.

Part II.

Domain Knowledge

2. Application

2.1. Application of Magic Wand with an Arduino Nano 33 BLE Sense

The Magic Wand project aims to develop a handheld device that can interact with a computer or other systems through Bluetooth Low Energy (BLE) technology. The device is designed to recognize specific gestures made by the user and translate them into commands or actions. The Magic Wand uses the Arduino Nano 33 BLE Sense board, which is equipped with various sensors, including an accelerometer, gyroscope, magnetometer, and temperature sensor. These sensors allow the device to capture the user's movements and orientation in real-time, enabling gesture recognition and interaction with the system. The Magic Wand project involves programming the Arduino Nano 33 BLE Sense board to read sensor data, process the data to recognize gestures, and communicate wirelessly with a computer or other devices. The project also includes developing a user interface or application on the computer to receive and interpret the gesture commands sent by the Magic Wand. The Magic Wand project demonstrates the capabilities of the Arduino Nano 33 BLE Sense board and explores the possibilities of gesture-based human-computer interaction using BLE technology. The project has various applications in fields such as gaming, virtual reality, human-computer interaction, and assistive technology. By developing a responsive and efficient gesture recognition system, the Magic Wand project aims to provide an intuitive and interactive way for users to control and interact with digital systems and devices. The project also highlights the potential of using low-cost and accessible hardware platforms like the Arduino Nano 33 BLE Sense for developing innovative and interactive applications in various domains. The Magic Wand project showcases the power of combining hardware, software, and sensor technologies to create novel and engaging user experiences and demonstrates the possibilities of using gesture-based interaction as a means of human-computer communication. The project also emphasizes the importance of user-centered design and usability testing to ensure that the Magic Wand is intuitive, easy to use, and accessible to a wide range of users. By exploring the capabilities of the Arduino Nano 33 BLE Sense board and developing a gesture recognition system, the Magic Wand project aims to inspire creativity, innovation, and exploration in the field of interactive technology and human-computer interaction. The Magic Wand project represents an exciting opportunity to experiment with sensor technologies, wireless communication, and gesture recognition algorithms and to create a unique and engaging user experience that combines physical movement with digital interaction. The project has the potential to open up new possibilities for interactive technology and to inspire future projects and applications that leverage the capabilities of the Arduino Nano 33 BLE Sense board and other sensor-based platforms.

2.2. Problem

The project represents an exciting challenge in interactive technology using board Arduino Nano 33 BLE Sense, whereby the project derived is referred to as the Magic Wand. This project aims to develop and implement a handheld device similar to the magic wand by using the board Arduino Nano 33 BLE Sense due to its characteristics such as wireless communication implementation. The wand is expected to interact

with a suitable system, such as a computer, through the implementation of Bluetooth Low Energy (BLE) technologies. The challenge would be to develop a responsive and efficient gesture recognition system that can remap the wand movement of the user into actual commands. Moreover, the project might incorporate sensors like accelerometers or gyroscopes for accurately capturing and interpreting gestures. Thus, Successful implementation requires an understanding of programming Arduino, sensor integration, and BLE communication protocols. In this Magic Wand project, it will be essential to resolve the intricacies by documenting the board Arduino Nano 33 BLE Sense, sensor datasheets, and other relevant online resources covering gesture recognition algorithms. For this reason, the project not only comes up with a fantastic and exciting way of testing board Arduino Nano 33 BLE Sense capabilities but also opens new horizons in gesture-based human-computer interaction.

2.3. Data Acquisition

2.3.1. Database

In the lack of pre-existing databases adapted to our specific model, we developed a new database for our Magic Wand project. The information is organised as a.txt file and contains raw data from accelerometer readings taken by the Arduino Nano 33 BLE Sense. Recognising the inherent heterogeneity in how people capture gestures, we asked three members of our group to contribute, resulting in a richer and more diverse dataset. Each participant scrupulously documented at least 50 trials of each gesture W, O, and L to ensure a complete representation of gesture variability. This strategy takes advantage of the distinct hand movements of three different users, improving the database's accuracy, efficiency, and efficacy. [**War:2020**]
In the next part of our project, named Data Preparation and Quality," we intend to process the raw data collected from the database. This includes identifying and resolving outliers as needed, as well as further refining the dataset to ensure optimal performance when training our machine learning model. The comprehensive quality of our dataset, enriched by diverse contributions and purposefully limited in size, provides a solid foundation for the project's succeeding phases, promising an effective and adaptable model for Magic Wand gesture identification. Detailed Information about database in teh section **??**.

2.3.2. Data Preparation

This section discusses how to shape our data and how we can use it for training. Our data model will recognise three gestures: wing, slope and ring. Since we do not intend to use existing databases to prepare the data, we will give multiple inputs for a single gesture and save it as model data for the gesture. When a gesture, for example - "wing" is waved using the wand, the accelerometer present in the microcontroller detects the movement by using the coordinates in the x,y and z axes. The exact process is repeated multiple times for the dataset to be as large as possible. Creating a database with more inputs allows the data model to be more accurate and provides better and more efficient results. The exact process is repeated for the remaining gestures, namely ring, slope and unknown. With the data available for the above gestures, we can use it to train a model. The available data is split into two sets, namely training data and test data.
The major challenge while preparing data can be the presence of outliers or anomalies with unexpected values. This can be overcome by feeding our model with more data so that the model is efficient.[**War:2020**]
The different steps involved in preparing data can be as follows according to [**War:2020**]

Understanding the problem The problem, which in our case is the collection of data consisting of gestures. The main problem is divided into many parts for easy data preparation.

Preparation of data The step deals with converting the raw data into machine learning language that the compiler can interpret.

Evaluation of data This step deals with evaluating the model after the collection of the necessary data. This data is then checked for correctness, and analysis is done according to the accuracy of the output obtained.

Finalizing the data The model is tested with various parameters and the data is validated. If the results obtained are according to our requirements, the resulting dataset is finalized.

The same steps are repeated for various gestures and the final database is created.

Preparation of Datasets for the Magic Wand

Since the amount of data required for the creation of a database is very small, the database will be prepared by us. The first set of data that we are trying to capture the movement is for the gesture "W". The steps involved in capturing a W are mentioned below. [**War:2020**]

Figure 2.1.: **Wing Gesture [War:2020]**

- The device is first moved down and to the right.
- Then the device is moved up and to the right.
- The device is then moved down and to the right.
- The device is then moved up and to the right again.

Shows a sample of real data captured during the "wing" gesture, measured in milli-Gs. The process involved in capturing a ring is as follows. [**War:2020**]

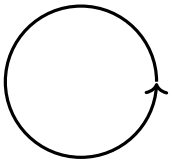


Figure 2.2.: **Ring Gesture [War:2020]**

- Trace a clockwise circle using the wand.
- Aim again to take around a second to perform the gesture.

The steps involved in waving the slope gesture are mentioned below [**War:2020**]:

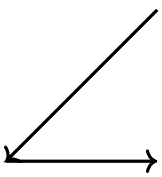


Figure 2.3.: **Slope Gesture [War:2020]**

- The device is first moved down and to the left.
- Then the device is moved to the right.
- Finally you should get a corner of the triangle as shown in the figure.

The process is repeated until around 15 readings are captured by the wand for Data Preparation and Transformation.

In order to consider the unknown readings, we will be carrying out another set of procedures to feed with unknown data. This would help the wand recognize any unknown gesture and give the output that it is false. [**War:2020**]

All readings are recorded in a unique text file for each gesture. The files `.txt` [**DataPrep:21**] contain raw accelerometer data that would later be transformed into machine learning language that should be interpreted by the compiler. [**War:2020**]

Since no existing databases exist for the model, a new database will be created.The database is a .txt file which contains raw data from the accelerometer readings recognized by the board Arduino Nano 33 BLE Sense. Since the way a person records the gesture varies from one to another, the gestures of three people are recorded as we are a group of three members. In order to the database to have a vast amount of data, each person will record at least 50 trials for each gesture. In order for the database to be more accurate data from both hands of the three users were used. This would improve the efficiency and effectiveness of the database making it more concrete. It is also possible to add further data to our model, improving it over time. The size of the database thus created will be petite, less than 2 MB. Obtaining sufficient data is one of the major concerns in a machine learning project and the amount of data involved in our data set is very small. The data thus created will be further used in the next step “Data Preparation and Transformation” where this raw data is further processed and the outliers are identified and removed if necessary.

2.4. Data Quantity

Edge Computing

The placement of computer and storage resources at the point where data produced is known as edge computing. This computes and stores the data source at the network edge, which is optimal. In other words, instead of sending raw data to a main data centre for processing and analysis, this work is done right where the data is generated. Edge computing is used to handle discrete tasks like determining if someone answered "yes" and responding appropriately. Instead of comparing the data with that on the web, the audio analysis is done on the edge. This drastically decreases expenses and complexity while also reducing the risk of data breaches.[**Big:2021**]

Edge computing has gained traction as a viable solution to a variety of issues related to transporting the massive amounts of data that today’s businesses generate and consume. It’s not just a matter of quantity, it’s also a matter of time; applications increasingly rely on processing and responses that are time-sensitive.The analysis and analysis of data produced by connected devices is made possible by edge computing, a crucial element of the Internet of Things (IoT).[**Haz:2021**] claims that edge computing enables data collecting, processing, and analysis at the network’s edge, facilitating in-the-moment decision-making and minimizing the need to send data to a central point for processing. Edge computing reduces processing times for data, allowing businesses to react swiftly to shifting operational circumstances.

2.5. Data Quality

Creating a magic wand with board Arduino Nano 33 BLE Sense involves integrating various sensors components and programming logic to ensure data quality for a successful project you need to consider the reliability and accuracy of the data collected from sensors here are some key considerations[**Daity:21**].

Sensor Calibration

Sensor calibration calibrate sensors such as accelerometers gyroscopes or magnetometers to ensure accurate readings understand the sensor s specifications and calibrate them accordingly to reduce errors in measurements

Sensor Fusion

Sensor fusion use sensor fusion techniques to combine data from multiple sensors this can improve the accuracy and reliability of the data implement sensor fusion algorithms to obtain a more comprehensive understanding of the wand s orientation and movement

Wireless Communication Reliability

Wireless communication reliability if your magic wand involves wireless communication ensure the bluetooth low energy ble connection is stable and reliable implement error checking mechanisms to handle potential data transmission issues [**Gomez:2012**]

Code Optimization

Write efficient and optimized code to minimize delays and improve the real-time response of the magic wand. Optimize algorithms to reduce processing time and enhance overall system performance.

2.6. Data Relevance

Real-Time Machine Learning

Real Time can be considered as a way of training the model by making it run through live data constantly in order to improve the model. This contradicts the traditional way of machine learning when machine learning engineers were dependent already existent data inputs for creating the model. A method by which we can implement real time learning in to a machine learning model is by continuously feeding it with a data stream and improving the model over time. This is achieved by using an event driven architecture. Event driven architecture is a modern design and development approach that centers about the events occurring in the model. Event-driven architectures create, detect, consume, and react to events.[**Haz:2021**]. By using a scalable event driven architecture, large number of events in real time can be responded, to which are suitable for loosely coupled softwares(For eg: web services). They also work well with unpredictable and non linear events by making it versatile. [**Haz:2021**]

2.7. Outliers

Outliers refer to exceptional or irregular data points that deviate significantly from the expected sensor readings or user interactions. These anomalies can arise from various sources such as sudden movements, extreme sensor values, interference, or unexpected user inputs.outliers in sensor data might occur if the accelerometer or gyroscope

registers an unusually rapid or erratic movement that does not align with typical usage patterns. Similarly, outliers in user interaction may manifest as unintended button presses or variations in touch sensor sensitivity.
Detecting and handling outliers is crucial for ensuring the accuracy and reliability of the magic wand's functionality. Strategies may involve implementing filtering algorithms, calibration techniques, and debouncing mechanisms to mitigate the impact of outliers on sensor readings and user inputs.[**Munoz:2019**]

2.8. Anomalies

Creating a magic wand with Arduino Nano 33 BLE involves considering potential anomalies or unexpected behaviors that might arise during the development and usage of the wand. The major challenge while preparing data can be the presence of outliers or anomalies where there are unexpected values. This can be overcome by feeding our model with more data so that the model is efficient.[**War:2020**]

Sensor Anomalies

- Accelerometer/Gyroscope Drift - Sensors like accelerometers and gyroscopes may experience drift over time, leading to inaccuracies in orientation tracking. Calibration techniques can help mitigate this issue.[**Chow:2021**]
- Magnetic Interference - Magnetometers can be affected by nearby magnetic fields, leading to anomalies in readings. Shielding or compensation algorithms may be required.

Power Anomalies

- Unexpected Power Drain - Identify and address any unexpected power consumption patterns. Implement power-efficient code and periodically monitor power usage to detect anomalies.
- Battery Voltage Fluctuations - Voltage fluctuations in the power supply can affect the stability of the Arduino Nano 33 BLE. Implement voltage regulation and monitoring to handle such anomalies.

Firmware and Software Anomalies

- Memory Issues - Anomalies may occur due to memory limitations on the Arduino Nano 33 BLE. Monitor and optimize memory usage to avoid crashes or unexpected behavior.
- Firmware Bugs - Identify and address potential bugs in your firmware. Regular testing, debugging, and code reviews can help mitigate software anomalies.

3. Hardware Description

3.1. Board Arduino Nano 33 BLE Sense

The Arduino Nano 33 BLE Sense board serves as a prime example of edge computing. It integrates multiple sensors alongside BLE connectivity and wireless communication features [**Raj:2019**], making it suitable for a wide range of IoT and wearable applications. Equipped with sensors for temperature, pressure, magnetometry, acceleration, and gyroscopic measurements [**Raj:2019**], the board also includes a microphone and a proximity sensor. Its built-in BLE capabilities enable seamless interaction with other BLE-compatible devices, such as smartphones, facilitating remote data collection and control [**Bagur:2023**]. At Magic Wand, our focus lies in leveraging the Accelerometer, Gyroscope, and Magnetometer sensors. The Arduino Nano 33 BLE Sense stands out as a versatile board, ideal for applications requiring sensor integration and wireless communication in IoT and wearable technologies [**Raj:2019**].
The Arduino Nano 33 BLE Sense is a compact, microcontroller-based development board designed to operate at a maximum of 3.3V. It is essential to avoid exceeding this voltage on its digital and analog pins [**Arduino:2021**]. The board features a Bluetooth Low Energy (BLE) module, making it suitable for IoT applications [**Bagur:2023**]. Powered by an nRF52840 processor, it boasts a 64 MHz clock speed, 256 KB of SRAM, and 1 MB of flash memory. Additionally, it includes 14 digital I/O pins, eight of which are analog input pins for connecting external components and sensors. The board's power consumption is remarkably low, drawing only 10 mA per I/O pin [**Arduino:2021**].
The Arduino Nano 33 BLE Sense supports high-level programming languages similar to C/C++ [**Emeritus:2023**]. Programming for microcontrollers, including Arduino, is done on a host computer. Code is written and compiled using Arduino's Integrated Development Environment (IDE) before being uploaded to the microcontroller for execution. To connect the board, a micro-USB cable is used, linking the board to a laptop. Once powered, a green LED next to the micro-USB port indicates a successful connection.
Additionally, Arduino offers several versions of the Nano 33 BLE board. This includes the Nano 33 BLE Sense and Nano 33 BLE Sense Lite. The Lite version lacks the HTS221 temperature and humidity sensor, instead featuring the LPS22HB pressure sensor, which can measure temperature but not humidity [**Arduino:2022**]. Since our project focuses on the IMU sensor for gesture recognition, both versions are suitable for use. The following descriptions and hardware tests are based primarily on the Arduino Nano 33 BLE Sense.

3.2. Interfaces

3.2.1. Board Arduino Nano 33 BLE Sense: Components Overview

The Arduino Nano 33 BLE Sense is equipped with several embedded sensors, including humidity, temperature, barometric pressure, proximity, and a microphone. These built-in components allow the board to be utilized for various practical applications without the need for additional circuitry [**Arduino:2021**]. It supports standard communication protocols such as UART, I2C, and SPI, enabling seamless interaction with external circuits and sensors for data exchange. The board includes a micro-USB

Figure 3.1.: **Top View of Board Arduino Nano 33 BLE Sense** [**Arduino:2023**]

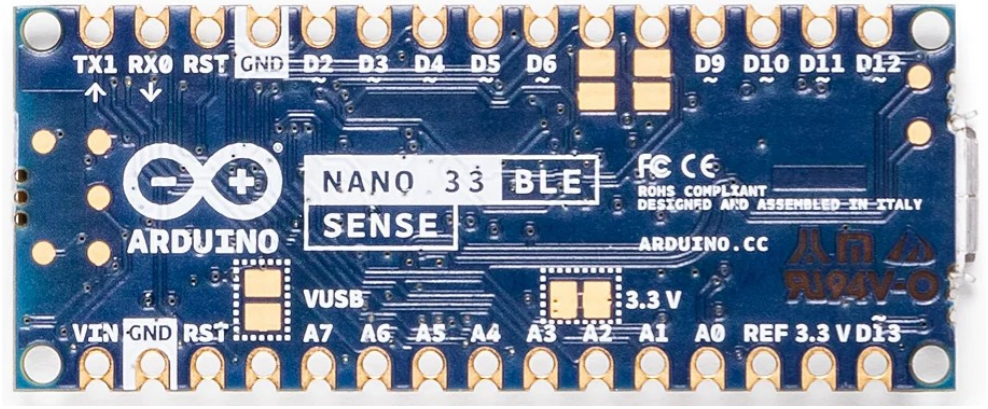


Figure 3.2.: **Bottom View of Board Arduino Nano 33 BLE Sense** [**Arduino:2023**]

port for connecting to a laptop or desktop, which serves as both a power supply and a medium for file or data transfer via serial communication. While operating the board, it is crucial to ensure that no more than 3.3V is applied to its pins, as exceeding this voltage can result in permanent damage [**Arduino:2021**].

Below is a detailed description of each sensor embedded on the board:

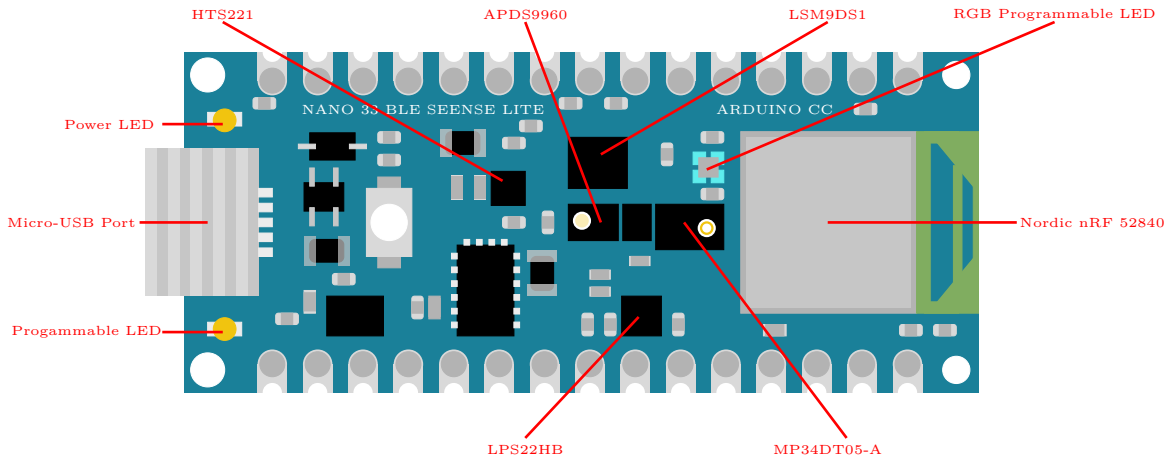


Figure 3.3.

- LSM9DS1 - Inertial Measurement Unit (IMU) features a 3D accelerometer, gyroscope and magnetometer and allows you to detect orientation, motion or vibrations in your project [**Alushi:2023**].
- APDS-9960 - The APDS-9960 chip allows for measuring digital proximity and ambient light as well as for detecting RGB colors and gestures [**Avago:2015**].
- LPS22HB - The LPS22HB picks up on barometric pressure and allows for a 24-bit pressure data output between 260 and 1260 hPa. This data can also be processed to calculate the height above sea level of the current location [**Stm:2017**].
- HTS221 - The HTS221 capacitive digital sensor measures relative humidity and temperature. It has a temperature accuracy of ± 0.5 °C (between 15-40 °C) and is perfectly suited to detect ambient temperature [**Stm:2023**].
- MP34DT05 - The MP34DT05 microphone allows you to capture and analyze sound in real-time and can be used to create a voice interface for your project[**Stm:2021**].
- USB port - USB port allows you to connect Arduino Nano 33 BLE sense to your machine.
- LEDs - There are 3 different LEDs that can be accessed on the Nano BLE Sense: Rot-Grün-Blau (RGB)(Programmable LED), the built-in LED(Programmable LED) and the power LED

LEDs function in Board Arduino Nano 33 BLE Sense

Apart from the Power LED indicate the board is powered, there are two LEDs in the board: Programmable LED(orange) and the RGB Programmable LED. The orange LED sparkles when it is connected to the computer. The RGB LED can be used during the creation of the actions that we perform. For example, in our project, each gesture can use one color to represent a specific gesture, indicating as the recognizing function.

3.3. Arduino Nano 33 BLE Pin Configuration

The **Arduino Nano 33 BLE** is an advanced version of the Arduino Nano board that is based on a powerful processor, the nRF52840. The following is the pin configuration of the board:

3.3.1. Pin Configuration

Digital Pins:

- The board has **14 digital I/O pins** that receive only two values: HIGH or LOW.
- These pins can function as input or output based on the requirement.
- When the pins receive 5V, they are in a HIGH state; when they receive 0V, they are in a LOW state.

Analog Pins:

- The board has a total of **8 analog pins** (A0–A7).
- These pins measure analog voltage ranging between 0 to 5V and can get any value as opposed to digital pins, which only receive HIGH or LOW values.

PWM Pins:

- All digital pins can be used as PWM pins.
- These pins generate analog results using digital means.

SPI Pins:

- The board supports the **Serial Peripheral Interface (SPI)** communication protocol.
- SPI is used to communicate between the controller and peripheral devices such as shift registers and sensors.
- Two pins, **MISO** (Master Input Slave Output) and **MOSI** (Master Output Slave Input), are used for SPI communication.

I2C Pins:

- The board supports the **I2C communication protocol**, a two-wire protocol.
- It includes two pins: **SDL** and **SCL**.

UART Pins:

- The board features the **UART communication protocol** for serial communication.
- It includes two pins: **Rx** (receiving pin) and **Tx** (transmitting pin).

External Interrupts Pins:

- All digital pins can be used as external interrupt pins.
- This feature allows the main running program to be interrupted and handle emergency instructions.

LED at Pin 13 and AREF Pin:

- There is an **LED connected to pin 13** of the board.
- The **AREF pin** is used as a reference voltage for input voltage.

3.4. Data Quality in Hardware Description

In the hardware description of our Magic Wand project with Arduino Nano 33 BLE Sense, we consider the following aspects related to data quality:

1. **Sensor Accuracy:** The sensors embedded in the Arduino Nano 33 BLE Sense board deliver precise measurements of environmental parameters such as motion, orientation, temperature, humidity, and pressure [**Dhow:2024**].

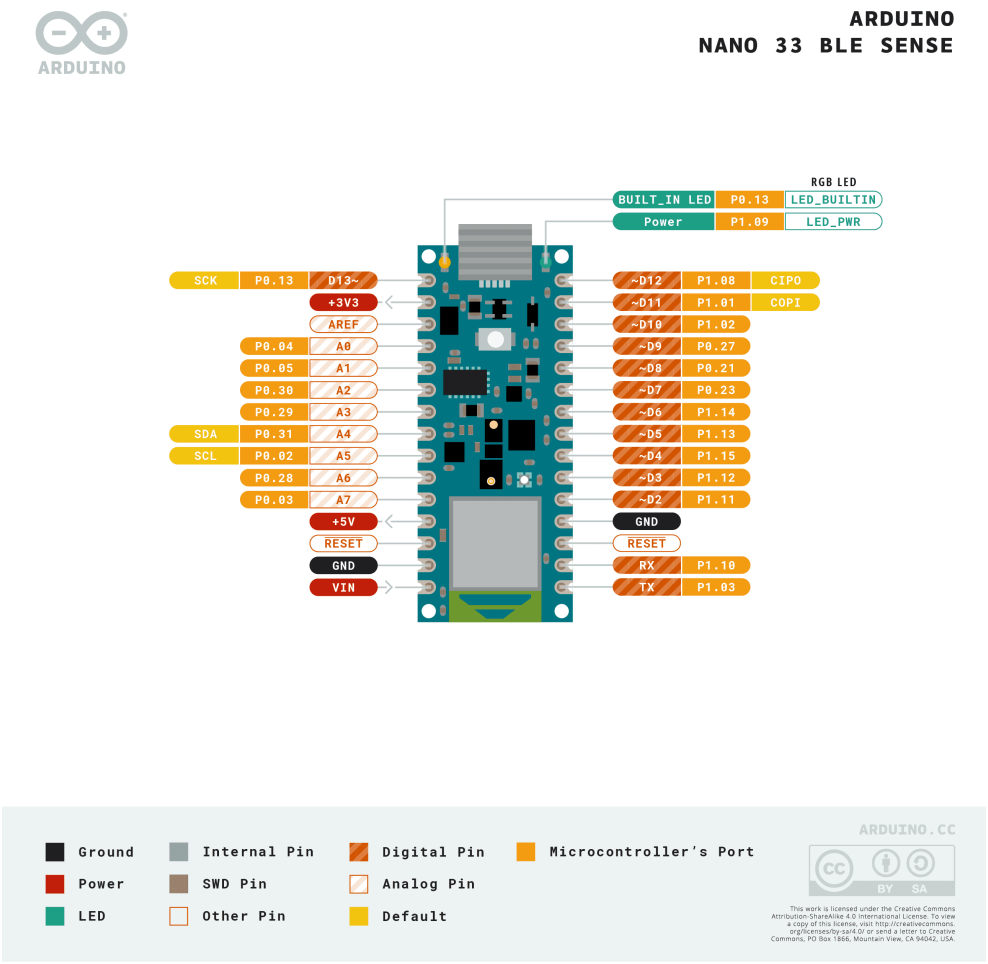


Figure 3.4.: **Arduino Nano 33 BLE Pin Configuration** [**Arduino:2023**]

2. **Resolution:** Each sensor has a defined resolution, representing the smallest detectable input change. For instance, the accelerometer's resolution is <insert value>, allowing detection of subtle motion variations.

3. **Sampling Rate:**The board supports high sampling rates, enabling real-time data acquisition at <insert frequency> Hz. This is essential for applications requiring continuous and responsive data collection.

4. **Noise Level:** Sensor data may exhibit noise due to factors such as environmental conditions, sensor imperfections, or electromagnetic interference. The board minimizes noise using calibration and filtering techniques, ensuring high data fidelity.

5. **Calibration:** Calibration processes are applied to enhance the accuracy and reliability of sensor readings. These procedures involve fine-tuning sensor settings and using correction factors to address systematic errors [**Edm:2015**].

6. **Data Transmission:** Sensor data is transmitted from the Arduino Nano 33 BLE Sense board to other devices using wireless communication protocols such as Bluetooth Low Energy (BLE). Data transmission is efficient, ensuring minimal latency and reliable delivery.

7. **Data Integrity:** To maintain data integrity during transmission and processing, error detection and correction mechanisms are implemented. These measures help identify and resolve issues like data loss or corruption [**TensorFlow:2023**].

8. **Power Consumption:** The Arduino Nano 33 BLE Sense is designed for low power consumption, making it ideal for battery-powered applications. Its optimized power management extends battery life without compromising continuous data collection.

By addressing these aspects of data quality in the hardware description, we ensure the reliability and accuracy of sensor data used in our Magic Wand project.

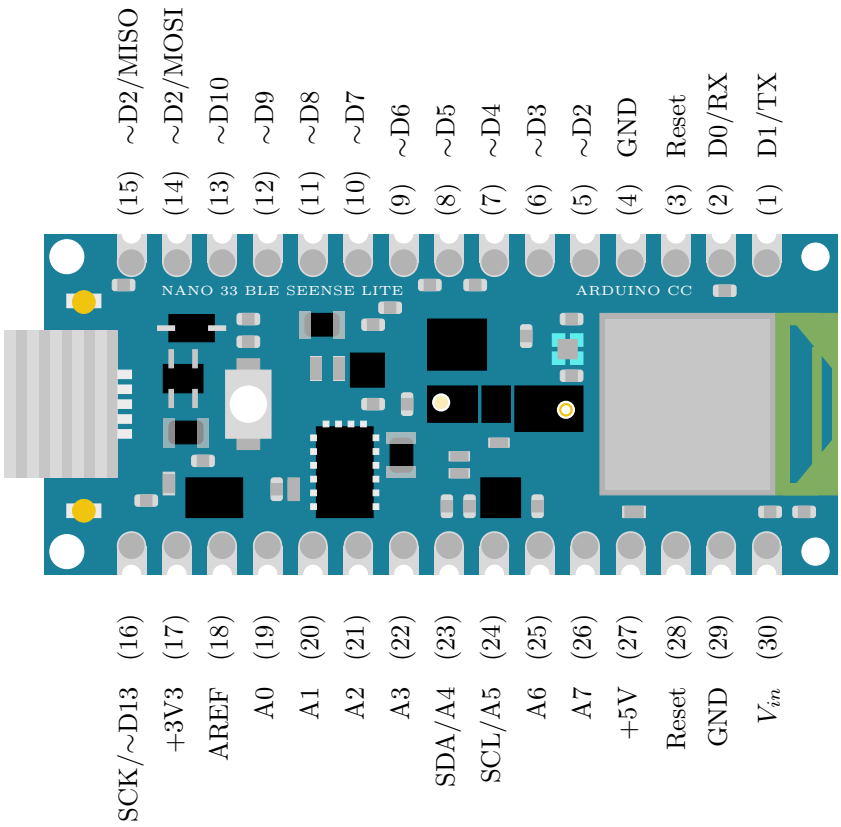


Figure 3.5.: Pin assignment of the Arduino Nano 33 BLE Sense; note that the orange built-in LED is connected to pin D13 and the power LED to pin D1. The built-in RGB LED occupies pins D18, D19 and D20.

3.4.1. Specifications of Arduino Nano 33 BLE Sense

NINA B306 Module

1. Processor:
   - 64 MHz Arm® Cortex-M4F (with FPU)
   - 1 MB Flash + 256 KB RAM

2. Bluetooth® 5 multiprotocol radio:
   - 2 Mbps
   - +8 dBm TX power

- -95 dBm sensitivity
- 4.8 mA in TX (0 dBm)
- 4.6 mA in RX (1 Mbps)
- Integrated balun with 50 Ohm single-ended output
- IEEE 802.15.4 radio support

3. Peripherals:
   - 12 Mbps USB
   - NFC-A tag
   - Arm CryptoCell CC310 security subsystem
   - QSPI/SPI/TWI/I2S/PDM/QDEC
   - 32 MHz SPI
   - Quad SPI interface 32 MHz
   - 12-bit 200 ksps ADC
   - 128 bit AES/ECB/CCM/AAR co-processor

LSM9DS1 (9 axis IMU)

[**Warden:2020**]

1. 3 acceleration channels, 3 angular rate channels, 3 magnetic field channels
2. ±2/±4/±8/±16 g linear acceleration full scale
3. ±4/±8/±12/±16 gauss magnetic full scale
4. ±245/±500/±2000 dps angular rate full scale
5. 16-bit data output

3.5. Data quantity in Hardware Description

1. **Sensor Data**: The Arduino Nano 33 BLE Sense board is equipped with a variety of sensors, including an accelerometer, a gyroscope, and a magnetometer. These sensors generate a large amount of data as you wave the magic wand. For instance, in one experiment, a window size of 2 seconds was used, which means 200 rows of accelerometer data or 600 values of x, y, and z acceleration axis were fed into the model. [**Miller:2022**]
2. **Data Processing Capacity**: The Arduino Nano 33 BLE Sense board has a 32-bit ARM Cortex-M4 CPU running at 64 MHz, which allows it to process a significant amount of sensor data in real-time.
3. **Memory**: The board has 1MB of flash memory and 256KB of SRAM. This memory is used to store the program code and handle runtime operations, including storing sensor data and machine learning model parameters.
4. **Machine Learning Model**: The size of the machine learning model used in the project also affects the quantity of data. The model needs to be small enough to fit into the board's memory along with the program code.
5. **Data Transmission**: The board also has a built-in Bluetooth Low Energy module, which can be used to transmit sensor data to another device for further processing. [**TensorFlow:2023**]

3.6. Constraints

Arduino Nano 33 BLE Sense

- Operating Voltage: 3.3V
- Power Consumption:
  - Maximum 15mA in low power mode
  - Maximum 60mA in active mode
- Operating Temperature Range: -40°C to 85°C
- Memory Constraints: 1 MB Flash + 256 KB RAM
- Communication Interfaces: USB, Bluetooth 5, NFC-A, SPI, I2C, QSPI, etc.[**Arduino:2015**]

Sensors

- Accelerometer:
  - Measurement Range: ±8g
  - Operating Temperature Range: -40°C to 85°C
- Gyroscope:
  - Measurement Range: ±2000 dps
  - Operating Temperature Range: -40°C to 85°C [**Arduino:2021**]

Actuators

- RGB LEDs:
  - Power Requirements: Voltage, Current
  - Operating Temperature Range
- Buzzer/Speaker:
  - Voltage, Current, Sound Output Levels

Power Supply

- Input Voltage Range: Specify the acceptable input voltage range.
- Power Consumption: Estimate the overall power consumption of the system.

Physical Constraints

- Size and Dimensions: Ensure compatibility with the project enclosure or housing.
- Mounting Requirements: Specify any specific mounting requirements for the components.

Environmental Constraints

- Environmental Protection: Ensure components are suitable for the intended environmental conditions (e.g., moisture resistance, dust resistance).
- Operating Conditions: Specify any limitations or special considerations for operating in certain environments.[**Arduino:2021**]

3.7. Dimensions of the Arduino Nano 33 BLE Sense

The Arduino Nano 33 BLE Sense is a highly compact development board, measuring just 45mm x 18mm. Its small form factor makes it particularly well-suited for wearable devices and other space-constrained applications. Despite its size, the board is equipped with a wide range of sensors and features, offering versatility for various projects.

It is important to note that these dimensions apply to the board without headers. If you are using a version with pre-soldered headers or attaching additional components, the overall dimensions of your hardware setup may change. For precise measurements, always consult the specifications of your specific board and components [**Arduino:2023**].

3.8. Sensor Accelerometer, Gyroscope, and Magnetometer LSM9DS1

Magic Wand gesture detection is mainly based on the sensor LSM9DS1 at the board. It is a system-in-package featuring a 3D digital linear acceleration sensor, a 3D digital angular rate sensor, and a 3D digital magnetic sensor. A tiny device called an Inertial Measurement Unit (IMU) in the sensor is used to detect the motions physically [**Alushi:2023**]. It has several components, including magnetometer, gyroscope, and accelerometer. By monitoring acceleration and angular velocity changes, the IMU sensor is critical for identifying an object's orientation and movement in real-time [**Alushi:2023**].

Example Code for Arduino Nano 33 BLE Sense

Below is a simple example demonstrating how to use the built-in microphone on the Arduino Nano 33 BLE Sense:

```
1000  // Include the PDM library
      #include <PDM.h>
1002
      // Buffer to store the microphone data
1004  short sampleBuffer[256];

1006  // Variable to store the sound level
      int soundLevel = 0;
1008
      // Callback function for PDM data
1010  void onPDMdata() {
        // Read the PDM data
1012    int bytesAvailable = PDM.available();
        PDM.read(sampleBuffer, bytesAvailable);
1014
        // Calculate the sound level
1016    soundLevel = 0;
        for (int i = 0; i < bytesAvailable / 2; i++) {
1018      soundLevel += abs(sampleBuffer[i]);
        }
1020    soundLevel /= bytesAvailable / 2;
      }
1022
      void setup() {
1024    // Initialize serial communication
        Serial.begin(9600);
1026    while (!Serial);

1028    // Initialize PDM with a sample rate of 16 kHz and 16-bit
      resolution
        PDM.begin(1, 16000);
1030    PDM.onReceive(onPDMdata);
      }
1032
      void loop() {
1034    // Print the sound level to the serial monitor
        Serial.println(soundLevel);
1036    delay(100);
      }
1038
```

Listing 3.1.: Simple example using of the builtin microphone of the Arduino Nano 33 BLE Sense

```
1000    // Include the ArduinoBLE library
        #include <ArduinoBLE.h>
1002
        // Create a BLE service
1004    BLEService batteryService("1101");

1006    // Create a BLE characteristic
        BLEUnsignedCharCharacteristic batteryLevelChar("2101", BLERead |
        BLENotify);
1008
        void setup() {
1010      // Initialize serial communication
          Serial.begin(9600);
1012      while (!Serial);

1014      // Set up the built-in LED pin
          pinMode(LED_BUILTIN, OUTPUT);
1016
          // Initialize BLE
1018      if (!BLE.begin()) {
            Serial.println("Starting BLE failed!");
1020        while (1);
          }
1022
          // Set the BLE local name and advertised service
1024      BLE.setLocalName("BatteryMonitor");
          BLE.setAdvertisedService(batteryService);
1026
          // Add characteristic to the service
1028      batteryService.addCharacteristic(batteryLevelChar);

1030      // Add the service and start advertising
          BLE.addService(batteryService);
1032      BLE.advertise();

1034      Serial.println("Bluetooth device active, waiting for connections
        ...");
        }
1036
        void loop() {
1038      // Wait for a BLE central to connect
          BLEDevice central = BLE.central();
1040
          if (central) {
1042        Serial.print("Connected to central: ");
            Serial.println(central.address());
1044
            // Turn on the built-in LED
1046        digitalWrite(LED_BUILTIN, HIGH);

1048        // Loop while the central device is connected
            while (central.connected()) {
1050          int battery = analogRead(A0);
              int batteryLevel = map(battery, 0, 1023, 0, 100);
1052
              Serial.print("Battery Level is now: ");
1054          Serial.println(batteryLevel);

1056          // Update the characteristic value
              batteryLevelChar.writeValue(batteryLevel);
1058
              delay(200);
1060        }

1062        // Turn off the built-in LED
            digitalWrite(LED_BUILTIN, LOW);
1064        Serial.print("Disconnected from central: ");
            Serial.println(central.address());
1066      }
        }
1068
```

Listing 3.2.: Example of a BLE Battery Monitor using Arduino Nano 33 BLE Sense

3.9. Inertial Measurement Unit (IMU)

3.9.1. Description

An Inertial Measurement Unit (IMU) is an electronic system that measures movement across multiple axes using three primary sensors: an accelerometer, a gyroscope, and a magnetometer.[**Qureshi:2017**] These sensors work together to provide data on acceleration, rotational motion, and magnetic fields, making IMUs essential for a wide range of applications such as navigation, fitness tracking, and robotics.[**Qureshi:2017**] General IMUs have become increasingly common in microcontroller projects, with some boards, such as the Arduino Nano 33 BLE Sense, featuring integrated IMUs for seamless development of motion-sensitive applications.[**Zhou:2020**]

3.9.2. Specific Sensors

Continuous Operation ("Always-On" Mode) The LSM9DS1 supports an "Always-On" mode, ensuring continuous operation even when the main system is in a low-power state.[**Zhou:2020**] This is critical for uninterrupted motion monitoring in devices like the Magic Wand, where gestures must be tracked instantly. With a power consumption of just 0.55 mA in high-performance mode, it provides precise and reliable motion data while preserving battery life.[**Zhou:2020**]
Tilt Detection Using the accelerometer, the IMU can detect orientation changes with minimal power usage. Tilt detection is particularly useful for identifying subtle shifts in position, enhancing the responsiveness of gesture-based controls.[**Zhou:2020**]
Significant Motion Detection The accelerometer enables significant motion detection (SMD), which recognizes large-scale movements. SMD can be used to activate specific device functions, such as waking the system from sleep mode or triggering predefined actions based on significant gestures.[**Zhou:2020**]
The LSM9DS1 is a versatile and energy-efficient IMU with advanced sensing capabilities, supporting a range of applications where motion detection, orientation tracking, and gesture recognition are essential. Its compact design, low power consumption, and robust environmental tolerance make it a reliable choice for portable, battery-powered devices like the Magic Wand.[**Zhou:2020**]
An IMU consists of three sensors that measure an object's orientation, position, vibration and movement in 3D space in real-time. These sensors are typically arranged in a pattern, including a tri-axial accelerometer, gyroscope, and magnetometer[**Ahmad:2013**].
An accelerometer is used to measure the change in velocity of a moving or vibrating object[**Ahmad:2013**]. Gyroscope measures the angular rotation[**Ahmad:2013**]. A Magnetometer is used to measure yaw angle rotation, calibrating to the gyroscope data to improve the big drift issue[**Ahmad:2013**].
The accelerometer functions by gauging the acceleration (Ax, Ay, Az), which denotes the rate of acceleration change over time. The acceleration values along each axis are conventionally expressed in units of meters per second squared (m/s^2) [**Vernier:2023**]. To illustrate, if the LSM9DS1 records an acceleration of 15 m/s^2 along the x-axis, this signifies that the object under observation is experiencing a velocity increment of 15 meters per second every second in the x-direction.
A gyroscope, designed to quantify angular speed (ωx, ωy, ωz), serves as an indicator of the object's orientation change rate along each axis. Angular velocity along each axis is commonly expressed in degrees per second ($\circ$/s) [**Zhuang:2020**]. For instance, when the LSM9DS1 records an angular velocity of 100$\circ$/s along the z-axis, it signifies that the measured object is undergoing rotation at a rate of 100 degrees per second around the z-axis. Refer to the accompanying figure for visualization.
In the context of a magnetometer, the x, y, and z axes (Mx, My, Mz) typically delineate the three-dimensional space in which the magnetic field is being assessed. The x-axis typically corresponds to the horizontal component of the magnetic field, the y-axis

signifies the vertical component, and the z-axis reflects the magnetic field strength [**Kostiainen:2023**]. This three-dimensional measurement provides a comprehensive understanding of the magnetic field in the surrounding space.

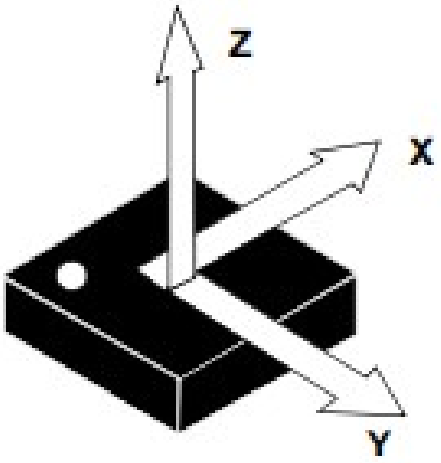


Figure 3.6.: **Accelerometer Accerlerations Directions** [**Stm:2015**]

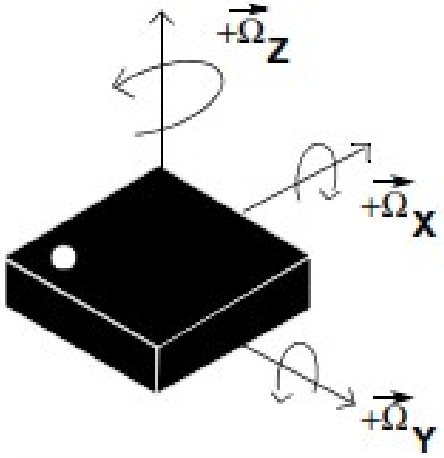


Figure 3.7.: **Gyroscope Angular Directions** [**Stm:2015**]

IMUs can be classified into two main categories based on the type of sensors used: Microelectromechanical Systems (MEMS) IMUs and Fiber Optic Gyro (FOG) IMUs. MEMS IMUs use small mechanical sensors that are etched onto a silicon chip, while FOG IMUs use optical fibers to measure angular velocity. MEMS IMUs are typically smaller and less expensive but are also less accurate and have a shorter lifespan compared to FOG IMUs[**Deppe:2017**].
IMU is sent to a microcontroller or computer, and software is written to read the data from the sensors and perform the necessary calculations. There are also many libraries and software packages available that can simplify the process of working with IMUs. IMUs are commonly used in a variety of applications, Industry Quality Control, Medical Rehabilitation, Robotics, Navigation Systems, Sports Learning and Augmented Reality Systems since they are essential to accurately measure the movement and orientation of the device for further function development [**Ahmad:2013**]. In general, they are

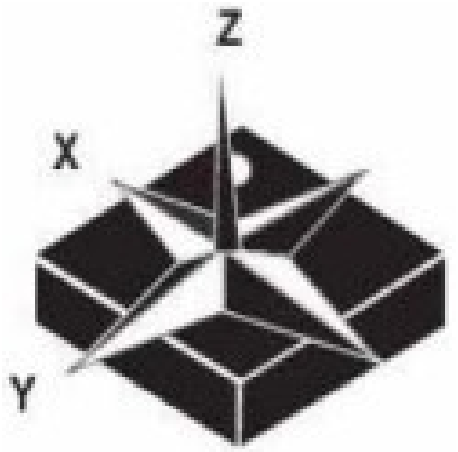


Figure 3.8.: **Magnetometer Directions**
[**Stm:2015**]

used for:
Orientation tracking: IMUs can follow the orientation of an object in space by using a combination of acceleration and angular velocity sensors. The data from these sensors can be used to calculate the object's orientation. Robotics technology is one of the regions that require linear and angular data for the movements so robots can perform daily routines and advanced tasks similar to humans [**Ahmad:2013**].
Motion sensing: Detecting motion and acceleration changes is also one function that IMUs can achieve. Ryan et al. suggested integrating a miniaturized IMU sensor into the ball, comprising a tri-axial gyroscope and tri-axial accelerometer. Their initial experiment employed ball kinematics knowledge to estimate the drift error arising from measurements [**McGinnis:2011**]. Subsequent research efforts enhanced data accuracy by cross-referencing the measurements with a high-speed motion analysis system consisting of 10 cameras (VICON) [**McGinnis:2012**].
Navigation System: IMUs are marked as an upgrade in the Global Positioning System (GPS) to navigate by combining data from the acceleration and angular velocity sensors with information for better data accuracy. Also, IMU can provide locations where no GPS signals are detected as an alternative option in the car and aircraft in case of emergency [**Ahmad:2013**].
However, IMUs face a significant challenge due to the susceptibility of the underlying gyroscopes and accelerometers to measurement errors. This poses a fundamental issue for all IMUs as gyroscopes inherently exhibit drift, and accelerometers may introduce inaccuracies, resulting in misestimations of orientation concerning gravity. Despite incorporating minor measurement errors, the guidance system continually integrates the calculated position results with the original position information (refer to trajectory calculation). Although individual errors may be minor, their persistence across positions leads to cumulative effects known as "drift." Over time, this accumulation causes a widening disparity between the system's perceived and actual locations, and it becomes impossible to eliminate these errors [**Harris:2023**]. Therefore, drift remains a fundamental challenge for any IMUs.

3.9.3. Specifications

- 3 acceleration channels, 3 angular rate channels, 3 magnetic field channels.
- $\pm 2/\pm 4/\pm 8/\pm 16$ g linear acceleration full scale.
- $\pm 4/\pm 8/\pm 12/\pm 16$ gauss magnetic full scale.

- $\pm 245/\pm 500/\pm 2000$ dps angular rate full scale.
- 16-bit data output.

3D digital linear acceleration sensor: Measures linear motion with a full scale of ±2g/±4g/±8g/±16g.
3D digital angular rate sensor: Tracks rotational motion with an angular rate range of ±245/±500/±2000 degrees per second (dps).
3D digital magnetic sensor: Detects magnetic fields with a full scale of ±4/±8/±12/±16 gauss.
This compact and versatile IMU includes both I²C and SPI serial interfaces and supports a power-down mode for smart power management. It is available in a plastic land grid array (LGA) package and operates reliably across a wide temperature range from -40 °C to +85 °C.

3.9.4. Libraries

A library refers to a collection of pre-written code that can be used by developers to perform specific tasks or functions without having to write the code from scratch. Libraries are designed to make the development process easier and more efficient by providing pre-built solutions to common programming challenges.

Wire.h

`Wire.h` is a library in Arduino that allows for communication between I2C devices. I2C stands for Inter-Integrated Circuit, which is a synchronous serial communication protocol used for connecting microcontrollers to peripheral devices. `Wire.h` provides functions for initializing the I2C bus, sending and receiving data over the bus, and managing multiple devices on the same bus. With this library, developers can easily connect multiple I2C devices, such as sensors or displays, to an Arduino board [**Ardc; Ari21**].

Kalman.h

`Kalman.h` is a library that implements the Kalman filter algorithm. The Kalman filter is a mathematical technique used to estimate the state of a system based on incomplete measurements. It is commonly used in control systems, robotics, and navigation applications to improve the accuracy of sensor measurements and reduce errors. `Kalman.h` provides a simple interface for developers to implement the Kalman filter in their Arduino projects [**Ard19; Fet21**].

Arduino_LSM6DSOX.h

`Arduino_LSM6DSOX.h` is a library that provides access to the LSM6DSOX accelerometer and gyroscope sensor on the Arduino Mbed OS Nicla Board. The LSM6DSOX sensor is a 6-axis sensor that can measure both linear acceleration and angular velocity. `Arduino_LSM6DSOX.h` provides functions for initializing the sensor, reading data from the sensor, and configuring the sensor parameters. With this library, developers can easily integrate the LSM6DSOX sensor into their Arduino projects and use the sensor data for various applications, such as gesture recognition or orientation detection [**Lib21**].

LSM6DSOXSensor.h

`LSM6DSOXSensor.h` is a library that provides an interface for interacting with the LSM6DSOX sensor. The LSM6DSOX is a 6-axis inertial measurement unit (IMU)

sensor that combines a 3-axis accelerometer and a 3-axis gyroscope in a single chip. It is commonly used in applications that require motion sensing and orientation tracking, such as robotics, drones, wearable devices, and Internet of Things (IoT) devices. The `LSM6DSOXSensor.h` library allows developers to easily interact with the LSM6DSOX sensor by providing functions and classes for configuring the sensor, reading raw sensor data, and performing sensor fusion to obtain calibrated accelerometer and gyroscope data, as well as derived data such as orientation, linear acceleration, and angular velocity. The library abstracts the low-level communication with the sensor, providing a higher-level API that simplifies the process of working with the sensor.

3.9.5. Calibration of Sensors

There are various methods to calibrate the sensors involved. It is necessary to know whether the sensor is balanced since the time between each calibration needs to be explicitly defined. Regular calibration should be done, especially when strange outputs are noticed. Some methods of calibration are briefed below:

Low and high limit method

The low and high limit method involves recording the minimum and maximum values on all three axes of a sensor by performing simple scratches or movements to determine their absolute extremes. The sensor is subjected to circular rotations along each axis multiple times [**Edm:2015**]. The center point is calculated as the midpoint between these recorded extremes. Increasing the number of rotations improves the chances of identifying the absolute peak values. Ideally, the center point should be close to zero, indicating minimal sensor offset. However, deviations may signal a hard iron offset, often caused by distortions from the Earth's magnetic field [**Edm:2015**]. This method assumes minimal soft iron distortion, which would otherwise alter the sensor readings. The absence of significant soft iron interference is typically confirmed by observing rounded outlines in the resulting data graph.
It is crucial to recognize that the low and high limit method requires recalibration periodically to maintain performance, as sensor components can drift or degrade over time [**Edm:2015**]. For devices powered by primary batteries, recalibration is particularly important after every battery replacement. This is because the battery often acts as a significant source of magnetic disturbance, and newly installed batteries may introduce different interference compared to the previous ones [**Edm:2015**].

FreeIMU Calibration Application Magnetometer

The **FreeIMU Calibration Application Magnetometer method** involves pre-processing raw magnetometer data by applying axis-specific gain correction to convert it into nanoTesla. The conversion formula for each axis is as follows:

$$Xm\text{-nanoTesla} = \text{rawCompass.m.x} \times \left(\frac{100000.0}{1100.0}\right)$$

$$Ym\text{-nanoTesla} = \text{rawCompass.m.y} \times \left(\frac{100000.0}{1100.0}\right)$$

$$Zm\text{-nanoTesla} = \text{rawCompass.m.z} \times \left(\frac{100000.0}{980.0}\right)$$

These scaling factors (e.g., $\frac{100000.0}{1100.0}$) should be replaced with sensor-specific values to ensure accurate conversion. The processed data is then saved in a file named `Mag-raw.txt`, which can be opened using the **Magneto program**. Magneto generates twelve calibration parameters to correct for various sensor errors, including bias, hard iron distortion, scale factor errors, soft iron distortion, and misalignment [**Edm:2015**].

Additionally, this method can be applied to accelerometers. By pre-processing raw accelerometer data while accounting for bit depth and G sensitivity, the data can be converted into milliGalileo (mGal). A gravitational field "norm" value of 1000 mGal can also be used to refine calibration [**Edm:2015**]. This approach ensures improved accuracy and reliability of sensor measurements for both magnetometers and accelerometers.

3.9.6. Code

```
1000  // Include the necessary libraries
      #include <Wire.h>
1002  #include <MPU6050.h>

1004  // Create an MPU6050 object
      MPU6050 mpu;
1006
      // Setup function
1008  void setup() {
        // Initialize I2C communication and Serial communication
1010    Wire.begin();
        Serial.begin(9600);
1012
        // Initialize the MPU6050
1014    mpu.initialize();

1016    // Check if the MPU6050 is connected
        if (mpu.testConnection()) {
1018      Serial.println("MPU6050 connection successful");
        } else {
1020      Serial.println("MPU6050 connection failed");
        }
1022  }

1024  // Loop function
      void loop() {
1026    // Declare variables for accelerometer and gyroscope data
        int16_t ax, ay, az;
1028    int16_t gx, gy, gz;

1030    // Get the motion data from the MPU6050
        mpu.getMotion6(&ax, &ay, &az, &gx, &gy, &gz);
1032
        // Print the accelerometer and gyroscope data
1034    Serial.print("a/g:\t");
        Serial.print(ax); Serial.print("\t");
1036    Serial.print(ay); Serial.print("\t");
        Serial.print(az); Serial.print("\t");
1038    Serial.print(gx); Serial.print("\t");
        Serial.print(gy); Serial.println(gz);
1040
        // Delay for 100ms before the next reading
1042    delay(100);
      }
1044
```

Listing 3.3.: Example of interfacing MPU6050 with Arduino for motion data

3.9.7. Applications

The IMU sensor on the Arduino Nano 33 BLE Sense can be used in the following applications:

- **Gesture Recognition**: Use accelerometer and gyroscope data to detect tilt, shake, or rotation gestures.
- **Fitness Tracker**: Monitor motion and orientation to calculate steps, measure speed, or track physical activities.
- **Robot Navigation**: IMU data can help track robot movement and determine orientation changes.
- **Virtual Reality**: IMUs provide orientation data for head tracking in VR applications.

3.9.8. Tests

Tilt Measurement Test

- Use the accelerometer data to calculate the tilt angle of the sensor relative to the ground.
- Compare the calculated tilt angle with a known reference (e.g., a protractor) to verify accuracy.

Motion Tracking Test

- Use both accelerometer and gyroscope data to track the sensor's motion in 3D space.
- Compare the tracked motion with a known reference (e.g., a motion capture system) to verify accuracy.

Environmental Tests

- **Temperature Stability Test**: Place the sensor in different temperature environments (e.g., cold, room temperature, hot) and record the readings. Verify that the sensor maintains accuracy across different temperatures. Any significant deviation may require temperature compensation.
- **Vibration Test**: Subject the sensor to different levels of vibration and record the readings. Verify that the sensor maintains accuracy under vibration. This is particularly important for applications like drones or vehicles.

3.9.9. Further Readings

- LSM9DS1 Datasheet - STMicroelectronics: LSM9DS1 Datasheet
- IMU Testing and Calibration: IMU Testing and Calibration

3.10. USB Cable

3.10.1. USB Type A Male to Micro-B Male

Professional USB 2.0 Type A Male to Micro-B Male Cable for High-Performance Commercial AV and IT Applications

- Supports data rates of up to **480Mbps**.
- Robust PVC housing with gold-plated contacts and nickel-coated connector sleeves.
- 2-fold shielded cable with corrosion-resistant, tinned copper conductor.
- **USB 2.0**, 480Mbps.

3.10.2. Cable Lines Concept

Cable Lines stands for the concept of contemporary, wired connectivity solutions from **Lindy**. The **Anthra Line USB 2.0 Type A Male to Micro-B Male Cables** from the Cable Line concept are the professional choice when it comes to realizing connections for the highest resolutions in commercial AV and IT applications.
The Anthra Line USB 2.0 cables feature:

- **2-fold shielding** and tinned copper conductors for the highest and lossless transmission performance.
- Permanent corrosion resistance and maximum reliability guaranteed by high--quality, gold-plated contacts and nickel-plated connector sleeves.

3.10.3. Key Features

- Data transfer speeds of up to **480Mbps** enable fast and smooth transfer of large volumes of data.

3.10.4. Specifications

General

- **Type:** USB 2.0 Cable
- **Execution:** Straight
- **Color:** Black
- **Material:** Plastic

Connections / Interfaces

- **Connection Input:** A-connector
- **Connection Output:** Micro-B connector

Metrics

- **Cable Length:** 0.20 m

Other

- **Specification:** USB 2.0
- **Manufacturer:** LINDY
- **Manufacturer's Article Number:** 36730
- **Tare:** 0.019 kg
- **RoHS:** Compliant
- **EAN / GTIN:** 4002888367301

3.11. Sticky Tape

tesa® WALLPAPER is a thin but tear-resistant masking tape that has been specially developed for use on very sensitive surfaces such as paper wallpaper or fine plaster. The wallpaper tape is easy to apply, allows precise work with clean paint edges, and can be removed within seven days without leaving any residue. Since the masking tape is suitable for all types of paint, it is the ideal basis for renovation work where you want to protect sensitive surfaces indoors.

Figure 3.9.: **USB**

3.11.1. Features

- Specially designed for sensitive surfaces such as paper wallpaper.
- Masking tape can be removed without leaving any residue for up to seven days.
- Solvent-free and primarily made from renewable raw materials.
- Suitable for all types of paint.

3.11.2. Dimensions

- **Width:** 25 mm
- **Length:** 25 m
- **Quantity:** 1 Roll

Metrics

- **Length:** 25 m
- **Width:** 25 mm

Other

- **Specification:** For sensitive substrates

Packaging

- **Packaging:** 1 roll

Manufacturer

- **Manufacturer:** TESA
- **Manufacturer's Article Number:** 56260-00000-03
- **Tare:** 0.0787 kg
- **RoHS:** Compliant
- **EAN / GTIN:** 4042448149992

Figure 3.10.: **Tapes**

4. Sensor/Actor

4.1. Accelerometer

4.1.1. Description

The accelerometer enables significant motion detection (SMD), which recognizes large-scale movements. SMD can be used to activate specific device functions, such as waking the system from sleep mode or triggering predefined actions based on significant gestures.[**Zhou:2020**]
The LSM9DS1 is a versatile and energy-efficient IMU with advanced sensing capabilities, supporting a range of applications where motion detection, orientation tracking, and gesture recognition are essential. Its compact design, low power consumption, and robust environmental tolerance make it a reliable choice for portable, battery-powered devices like the Magic Wand.[**Zhou:2020**]
An IMU consists of three sensors that measure an object's orientation, position, vibration and movement in 3D space in real-time. These sensors are typically arranged in a pattern, including a tri-axial accelerometer, gyroscope, and magnetometer[**Ahmad:2013**].
An accelerometer is used to measure the change in velocity of a moving or vibrating object[**Ahmad:2013**]. Gyroscope measures the angular rotation[**Ahmad:2013**]. A Magnetometer is used to measure yaw angle rotation, calibrating to the gyroscope data to improve the big drift issue[**Ahmad:2013**].
The accelerometer functions by gauging the acceleration (Ax, Ay, Az), which denotes the rate of acceleration change over time. The acceleration values along each axis are conventionally expressed in units of meters per second squared (m/s^2) [**Vernier:2023**].
To illustrate, if the LSM9DS1 records an acceleration of 15 m/s^2 along the x-axis, this signifies that the object under observation is experiencing a velocity increment of 15 meters per second every second in the x-direction.

4.1.2. Specific Sensors

Tilt Detection Using the accelerometer, the IMU can detect orientation changes with minimal power usage. Tilt detection is particularly useful for identifying subtle shifts in position, enhancing the responsiveness of gesture-based controls.[**Zhou:2020**]
Significant Motion Detection The accelerometer enables significant motion detection (SMD), which recognizes large-scale movements. SMD can be used to activate specific device functions, such as waking the system from sleep mode or triggering predefined actions based on significant gestures.[**Zhou:2020**]
The LSM9DS1 is a versatile and energy-efficient IMU with advanced sensing capabilities, supporting a range of applications where motion detection, orientation tracking, and gesture recognition are essential. Its compact design, low power consumption, and robust environmental tolerance make it a reliable choice for portable, battery-powered devices like the Magic Wand.[**Zhou:2020**]
An IMU consists of three sensors that measure an object's orientation, position, vibration and movement in 3D space in real-time. These sensors are typically arranged in a pattern, including a tri-axial accelerometer, gyroscope, and magnetometer[**Ahmad:2013**].
An accelerometer is used to measure the change in velocity of a moving or vibrating object[**Ahmad:2013**]. Gyroscope measures the angular rotation[**Ahmad:2013**]. A Magnetometer is used to measure yaw angle rotation, calibrating to the gyroscope data to improve the big drift issue[**Ahmad:2013**].

The accelerometer functions by gauging the acceleration (Ax, Ay, Az), which denotes the rate of acceleration change over time. The acceleration values along each axis are conventionally expressed in units of meters per second squared (m/s^2) [**Vernier:2023**]. To illustrate, if the LSM9DS1 records an acceleration of 15 m/s^2 along the x-axis, this signifies that the object under observation is experiencing a velocity increment of 15 meters per second every second in the x-direction.

4.1.3. Specification

4.1.4. Library

Details about libraries used to interface with accelerometers (e.g., Adafruit Sensor Library, Arduino libraries). Installation process and setup. Features provided by these libraries (e.g., data acquisition, filtering, scaling).

```
// Include the necessary libraries
#include <Wire.h>
#include <Adafruit_Sensor.h>
#include <Adafruit_LSM9DS1.h>
```

4.1.5. Functions

```
// Check if acceleration data is available
if (IMU.accelerationAvailable()) {
 // Read the acceleration values
 IMU.readAcceleration(x, y, z);

 // Print the acceleration values
 Serial.print("AccX: ");
 Serial.print(x);
 Serial.print(", AccY: ");
 Serial.print(y);
 Serial.print(", AccZ: ");
 Serial.println(z);
}
```

4.1.6. Calibration

Steps for calibrating an accelerometer:

- Importance of calibration for accelerometer sensors.
- Methods for calibrating accelerometers (e.g., zeroing, scaling, temperature compensation).
- Tools and techniques for accelerometer calibration (e.g., software tools, calibration kits).

4.1.7. Simple Code

```
 #include <Wire.h>
 #include <Adafruit_Sensor.h>
 #include <Adafruit_LSM9DS1.h>

 // Create LSM9DS1 object
 Adafruit_LSM9DS1 lsm = Adafruit_LSM9DS1();
```

```
void setup() {
// Start the Serial Monitor
Serial.begin(115200);

// Initialize the LSM9DS1
if (!lsm.begin()) {
 Serial.println("Failed to initialize LSM9DS1! Check wiring.");
 while (1);
}

// Set accelerometer range (default is ±2g)
lsm.setupAccel(lsm.LSM9DS1_ACCELRANGE_2G);
// Options: 2G, 4G, 8G, 16G

// Optional: Set accelerometer data rate (default is 119 Hz)
lsm.setupAccelDataRate(lsm.LSM9DS1_ACCELDATARATE_119HZ);
}

void loop() {
// Read accelerometer data
sensors_event_t accelEvent;
lsm.getEvent(&accelEvent, NULL, NULL);
// Get accelerometer event data only

// Print accelerometer readings (in m/s²)
Serial.print("Accel X: ");
Serial.print(accelEvent.acceleration.x);
Serial.print(" m/s², Y: ");
Serial.print(accelEvent.acceleration.y);
Serial.print(" m/s², Z: ");
Serial.print(accelEvent.acceleration.z);
Serial.println(" m/s²");

// Delay to make output more readable
delay(100);
}
```

4.1.8. Applications

Practical uses of accelerometers:

- In smartphones (e.g., screen orientation, motion detection).
- Automotive industry (e.g., airbag deployment, stability control).
- Robotics (e.g., navigation, balance).
- Wearable devices and fitness trackers.
- Industrial applications (e.g., vibration monitoring, machine diagnostics).

4.1.9. Tests

Techniques to test an accelerometer's functionality: This is the C++ code for testing the accelerometer on the Arduino Nano 33 BLE Sense using the `Adafruit_LSM9DS1`

library.

Listing 4.1: Testing the accelerometer

```
#include <Wire.h>
#include <Adafruit_Sensor.h>
#include <Adafruit_LSM9DS1.h>

// Create an instance of the LSM9DS1 sensor
Adafruit_LSM9DS1 lsm = Adafruit_LSM9DS1();

void setup() {
    // Start serial communication
    Serial.begin(115200);

    // Initialize the LSM9DS1 sensor
    if (!lsm.begin()) {
        Serial.println("Could not find a valid LSM9DS1 sensor, check wiring!"
            ↪ );
        while (1);
    }

    Serial.println("LSM9DS1 test initialized.");
}

void loop() {
    // Read accelerometer data
    sensors_event_t event;
    lsm.getEvent(&event);

    // Print accelerometer values
    Serial.print("X: ");
    Serial.print(event.acceleration.x);
    Serial.print(" Y: ");
    Serial.print(event.acceleration.y);
    Serial.print(" Z: ");
    Serial.println(event.acceleration.z);

    // Delay before the next reading
    delay(1000);
}
```

Figure 4.1.: C++ code for testing the accelerometer on the Arduino Nano 33 BLE Sense.

This is the Python code for reading the accelerometer data from the Arduino via serial communication. The `pyserial` library is required for this script.

4.1.10. Further Readings

4.2. Gyroscope

4.2.1. Description

The STMicroelectronics LSM9DS1 gyroscope is a precision instrument for measuring angular velocity around the x, y, and z axes. With adaptable measurement ranges of ±245, ±500, and ±2000 degrees per second (dps), the gyroscope is highly versatile for various applications such as inertial navigation, robotics, and drone stabilization.[**St:2024**] Key Features:

- Sampling Rates: The gyroscope offers adjustable sampling rates, allowing it to operate at frequencies of 14.9 Hz, 59.5 Hz, 119 Hz, 238 Hz, 476 Hz, or 952 Hz.[**St:2024**] This flexibility ensures compatibility with a broad range of motion measurement requirements.

Listing 4.2: Accelerometer testing in pyserial

```
import serial
import time

# Replace with the correct port for your system (e.g., 'COM3' on Windows or
    ↪ '/dev/ttyACM0' on Linux)
arduino_port = '/dev/ttyACM0'
baud_rate = 115200

# Establish connection to the Arduino
arduino = serial.Serial(arduino_port, baud_rate, timeout=1)
time.sleep(2)  # Wait for Arduino to initialize

# Function to read accelerometer data from the Arduino
def read_accelerometer():
while True:
line = arduino.readline().decode('utf-8').strip()
if line:
print(line)

# Start reading accelerometer data
read_accelerometer()
```

Figure 4.2.: Python code for reading accelerometer data from the Arduino Nano 33 BLE Sense.

- Voltage and Power Consumption: The gyroscope operates within a voltage range of 1.71 V to 3.6 V, making it suitable for different system configurations.[**St:2024**] Its low power consumption—between 1 mA to 2 mA—enhances its suitability for portable, battery-operated devices.[**St:2024**]

- Compact Design: The LSM9DS1 integrates seamlessly into compact systems, offering engineers and designers a robust yet space-efficient solution.[**Maker:2024**]

- Operation Principle: The gyroscope operates based on Coriolis acceleration, a phenomenon observed when a vibrating object moves in a rotating reference frame. Piezoelectric crystals are used to detect changes in angular velocity by converting inertial forces into electrical signals.

- Applications: The LSM9DS1 gyroscope is designed for:

Inertial Measurement Units (IMUs): Combined with accelerometers and magnetometers, it forms a 9-DOF motion tracking system.[**Ahmad:2013**] Drone Stabilization: Essential for real-time attitude adjustments.[**Ahmad:2013**] Navigation Systems: Key in applications like gyrocompasses or attitude heading reference systems.[**Ahmad:2013**]

$$\text{Angular velocity} = \left(\frac{\text{Gyroscope axis raw data}}{65536} \times \text{full scale Gyroscope range} \right) \frac{^\circ}{\text{s}}$$

For example, if the gyroscope's raw data along the X axis is 16384 and the range is $\pm 250°/\text{s}$, the calculation for angular velocity along the X axis would be:

$$\text{Angular velocity along the X axis} = \left(\frac{16384}{65536} \times 500 \right) \frac{^\circ}{\text{s}} = 125 \frac{^\circ}{\text{s}}$$

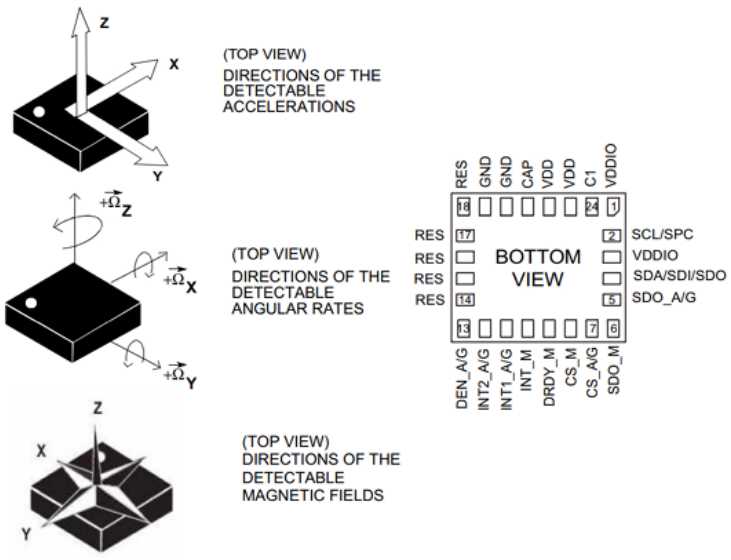


Figure 4.3.: **Gyroscope**

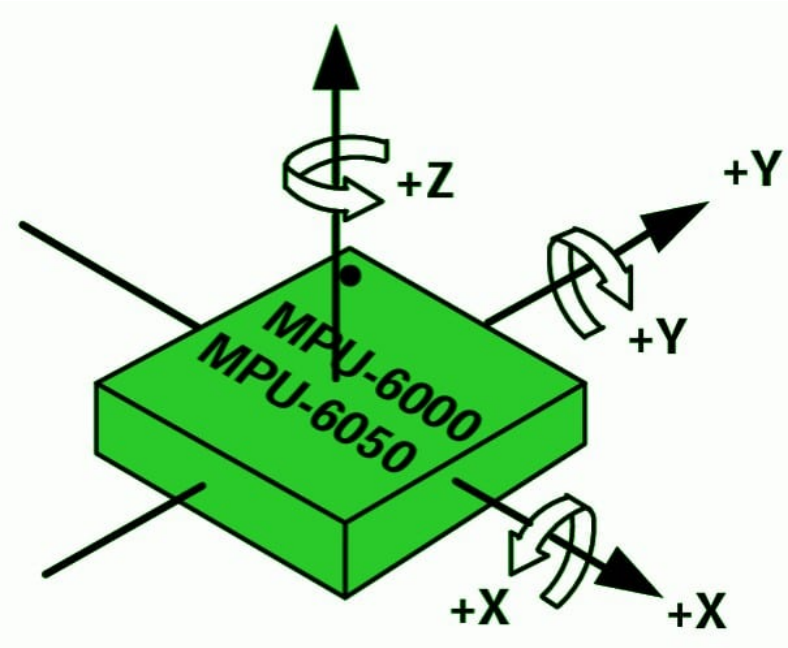


Figure 4.4.: **Orientation Axes of Gyroscope**

4.2.2. Specification

Gyroscope Specifications of MPU6050:

Full scale range	FS_SEL	Range
	0	±250°/s
	1	±500°/s
	2	±1000°/s
	3	±2000°/s
Sensitivity Scale Factor Tolerance		±3%
Gyroscope start-up time		30 ms
Output data rate		4 to 8000 Hz

4.2.3. Library

Description To meet the project's needs, three essential libraries must be integrated, each serving a specific purpose that ensures smooth operation and functionality of the system. Here's a breakdown of each library's role:

1. Wire.h: This is an Arduino standard library used for I2C communication, a protocol that allows devices to communicate with each other using only two wires, reducing the complexity of wiring in systems that need to connect multiple devices. I2C is widely used for connecting sensors, displays, and other peripherals to a microcontroller. The 'Wire.h' library simplifies interactions with I2C devices by handling the low-level details of communication. Developers can initialize the I2C bus, send and receive data, and manage multiple I2C devices on the same bus seamlessly. This is crucial for systems that need to manage several sensors or external modules, as it facilitates smooth and efficient data transfer.[**Passaro:2017**]

2. Kalman.h: The Kalman filter is an advanced mathematical algorithm used to process noisy sensor data and provide more accurate estimates of a system's state. The 'Kalman.h' library allows developers to easily implement this filter into Arduino-based projects. It is particularly useful in applications where precise data is essential, such as robotics, control systems, and navigation. By filtering out noise and accounting for uncertainties in sensor readings, the Kalman filter improves the accuracy of measurements like position, velocity, and orientation, which are crucial in motion tracking, sensor fusion, or stabilizing systems like drones and robots.[**Passaro:2017**]

3. Arduino_LSM9DS1.h: This library is specifically designed to interface with the LSM9DS1 IMU (Inertial Measurement Unit) sensor, developed by STMicroelectronics. The LSM9DS1 combines three essential motion sensors in one package: an accelerometer, a gyroscope, and a magnetometer. The 'Arduino_LSM9DS1.h' library allows developers to easily access data from these sensors, such as acceleration, angular velocity, and magnetic field strength. Additionally, the library offers functions to configure sensor parameters like measurement ranges, sampling rates, and operating modes, enabling customization based on the needs of the application. This is particularly useful in systems that rely on accurate motion tracking or orientation sensing.[**Passaro:2017**]

Together, these libraries enable the integration of multiple sensors into a cohesive system, facilitating communication, data processing, and precise motion measurement.

4.2.4. Installation

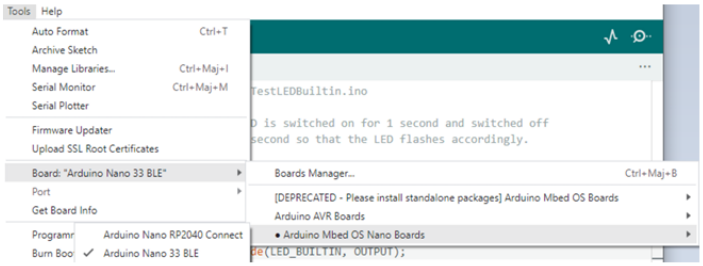


Figure 23.5.: Board card installation

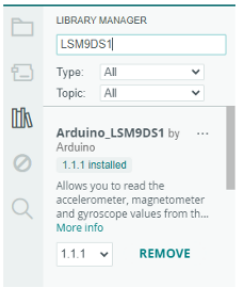


Figure 4.5.: **Installation of Libraries**

```
1  #include <Wire.h>
2
```

Figure 4.6.: **Installation of wire.h**

4.2.5. Functions

The LSM9DS1 library for the Arduino Nano 33 BLE Sense provides a set of functions to interact with the LSM9DS1 IMU (Inertial Measurement Unit) sensor. The functions available depend on the communication protocol being used, either I2C or SPI.[**Passaro:2017**]

Function IMU.readGyroscope() The function IMU.readGyroscope() is used to retrieve data from an Inertial Measurement Unit's (IMU) gyroscope. It returns the angular velocity in degrees per second (dps), providing information about the rotational movement detected by the IMU. This data is crucial for applications like robotics, motion tracking, and navigation systems, where precise orientation control and stabilization are required. The function typically returns x, y, and z values, representing the angular velocity along each of the three axes (x, y, and z). These values are floating-point numbers that describe the rotational velocity along each axis, enabling detailed tracking of changes in orientation. By regularly calling this function and analyzing the data over time, it becomes possible to monitor and respond to rotational movements accurately. For example, in robotics, these readings can be used to adjust a robot's position or orientation in real-time, while in drones, they assist with stabilization during flight.

Function IMU.gyroscopeAvailable() The function IMU.gyroscopeAvailable() is used to check whether new gyroscope data is available from the IMU. It returns a value of 1 if new data is ready to be retrieved, and 0 if no new data is available. This function is particularly useful in real-time applications, such as drone stabilization, virtual

```
1000  // Declare variables to store gyroscope data
      float x, y, z;
1002
      // Check if gyroscope data is available
1004  if (IMU.gyroscopeAvailable()) {
        // Read the gyroscope data into variables x, y, z
1006    IMU.readGyroscope(x, y, z);

1008    // Print the x-axis angular velocity
        Serial.print(x);
1010
        // Print a tab space for separation
1012    Serial.print('\t');

1014    // Print the y-axis angular velocity
        Serial.print(y);
1016
        // Print another tab space for separation
1018    Serial.print('\t');

1020    // Print the z-axis angular velocity and move to the next line
        Serial.println(z);
1022  }
```

Listing 4.1.: Sample Code of IMU.readGyroscope()

reality systems, or inertial navigation, where the system must determine whether to wait for updated gyroscope data or proceed with processing the available information.

For example, in a drone application, when the function returns 1, the system knows that new gyroscope data is available, and it can proceed to retrieve and use this information to adjust the drone's orientation or stabilize its motion. Conversely, a return value of 0 means that no new data has been received, prompting the system to wait until new data is available before continuing the process.

In the provided code example:
The program checks if new gyroscope data is available. If it is, the data is read and output to the serial monitor, showing the angular velocity along the x, y, and z axes. This function is integral for applications that require continuous monitoring of rotational movement, ensuring that the system can react promptly to changes in orientation.

Function IMU.gyroscopeSampleRate() The function IMU.gyroscopeSampleRate() is used to retrieve the rate at which the gyroscope integrated into the Inertial Measurement Unit (IMU) collects samples, typically expressed in Hertz (Hz). This sample rate indicates how frequently the gyroscope measures angular velocity, providing insights into its operational efficiency and performance.

Sample Rate: It is the frequency at which the gyroscope takes measurements. For example, a sample rate of 100 Hz means the gyroscope takes 100 readings per second.
Function Breakdown: IMU.gyroscopeSampleRate(): This function returns the sample rate of the gyroscope in Hertz (Hz). The Serial.print() functions then display the sample rate and additional information about the angular speed readings in degrees per second for the X, Y, and Z axes.

This function is especially useful in applications that require accurate and efficient motion tracking, such as drone stabilization, robotics, or virtual reality systems. Knowing the sample rate helps determine how often new data is available, which influences the system's ability to respond to changes in rotational movement.

```
1000  // Declare variables to store gyroscope data
      float x, y, z;
1002
      // Check if gyroscope data is available
1004  if (IMU.gyroscopeAvailable()) {
        // Read the gyroscope data into variables x, y, z
1006    IMU.readGyroscope(x, y, z);

1008    // Print the x-axis angular velocity
        Serial.print(x);
1010
        // Print a tab space for separation
1012    Serial.print('\t');

1014    // Print the y-axis angular velocity
        Serial.print(y);
1016
        // Print another tab space for separation
1018    Serial.print('\t');

1020    // Print the z-axis angular velocity and move to the next line
        Serial.println(z);
1022  }

1024
```

Listing 4.2.: Sample Code of IMU.gyroscopeAvailable()

```
1000  // Print the gyroscope sample rate
      Serial.print("Gyroscope sample rate= ");
1002  Serial.print(IMU.gyroscopeSampleRate());  // Print the gyroscope
        sample rate
      Serial.println("Hz");
1004  Serial.println();  // Print a blank line for separation

1006  // Print a label for angular speed in degrees/second
      Serial.println("Angular speed in degrees/second");
1008
      // Print the axis labels with tab separation
1010  Serial.println("X\tY\tZ");

1012
```

Listing 4.3.: Sample Code of IMU.gyroscopeSampleRate()

4.2.6. Calibration

There are various methods to calibrate the sensors involved. It is necessary to know whether the sensor is balanced since the time between each calibration needs to be explicitly defined. Regular calibration should be done, especially when strange outputs are noticed.
Some methods of calibration are briefed below:

- **Low and high limit method**
  The low and high limit method involves recording minimum and maximum values on all three axes using a simple scratch to determine their absolute values. The sensor undergoes circular rotations along each axis multiple times. The centre point is then identified between these extremes.[**Gyroplace:2023**]

Increasing the number of rotations enhances the likelihood of capturing the absolute peak. The center point will be close to zero if the sensor exhibits no offset. However, slight variations may indicate a hard iron offset attributed to distortion caused by the Earth's magnetic field.[**Gyroplace:2023**] This method assumes minimal soft iron distortion, evident from the rounded outlines in the graph.[**Gyroplace:2023**] It is important to note that this method necessitates capturing values each time to prevent performance degradation due to component drift and aging sensors. For devices relying on primary batteries, calibration becomes essential after each battery change, as the battery inevitably serves as the main source of magnetic disturbance, and new batteries may behave differently from their predecessors.[**Gyroplace:2023**]

- **Scale Factor and Non-Orthogonal Calibration**
  With given initial attitude derived from alignment procedure, the gyroscope measurement can be integrated to calculate the orientation information through Strapdown inertial navigation algorithm. However, the computed attitude will drift over time and the error is gradually accumulated because of the sensor error. In stationary or low dynamic condition, the accelerometer output can be used to estimate the orientation relative to horizontal plane (i.e., pitch and roll).[**Gyroplace:2023**] The attitude derived from accelerometer output is independent in different time epochs and not affected by accumulated error. Hence, based on the different sensors' complimentary error propagation characteristics, we can make use of the accelerometer-derived attitude as reference signal to evaluate the attitude error introduced during integration process, and consequently determine the gyroscope error.[**Gyroplace:2023**]

During the calibration process, the IMU is handheld by user and rotated along its axes slowly to avoid introducing external acceleration. The IMU orientation keeps varying during this procedure and the attitudes derived from different inertial sensors are compared to amend the attitude error and determine the sensor errors. A Kalman filter is designed to estimate the scale factor and non-orthogonal errors of gyroscope. The attitude error propagation equation, which includes sensor error, is utilized as the system dynamic model. The relationship between the accelerometer output and attitude error is modeled as the measurement equation.[**Gyroplace:2023**]

4.2.7. Simple Code

Example IMU: Accelerometer and Gyroscope

4.2.8. Simple Application

Gyroscopes in IMUs are essential for detecting and maintaining orientation and rotational motion in a wide range of devices:

- **Smartphones/Tablets**: They enable features like screen auto-rotation and motion controls for gaming and virtual reality (VR).[**Passaro:2017**]
- **Drones**: Gyroscopes stabilize flight by measuring angular velocity, allowing the drone to correct tilt and maintain stable orientation.[**Passaro:2017**]
- **Robotics**: In robots, gyroscopes assist in balancing and navigation by tracking rotational movements, ensuring accurate motion control.[**Passaro:2017**]
- **VR/AR**: In virtual and augmented reality systems, gyroscopes track head movement to provide an immersive experience by adjusting visuals based on real-time orientation.[**Passaro:2017**]

```
#include <Arduino_LSM9DS1.h>

void setup() {
Serial.begin(9600);

// Initialize the IMU
if (!IMU.begin()) {
  Serial.println("Failed to initialize IMU!");
  while (1); // Stop execution if initialization fails
}
}
void loop() {
float x, y, z;

// Check if new acceleration data is available
if (IMU.accelerationAvailable()) {
  IMU.readAcceleration(x, y, z);
  Serial.print("AccX: ");
  Serial.print(x);
  Serial.print(", AccY: ");
  Serial.print(y);
  Serial.print(", AccZ: ");
  Serial.println(z);
}
// Check if new gyroscope data is available
if (IMU.gyroscopeAvailable()) {
  IMU.readGyroscope(x, y, z);
  Serial.print("GyroX: ");
  Serial.print(x);
  Serial.print(", GyroY: ");
  Serial.print(y);
  Serial.print(", GyroZ: ");
  Serial.println(z);
}
delay(1000); // Wait for 1 second before reading again
}
```

Listing 4.4.: Example code for reading accelerometer and gyroscope data from the LSM9DS1 IMU

4.2.9. Testing

The following code reads the gyroscope data (X, Y, and Z axes) from the LSM9DS1 sensor on the Arduino Nano 33 BLE Sense and sends it to the Serial Monitor.

Listing 4.3: Arduino Code to Test the Gyroscope

```
#include <Wire.h>
#include <Adafruit_Sensor.h>
#include <Adafruit_LSM9DS1.h>

// Create an instance of the LSM9DS1 sensor
Adafruit_LSM9DS1 lsm = Adafruit_LSM9DS1();

void setup() {
    // Start serial communication
    Serial.begin(115200);

    // Initialize the LSM9DS1 sensor
    if (!lsm.begin()) {
        Serial.println("Could not find a valid LSM9DS1 sensor, check wiring!");
        while (1);
    }

    Serial.println("LSM9DS1 Gyroscope test initialized.");
}

void loop() {
    // Read gyroscope data
    sensors_event_t event;
    lsm.getEvent(&event);

    // Print gyroscope values
    Serial.print("Gyroscope X: ");
    Serial.print(event.gyro.x);
    Serial.print(" Y: ");
    Serial.print(event.gyro.y);
    Serial.print(" Z: ");
    Serial.println(event.gyro.z);

    // Delay before the next reading
    delay(1000);
}
```

4.2.10. Python Code

The following Python script reads the gyroscope data sent by the Arduino and prints the values. Ensure you have the `pyserial` library installed.
Install the `pyserial` library:

Listing 4.4: Installing pyserial

```
pip install pyserial
```

Python Code:

Listing 4.5: Python Code to Read and Test Gyroscope Data

```
import serial
import time

# Set up the serial connection (replace with your actual port)
ser = serial.Serial('COM3', 115200)  # For Windows, replace COM3 with your port,
    ↪ for Mac/Linux, it may be /dev/ttyACM0 or /dev/ttyUSB0

# Allow some time for the Arduino to reset and start transmitting data
time.sleep(2)

# Loop to continuously read gyroscope data
while True:
try:
# Read a line of data from Arduino
```

```
    line = ser.readline().decode('utf-8').strip()

    # Only print the line if it contains the "Gyroscope" data
    if 'Gyroscope' in line:
    print(line)

    # Delay before the next reading
    time.sleep(1)

    except KeyboardInterrupt:
    print("Exiting...")
    break

    # Close the serial connection when done
    ser.close()
```

4.2.11. Further Readings

Schanda, Janos: Colorimetry: Understanding the CIE System.Wiley, 2007.[[**Schanda:2007**]] Lukac, Rastislav and Plataniotis, Konstantinos N.: Color Image Processing: Methods and Applications.CTC Press,2018.[[**Lukac:2018**]]

4.3. Magnetometer

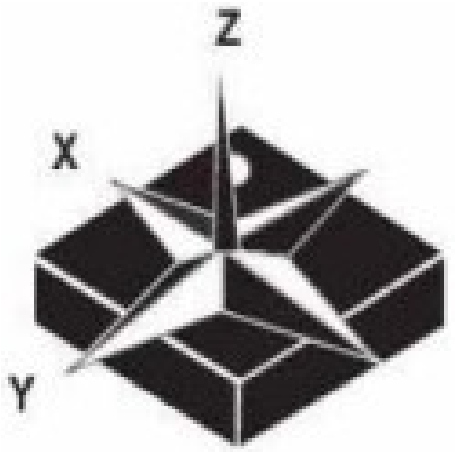


Figure 4.7.: **Magnetometer Directions** [**Stm:2015**]

4.3.1. Description

In the context of a magnetometer, the x, y, and z axes (Mx, My, Mz) typically delineate the three-dimensional space in which the magnetic field is being assessed. The x-axis typically corresponds to the horizontal component of the magnetic field, the y-axis signifies the vertical component, and the z-axis reflects the magnetic field strength [**Kostiainen:2023**]. This three-dimensional measurement provides a comprehensive understanding of the magnetic field in the surrounding space. A magnetometer is a sensor that measures the strength and direction of a magnetic field. It is commonly used to detect the Earth's magnetic field for navigation purposes, detect magnetic anomalies, or determine heading in devices like smartphones, drones, and robotics. Magnetometers are fundamental in compasses, GPS systems, and geological exploration (Ripka, 2001).

4.3.2. Specific Sensors

There are various types of magnetometers available, differing in precision, range, and applications. Here are some popular ones:

- HMC5883L: A widely used 3-axis digital magnetometer that provides low-cost magnetic field measurement [**Honeywell:2012**].
- LSM303DLHC: Combines an accelerometer and magnetometer, enabling tilt-compensated compass applications [**Adafruit:2021a**].
- AK8963: A 3-axis magnetometer typically used with IMUs like the MPU9250 for high accuracy [**AsahiKasei:2014**].
- MAG3110: A small, low-power 3-axis magnetometer for embedded systems [**NXP:2013**].

4.3.3. Specification

General magnetometer specifications include:

- 3D digital magnetic sensor: Detects magnetic fields with a full scale of $\pm 4/\pm 8/\pm 12/\pm 16$ gauss.
- Axes: Single-axis or 3-axis (most modern sensors are 3-axis).
- Sensitivity: Ability to detect weak magnetic fields (e.g., microteslas, nanoteslas).
- Range: Typically between ± 8 to ± 100 microteslas.
- Resolution: The smallest change in the magnetic field that the sensor can detect.
- Power Consumption: Important for portable systems.
- Interface: I2C, SPI, or analog output.

4.3.4. Library

Libraries simplify interfacing with magnetometer sensors, converting raw data into readable formats. Below are some libraries:
Arduino:

- HMC5883L: Adafruit_HMC5883_Unified.h (Adafruit, 2021b).
- LSM303: Adafruit_LSM303DLHC.h (Adafruit, 2021a).

Python:

- Use smbus or CircuitPython libraries for I2C sensors [**Adafruit:2020**].
- Example for Raspberry Pi: hmc5883l Python library [**Hollingworth:2019**].

To install the library for Arduino:

```
// Include the necessary libraries
#include <Adafruit_Sensor.h>
#include <Adafruit_HMC5883_U.h>
```

4.3.5. Calibration

Basic calibration steps:

- Rotate the sensor in all directions.
- Plot the data to ensure it forms a circle.
- Apply corrections for centering the data.

4.3.6. Simple Code

```
// Include the necessary libraries
#include <Wire.h>
#include <Adafruit_Sensor.h>
#include <Adafruit_HMC5883_U.h>
// Create an HMC5883 magnetometer object with a unique ID
Adafruit_HMC5883_Unified mag = Adafruit_HMC5883_Unified(12345);
// Setup function
void setup(void) {
  // Start serial communication at 9600 baud rate
  Serial.begin(9600);
  // Initialize the magnetometer
  if (!mag.begin()) {
    Serial.println("No HMC5883L detected ... Check your wiring!");
    while (1);  // Halt the program if the sensor is not detected
  }

  // Set the magnetometer gain
  mag.setMagGain(HMC5883_MAGGAIN_1_3);
}
// Loop function
void loop(void) {
  // Declare a variable to store magnetometer data
  sensors_event_t event;
  // Get the magnetometer event data
  mag.getEvent(&event);
  // Calculate the heading angle in radians
  float heading = atan2(event.magnetic.y, event.magnetic.x);
  if (heading < 0) heading += 2 * PI;
  // Convert the heading from radians to degrees
  float headingDegrees = heading * 180 / M_PI;
  // Print the heading in degrees
  Serial.println(headingDegrees);
  // Delay for 500ms before the next reading
  delay(500);
}

```

Listing 4.5.: Example code for reading magnetometer data

4.3.7. Applications

Magnetometers have diverse applications, including:

- Navigation: Electronic compasses in GPS, drones, and aircraft [**Sherwood:2013**].
- Robotics: Precise heading information for autonomous systems [**IEEE:2020**].
- Consumer Devices: Smartphones, wearables for direction detection [**AsahiKasei:2014**].

- Geological Exploration: Detect magnetic anomalies for mineral exploration [**Hansen:2017**].
- Space Exploration: Magnetic field mapping on planets.
- Security: Detection of ferromagnetic objects in metal detectors [**Williams:2015**].

4.3.8. Tests

The following C++ code reads and prints the magnetometer data (X, Y, Z values) from the LSM9DS1 sensor connected to the Arduino Nano 33 BLE Sense.

Listing 4.6: C++ Code for Arduino to Read Magnetometer Data

```
#include <Wire.h>
#include <Adafruit_Sensor.h>
#include <Adafruit_LSM9DS1.h>

// Create an instance of the LSM9DS1 sensor
Adafruit_LSM9DS1 lsm = Adafruit_LSM9DS1();

void setup() {
    // Start serial communication
    Serial.begin(115200);

    // Initialize the LSM9DS1 sensor
    if (!lsm.begin()) {
        Serial.println("Could not find a valid LSM9DS1 sensor, check wiring!");
        while (1);
    }

    Serial.println("LSM9DS1 test initialized.");
}

void loop() {
    // Read magnetometer data
    sensors_event_t event;
    lsm.getEvent(&event, Adafruit_LSM9DS1::MAGNETOMETER);

    // Print magnetometer values
    Serial.print("Mag X: ");
    Serial.print(event.magnetic.x);
    Serial.print(" Y: ");
    Serial.print(event.magnetic.y);
    Serial.print(" Z: ");
    Serial.println(event.magnetic.z);

    // Delay before the next reading
    delay(1000);
}
```

This Python code reads the magnetometer data from the Arduino via serial communication.

Listing 4.7: Python Code to Read Magnetometer Data via Serial

```
import serial
import time

# Replace with the correct port for your system (e.g., 'COM3' on Windows or '/dev
    ↪ /ttyACM0' on Linux)
arduino_port = '/dev/ttyACM0'
baud_rate = 115200

# Establish connection to the Arduino
arduino = serial.Serial(arduino_port, baud_rate, timeout=1)
time.sleep(2)  # Wait for Arduino to initialize

# Function to read magnetometer data from the Arduino
def read_magnetometer():
while True:
line = arduino.readline().decode('utf-8').strip()
if line:
print(line)

# Start reading magnetometer data
read_magnetometer()
```

4.3.9. Further Readings

For more information, refer to:
Datasheets: Sensor-specific datasheets (e.g., HMC5883L, LSM303). Books:

- Ripka, A. (2001) Introduction to Magnetometers. New York: Wiley.
- Sherwood, T. (2013) Magnetic Sensors in Navigation Systems. Berlin: Springer.

Online Resources:

- Adafruit tutorials on magnetometers [**Adafruit:2021a**].
- Research papers on magnetic anomaly detection [**Hansen:2017**].

Part III.

Development

4.4. Data Base

4.5. Data Characteristics

1. **Structure:**

   The JSON structure is organized into *strokes*, each containing an *index* and an array of *stroke points* with X-Y coordinates. This structured format is conducive to representing sequential information, allowing efficient processing and analysis of stroke data.

   Listing 4.8: Example of gesture data with sensor readings

```
{
    "gesture_data": [
    {
        "label": "W",
        "sensor_readings": [
        {"acceleration_x": 0.23, "acceleration_y": -0.15, "acceleration_z":
            ↪  0.98, ...},
        {"acceleration_x": 0.21, "acceleration_y": -0.18, "acceleration_z":
            ↪  0.95, ...},
        ...
        ]
    },
    {
        "label": "O",
        "sensor_readings": [
        {"acceleration_x": 0.14, "acceleration_y": 0.22, "acceleration_z":
            ↪ 0.93, ...},
        {"acceleration_x": 0.12, "acceleration_y": 0.20, "acceleration_z":
            ↪ 0.91, ...},
        ...
        ]
    },
    {
        "label": "L",
        "sensor_readings": [
        {"acceleration_x": -0.10, "acceleration_y": -0.25, "acceleration_z"
            ↪ : 0.88, ...},
        {"acceleration_x": -0.12, "acceleration_y": -0.28, "acceleration_z"
            ↪ : 0.85, ...},
        ...
        ]
    },
    ...
    ]
}
```

2. **Size:**

   The dataset comprises a total of 200 labeled instances, providing a robust foundation for training and evaluation. Each gesture type (W, O, L) is well-represented, with over 70 instances for each, ensuring balanced class distribution and supporting reliable model performance.

3. **Format:**

   The dataset is stored in the JSON format, a flexible and widely adopted standard for data representation. Each JSON entry contains a hierarchical structure, including stroke indices, an array of stroke points, and corresponding X-Y coordinates for each point. This structured organization facilitates efficient parsing and analysis, making it ideal for sequential data representation and machine learning applications.

4. **Anomalies:**

   Efforts were made to ensure clear and deliberate wand movements during gesture performances to minimize the occurrence of anomalies. A manual review process

was applied during the labeling phase to detect and correct any mislabeled or ambiguous instances, thereby enhancing the overall quality and reliability of the dataset.

5. **Measurement and Screen Size:**

   - **Accelerometer Measurements:**

     Motion data was captured in three dimensions (X, Y, Z) using the accelerometer on the Arduino Nano 33 BLE Sense board. Each gesture was performed within a duration of 1 to 2 seconds, ensuring consistency in the data collection process. The accelerometer's sensitivity enabled precise tracking of gesture dynamics.

   - **Screen Size:**

     A laptop screen with dimensions of 25x30 cm was used for real-time visualization and labeling of recorded gestures. The user interface provided a robust platform for reviewing, labeling, and correcting gesture data, ensuring accurate annotations and facilitating seamless data management.

6. **Origin:**

   The dataset for the Magic Wand project originates from the integration of real-world gestures with advanced technology, enabled by the Magic Wand website. Our team utilized the Magic Wand Capture sketch, which was uploaded onto the Arduino Nano 33 BLE Sense board, to collect motion data. The user-friendly interface of the website facilitated seamless recording, reviewing, and labeling of unique gestures. This collaborative effort ensured the creation of a diverse and representative dataset, encompassing authentic real-world scenarios and variations among users. The Magic Wand prototype, combined with the Arduino Nano 33 BLE Sense board, served as a crucial tool for capturing nuanced gesture variations. The structured methodology provided by the website significantly contributed to refining the dataset, aligning with the project's objectives and enhancing its utility for practical machine learning applications.

The key benefit of expanding a dataset with more input data is that it allows the machine learning model to improve in terms of accuracy, efficiency, and performance. By including additional data, the model can better learn the patterns and characteristics of the gestures. This approach is applied to the remaining gestures, such as ring, slope, and unknown, following the same process [**Wings:2023**]. With the data collected for these gestures, we can split the dataset into two parts: one for training the model and another for testing its performance.
A significant challenge in preparing the data is the presence of outliers or anomalies, which are values that deviate significantly from the expected range. These outliers can negatively impact the model's performance. However, one way to address this challenge is by increasing the volume of data to provide a more robust and diverse training set, making the model more resilient to such anomalies. Below are some common types of outliers that may be encountered during data preparation [**Munoz:2019**]:

- **Noise in Accelerometer Readings:**

  Environmental interference or external factors during data collection may introduce noise into the accelerometer readings. Outliers can appear as sudden spikes or drops in sensor values that do not correspond to genuine gestures. These are typically considered noise and can be minimized by increasing data points and ensuring proper sensor calibration.

- **Abnormal Gesture Patterns:**

Outliers may arise when users perform gestures that deviate from the expected behavior or in an unconventional manner. These abnormal gestures can negatively affect the training process and may distort the model's ability to correctly classify gestures.

- **Sensor Malfunctions:**

  Outliers can also be a result of sensor malfunctions or inaccuracies in the Arduino Nano 33 BLE Sense device. Issues like sudden jumps, constant offsets, or erratic sensor behavior are typically indicative of hardware problems and should be addressed by either recalibrating the sensors or discarding faulty readings.

- **Inconsistent Data Recording:**

  Variations in the way users record gestures—such as differences in speed, amplitude, or gesture duration—can lead to inconsistencies in the dataset. It is important to standardize the recording process and eliminate these variations to ensure that the model receives consistent and reliable data for training.

- **User-Specific Outliers:**

  Each individual may exhibit unique movement patterns or unintentional variations in how they perform gestures. These user-specific outliers can affect the model's ability to generalize across different users. Identifying and addressing these variations, such as normalizing gesture data for different users, is essential for creating a model that works well across a diverse group of individuals.

- **Data Transmission Errors:**

  Errors or interruptions during the data transmission from the Arduino Nano to the recording system can result in outliers, such as missing or corrupted data points. These transmission errors can be detected and rectified by performing error-checking and data validation techniques during preprocessing to ensure that the dataset is complete and reliable.

In practice, outliers can be handled in several ways. They can either be removed from the dataset if they are deemed irrelevant or incorrect, or they can be corrected if a reasonable method of rectification is available. Handling outliers effectively ensures that the model can learn from clean, reliable data, improving its accuracy and performance. Additionally, identifying and addressing outliers early in the data preparation process helps create a more robust and generalizable model.

4.6. Data Transformation and Data Mining

The data mining step is the phase where the model is developed and trained to recognize patterns and make predictions. However, before beginning this process, it is crucial to select a suitable algorithm that aligns with the project's objectives and the dataset's characteristics. Factors such as the type of data, computational constraints, and desired output play a key role in this decision. As detailed in the previous section, the dataset is split into training and testing subsets to ensure a fair evaluation of the model's performance. This division allows for effective testing and validation, enabling the identification of areas for improvement and fine-tuning of the model parameters. Additionally, cross-validation techniques can be employed to enhance reliability and minimize overfitting, ensuring that the model generalizes well to new data [**Warden:2020**].

Table 4.1.: **CNN sequence to classify gestures**

Layer (type)	Output Shape	Param#
conv2d(Conv2D)	(None, 128, 3, 8)	104
max_pooling2d(MaxPooling2D)	(None, 42, 1, 8)	0
dropout(Dropout)	(None, 42, 1, 8)	0
conv2d_1(Conv2D)	(None, 42, 1, 16)	528
max_pooling2d_1(MaxPooling2D)	(None, 42, 1, 16)	0
dropout_1(Dropout)	(None, 42, 1, 16)	0
flatten(Flatten)	(None, 224)	0
dense(Dense)	(None, 16)	3600
dropout_2(Dropout)	(None, 16)	0
dense_1(Dense)	(None, 4)	68

Training the Model

To begin our project, we first need to customize one of the example applications included in the SparkFun Edge Board Support Package (BSP) to accommodate the input of our captured dataset. As a prerequisite, follow SparkFun's "Using SparkFun Edge Board with Ambiq Apollo3 SDK" guide to configure the Ambiq SDK and the SparkFun Edge BSP. Once the initial setup is complete, modifications can be made to the example code to handle the dataset appropriately [**Laine:2022**].
After adapting the code, the program will be prepared to process the dataset as input. The next step involves building the modified application and flashing it onto the SparkFun Edge device. This ensures the program is ready for testing and data collection.
In the subsequent phase, the three group members will perform the designated motions to construct the dataset. To achieve this, open a terminal window and execute the following command:
`script output.txt`
In the terminal interface, connect to the "115200" device. Once connected, real-time measurements from the accelerometer will be displayed on the screen. These readings will also be saved in a file named `output.txt` for further processing. This process ensures an organized and accurate dataset, essential for training and validating the model. Additionally, these steps enable efficient data collection and synchronization, facilitating seamless integration into the project workflow [**Warden:2020**].
The text file will store data formatted to meet the training set's requirements. To build a high-quality dataset, each gesture is repeated multiple times to ensure sufficient variation before exiting the program. Once one group member completes this process, the next member records a similar file named `output.txt`. The logging of accelerometer data can be stopped by pressing the button labeled "14" [**Wings:2023**].
To maintain clarity, the output.txt files are renamed according to the individuals who performed the gestures. This naming convention helps differentiate datasets and ensures organized testing and validation [**Warden:2020**].
Additionally, data for the "unknown" category is collected and incorporated into the dataset. This step enables the model to classify gestures outside the predefined set as "unknown." With this inclusive data, the training process improves progressively, leading to enhanced validation accuracy over time.
The following steps are used to train the model:

- **Loading TensorBoard:** Set up TensorBoard to monitor the training process and visualize metrics.

- **Running Training Code:** Initiate the training process using scripts executed in PyCharm.
- **Data Augmentation:** Execute the data_augmentation script to enhance the training dataset by introducing variations in acceleration values, providing the model with more diverse input data.
- **Monitoring Output:** Observe the output values displayed on the screen, which include metrics such as the memory size of the model and training progress.

4.6.1. Model

In this project, the model processes a sequence of 128 three-axis accelerometer readings, corresponding to approximately five seconds of motion, and outputs an array of four probabilities: one for each predefined gesture and one for "unknown." Convolutional Neural Networks (CNNs) are employed due to their ability to capture patterns and relationships within adjacent data points effectively [**Warden:2020**]. The multi-layered CNN is designed to learn and recognize each gesture by analyzing its fundamental components. For example, it may learn to identify simple up-and-down movements and further understand how combining these with specific z- and y-axis motions forms a "wing" gesture [**Warden:2020**].
A CNN achieves this by employing a series of filters organized in hierarchical layers, where each filter is trained to recognize specific data features. In the initial layer, filters may detect basic structures like an upward acceleration. The identified features are then passed to the next layer, which combines them into more complex structures. For instance, in the "wing" gesture, the "W" shape could be identified by a sequence of four alternating upward and downward accelerations.
This hierarchical structure enables the CNN to progressively build an understanding of intricate patterns and gestures, resulting in robust and precise classification based on the input accelerometer readings.
For data acquisition, Inertial Measurement Unit (IMU) signals are utilized 4.8. The IMU is an electronic component that integrates the accelerometer. The IMU object is derived from the Arduino LSM9DS1 library, which facilitates seamless data collection and processing. This integration ensures efficient and reliable gesture data acquisition, forming the basis for effective training and testing of the CNN model.

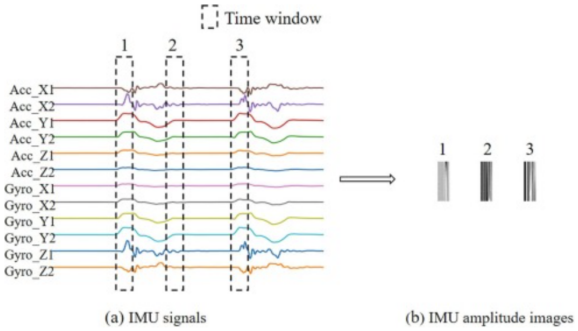


Figure 4.8.: **IMU signals [Xu:2022]**

The convolutional layer is the first component of the network to receive the raw accelerometer data as input. This data is structured into a specific shape, defined by the ‘input_shape‘ argument. The shape is ‘(seqlength, 3, 1)‘, where ‘seqlength‘ denotes the total number of accelerometer readings provided, which is 128 by default.

Each reading comprises three values corresponding to the x, y, and z axes of motion [**Xu:2022**].

This input format ensures that the network can accurately process the multidimensional nature of the accelerometer data, facilitating the extraction of relevant features for subsequent layers in the model.

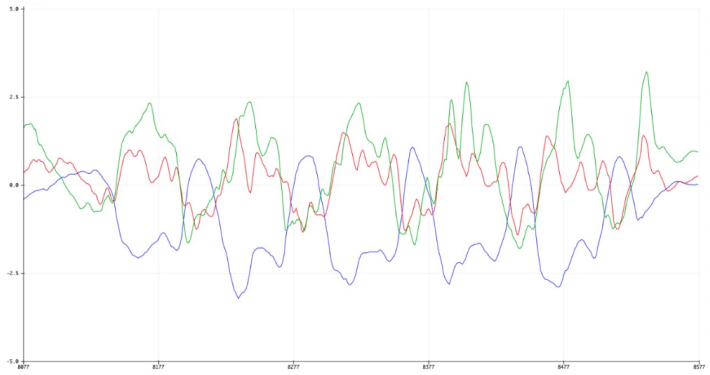


Figure 4.9.: **IMU Accelerometer Graph [Wings:2023]**

The convolutional layer is responsible for processing raw input data and identifying foundational features that subsequent layers can analyze and interpret. This is achieved through the use of the ‘Conv2D()‘ function, which specifies the parameters for feature extraction. The key parameter is the window size, defined in this instance as ‘(4, 3)‘. This configuration implies that the convolutional filter examines four consecutive accelerometer readings across all three axes.

By encompassing four consecutive measurements, each filter effectively captures and analyzes a brief snapshot of time. This allows the model to detect and represent variations in acceleration over time. The process of feature extraction using these filters is visually depicted in 4.10 [**Wings:2023**].

This capability to identify time-dependent changes forms the basis for understanding and classifying gestures, as the extracted features are subsequently passed through the network for further processing and interpretation.

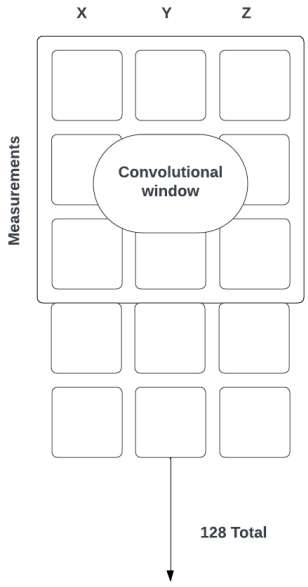


Figure 4.10.: **A convolution window overlaid on the data**

The padding argument defines how the filter window moves across the data during convolution operations. When set to "same," the layer's output dimensions remain consistent with the input, maintaining a length of 128 and a width of 3. Each movement of the filter window generates one output value, and with the "same" padding, the window slides across the width three times and down the length 128 times. Once the convolution window completes its traversal, the data is transformed into eight feature maps using the filters. These feature maps are passed to the next layer, MaxPool2D.

The MaxPool2D layer processes the (128, 3, 8) tensor output from the convolutional layer and reduces it to a smaller (42, 1, 8) tensor. This reduction is achieved by sliding a window across the data and selecting the largest value within each window, which is then passed to the output. The size of the sliding window is specified as (3, 3). By default, the window shifts to ensure it processes only new, non-overlapping data. Figure **??** demonstrates this process [**Warden:2020**].

The primary objective of a CNN is to condense a large, intricate input tensor into a smaller, simpler representation. The MaxPool2D layer contributes to this goal by summarizing the first convolutional layer's output into a concentrated, high-level abstraction of the most relevant information. This abstraction helps eliminate irrelevant details, retaining only the most significant features required to identify the gesture.

Following the pooling operation, the data passes through a Dropout layer. Dropout is a regularization method designed to mitigate overfitting by introducing noise. It randomly excludes certain data points between layers, forcing the neural network to adapt to variability and improving its robustness. This layer is active during training but inactive during inference, allowing all data to flow through at that stage [**Warden:2020**].

The Dropout layer further refines the input, distilling it into a multidimensional tensor with a shape of (14, 1, 16), representing the critical features of the input data. This process can be repeated by adding more convolutional and pooling layers, with the number of layers acting as a hyperparameter that can be adjusted. In this model, two convolutional layers were deemed sufficient for achieving the desired performance. Figure 4.11 illustrates the CNN's layer sequence.

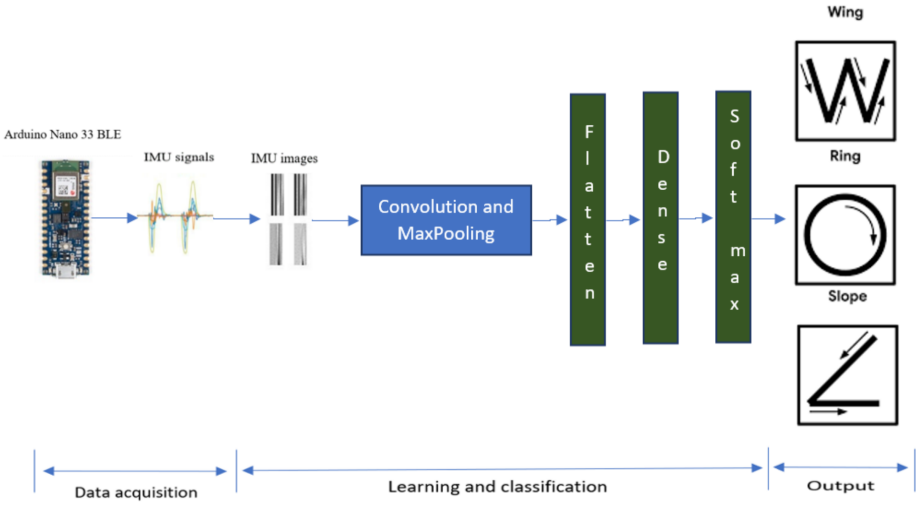


Figure 4.11.: **CNN sequence to classify Wing,Ring and Slope [Wings:2023]**

We begin by flattening the multidimensional data from the convolutional layers and feeding it into a Dense layer, also called a fully connected layer, to identify the major features in the input. The Flatten layer transforms a tensor with shape $(14, 1, 16)$ into a one-dimensional tensor with shape (224). This step condenses the multidimensional data into a single dimension, simplifying further processing.
The resulting flattened tensor is then fed into a Dense layer with 16 neurons. Each neuron in this layer is connected to every input, enabling the model to analyze all features simultaneously and learn the relationships among various combinations of inputs. The Dense layer generates a compressed representation of the input data, summarized into 16 key features [**Warden:2020**].
Next, these 16 values are reduced further to represent the four gesture classes: Wing, Ring, Slope, and Unknown. The final Dense layer contains four neurons, each corresponding to one class. These neurons are connected to all 16 outputs from the previous layer. During training, this layer learns the patterns and relationships that identify each gesture class. The layer uses a "softmax" activation function, producing output probabilities that sum to 1. This provides a probabilistic classification of the input data into the four classes.

Gesture Prediction and Validation

Once the model produces an output tensor containing gesture probabilities, a function called `PredictGesture()` ensures accurate classification by minimizing false positives. This function performs two key tasks:

1. **Threshold Check**: It verifies that the probability of the detected gesture meets a predefined minimum threshold.

2. **Inference Count Check**: It ensures the gesture is consistently detected across a required number of inferences to confirm its validity.

The number of inferences required varies based on the gesture, as each gesture takes a different amount of time to perform. These thresholds and inference requirements are specified in the `constants.cc` file [**Warden:2020**].
The `PredictGesture()` function returns a numeric value indicating the detected gesture:

- **0**: Wing
- **1**: Ring
- **2**: Slope

Workflow in `PredictGesture()`

1. The function receives the prediction scores from the main program.
2. It calculates a `maxPredictionScore` and identifies the corresponding `maxPredictionIndex`.
3. These values are compared with the `kNoGesture` and `kDetectionThreshold` constants to determine whether a valid gesture has been identified.
4. If a gesture is detected, its numeric value (equal to `maxPredictionIndex`) is assigned to the `foundGesture` variable.
5. The `foundGesture` value is returned to the main function and passed to the `output_handler()` function, which displays the recognized gesture.

This combination of convolutional and fully connected layers, paired with a robust validation function, ensures that the model accurately classifies gestures while minimizing false positives. This architecture is particularly effective for analyzing time-series sensor data, such as accelerometer readings, and enables reliable gesture recognition [**Warden:2020**].

5. Deployment

Following the KDD process, this chapter intends to explain the major components for the implementation in the Board Arduino nano 33 BLE Sense to detect specific gestures. This chapter will primarily be substantiated by previously covered topics, the deployment phase. Along with these concepts, the behavior of the board and system are also verified.

5.1. Application Description

The Magic Wand is project leveraging gesture recognition to provide intuitive and seamless control over devices. It uses the Arduino Nano 33 BLE Sense with built-in sensors (accelerometer, gyroscope, and microphone) and a TensorFlow Lite gesture recognition model.[**Daity:21**] This system interprets user gestures and translates them into commands, unlocking a wide range of applications, such as:

- **Home Automation:** Control lights, appliances, and other IoT devices.
- **Gaming and AR/VR:** Enhance interactive experiences through gesture-based controls.
- **Assistive Technology:** Empower individuals with disabilities to interact with devices.
- **Healthcare:** Enable gesture-based monitoring and control in rehabilitation.
- **Education:** Teach gesture recognition and machine learning fundamentals in classrooms.[**Daity:21**]

5.2. Structure

The deployment structure of the Magic Wand involves several interconnected components:

5.2.1. Hardware

- **Arduino Nano 33 BLE Sense:** Equipped with sensors for data collection.
- **Sensors:**
  - The accelerometer enables significant motion detection (SMD), which recognizes large-scale movements. SMD can be used to activate specific device functions, such as waking the system from sleep mode or triggering predefined actions based on significant gestures.[**Zhou:2020**]
  - The STMicroelectronics LSM9DS1 gyroscope is a precision instrument for measuring angular velocity around the x, y, and z axes. With adaptable measurement ranges of ±245, ±500, and ±2000 degrees per second (dps), the gyroscope is highly versatile for various applications such as inertial navigation, robotics, and drone stabilization.[**St:2024**]

  - a magnetometer, the x, y, and z axes (Mx, My, Mz) typically delineate the three-dimensional space in which the magnetic field is being assessed. The x-axis typically corresponds to the horizontal component of the magnetic field, the y-axis signifies the vertical component, and the z-axis reflects the magnetic field strength [**Kostiainen:2023**].

5.2.2. Software

- TensorFlow Lite provides one of the most popular model optimization techniques is called quantization.[**tensorflowlite:2025**]

- A trained gesture recognition model deployed on the Arduino Nano.

5.2.3. Data Processing

- Captures real-time sensor data.

- Preprocesses data for gesture recognition.

5.2.4. Output/Interface

- Visual feedback via serial monitor.

- Actuation of connected devices or systems.

5.3. Idea

The Magic Wand translates motion data into meaningful actions by:

1. Capturing real-time motion signals through sensors.[**Anh:2009**]

2. Preprocessing data to extract relevant features.

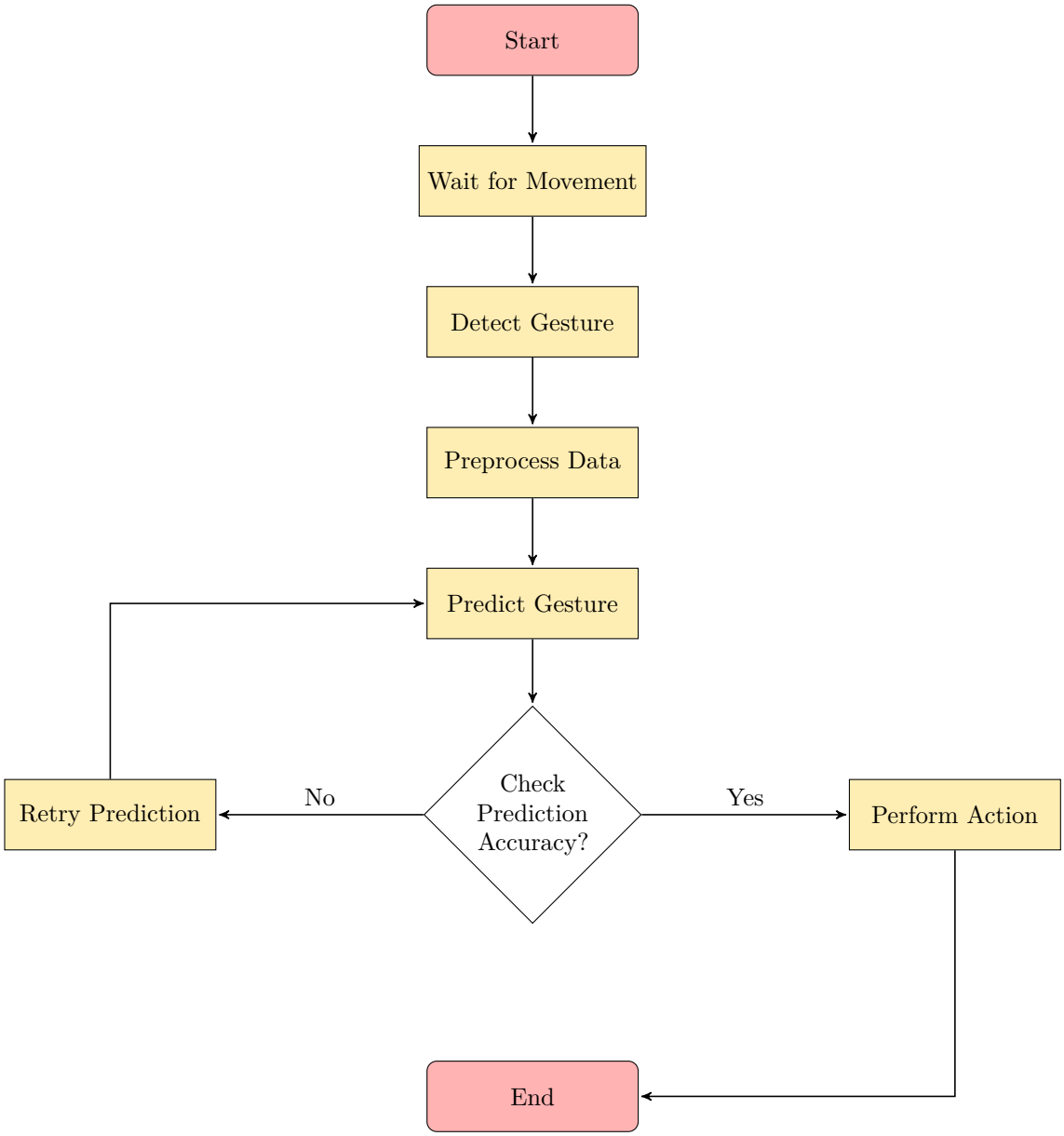
3. Feeding preprocessed data into a pre-trained ML model for classification.

4. Producing accurate gesture predictions.

5. Translating predictions into actionable commands for device control or feedback.

5.4. Flow Chart

5.4.1. Overall System Flow Chart



5.5. ML Pipeline

The ML pipeline for deploying the Magic Wand involves the following steps:

5.5.1. Step 1: Problem Definition

- Define the goal: Recognize hand gestures as specific commands.
- Input: Real-time sensor data from accelerometer and gyroscope.
- Output: Predicted gestures.

5.5.2. Step 2: Data Collection

- Collect raw sensor data for gestures using the Arduino Nano 33 BLE Sense.
- Store data in a structured format (e.g., JSON or CSV) with labeled gestures.

5.5.3. Step 3: Data Preprocessing

- Normalize sensor data to a consistent range.
- Extract meaningful features such as velocity and angular motion.
- Encode gesture labels numerically for model training.

5.5.4. Step 4: Dataset Splitting

- Divide the data into:
  - Training set: 70%
  - Validation set: 20%
  - Test set: 10%

5.5.5. Step 5: Model Design

- Choose a lightweight ML model such as a Convolutional Neural Network (CNN) or Recurrent Neural Network (RNN) optimized for time-series data.

5.5.6. Step 6: Model Training

- Train the model using the training dataset.
- Validate the model periodically to avoid overfitting.
- Save checkpoints for the best model.

5.5.7. Step 7: Model Evaluation

- Test the model on unseen data (test set).
- Evaluate accuracy, precision, recall, and F1-score.

5.5.8. Step 8: Model Optimization

- Convert the model to TensorFlow Lite format.
- Apply quantization techniques to reduce size and improve inference speed.

5.5.9. Step 9: Deployment

- Convert the quantized TensorFlow Lite model into a C header file.
- Integrate the model into the Arduino IDE.
- Upload the model to the Arduino Nano 33 BLE Sense.

5.5.10. Step 10: Testing and Debugging

- Test the system for real-time gesture recognition.
- Debug issues with sensor data, model inference, or device integration.

5.5.11. Step 11: Deployment to Real-World Application

- Use the Magic Wand for controlling devices, providing feedback, or integrating into larger systems.

5.6. TensorFlow Lite (TFLite)

5.6.1. Description

TensorFlow Lite provides one of the most popular model optimization techniques is called quantization. Quantization used to reduce the precision of the model's parameters such as weights and activation outputs into 8-bit integers.[**tensorflowlite:2025**] By default, all weights parameters are 32-bit floating-point numbers. It enables to greatly reduce the model size as 8-bit integers occupy less memory than 32-bit floating-point numbers. Although these 8-bit representations can be less precise so it turns outs little degradation in the model accuracy. TFLite supports techniques like 8-bit quantization, making models smaller and faster to run, and includes optimizations for CPUs, GPUs, and specialized accelerators.[**tensorflowlite:2025**]

5.6.2. Code Example

Listing 5.1: Running Inference with a TensorFlow Lite Model in Python

```python
import tensorflow as tf
import numpy as np

# Load the TensorFlow Lite model
interpreter = tf.lite.Interpreter(model_path="model.tflite")
interpreter.allocate_tensors()

# Get input and output tensors
input_details = interpreter.get_input_details()
output_details = interpreter.get_output_details()

# Create input data
input_data = np.array([[1.0, 2.0, 3.0]])

# Set the input tensor
interpreter.set_tensor(input_details[0]['index'], input_data)

# Run inference
interpreter.invoke()

# Retrieve the output
output_data = interpreter.get_tensor(output_details[0]['index'])
print("Model Output:", output_data)
```

5.6.3. Use of TensorFlow Lite (TFLite) in the Magic Wand Project

Quantization can take place during model training or after model training. Here we are referring to the quantization during model training as Quantization-aware training.[**tensorflowlite:2025**] LiteRT now supports converting weights to 8-bit precision as part of model conversion from TensorFlow GraphDefs to LiteRT's flat buffer format. Dynamic range quantization achieves a 4× reduction in model size. In addition, TensorFlow Lite (TFLite) supports on-the-fly quantization and dequantization of activations to allow for:

- Using quantized kernels for faster implementation when available.[**tensorflowlite:2025**]
- Mixing of floating-point kernels with quantized kernels for different parts of the graph.

The activations are always stored in floating point. For operations that support quantized kernels, the activations are quantized to 8-bit precision dynamically prior to processing and are dequantized to floating point precision after processing. Depending on the model being converted, this can provide a speedup over pure floating-point computation.

In contrast to quantization-aware training, the weights are quantized *post-training*, and the activations are quantized dynamically at inference in this method. Therefore, the model weights are not retrained to compensate for quantization-induced errors. It is important to check the accuracy of the quantized model to ensure that the degradation is acceptable.[**tensorflowlite:2025**]

- **Model Training and Conversion:** TFLite is utilized during the development phase to convert a pre-trained TensorFlow model (e.g., a gesture recognition model trained on accelerometer data) into a lightweight format. The conversion process includes techniques such as quantization, where model weights and activations are reduced from 32-bit floating-point numbers to 8-bit integers. This step ensures that the model is optimized for memory usage and inference speed, making it suitable for the microcontroller.[**tensorflowlite:2025**]
- **Testing and Validation:** TFLite is used to validate the model on a desktop or mobile device by running inference on example inputs (e.g., accelerometer data) and verifying the output classifications (e.g., *wing*, *ring*, or *slope*). This testing ensures that the model performs accurately before deployment.
- **Optimization for Edge Deployment:** Using TFLite tools, the model is fine-tuned and optimized to run efficiently on edge devices like the Arduino Nano 33 BLE Sense. Profiling and optimizations ensure that the deployed model meets the hardware constraints.[**tensorflowlite:2025**]

a TensorFlow model to train a simple Convolutional Neural Network (CNN) on the CIFAR-10 image dataset. We then evaluate the model's accuracy and save it for further quantization.

5.6.4. TensorFlow Model Code

In order to quantize model, we need a trained TensorFlow model. So, let's train a simple CNN model on cifar10 image dataset from scratch. And will compare model accuracy of original TensorFlow model and the converted model with quantization.[**tensorflowlite:2025**]

Listing 5.2: TensorFlow CNN Model for CIFAR-10

```
import tensorflow as tf
from tensorflow.keras import datasets, layers, models
import matplotlib.pyplot as plt
import numpy as np

# Check TensorFlow version
tf.__version__

# Load CIFAR-10 dataset
(x_train, y_train), (x_test, y_test) = datasets.cifar10.load_data()
# Normalize the data to range [0, 1]
x_train, x_test = x_train / 255.0, x_test / 255.0

# Define a simple CNN model
model = models.Sequential([
layers.Conv2D(32, (3, 3), activation='relu', input_shape=(32, 32, 3)),
layers.MaxPooling2D((2, 2)),
layers.Conv2D(64, (3, 3), activation='relu'),
layers.MaxPooling2D((2, 2)),
layers.Conv2D(64, (3, 3), activation='relu'),
layers.Flatten(),
layers.Dense(64, activation='relu'),
layers.Dense(10)  # Output layer for 10 classes
])
# Compile the model
model.compile(optimizer='adam',
loss=tf.keras.losses.SparseCategoricalCrossentropy(from_logits=True),
metrics=['accuracy'])
```

```
# Train the model
history = model.fit(x_train, y_train, epochs=10, validation_data=(x_test, y_test)
    ↪ )

# Evaluate model accuracy
test_loss, test_acc = model.evaluate(x_test, y_test, verbose=2)
print(f"Original Model Accuracy: {test_acc:.4f}")

# Save the model
model.save("cifar10_cnn.h5")
```

Float16 quantization reduces the model size by quantizing the model's weight parameters to float16 bit floating-point numbers for a minimal impact on accuracy and latency. This quantization technique significantly reduces the model size by half.[**tensorflowlite:2025**]
We add float16 quantization of weights while convert model into TensorFlow Lite. First set the optimizations flag to default optimizations that quantize all fixed parameters such as weights. Then specify float16 is the supported type on the target platform: By default converted model still considered input and output as a float data type. This quantization method only quantized weight parameters. However, activations are still stored in floating-point.

Listing 5.3: TensorFlow Quantization for Model Conversion

```
import tensorflow as tf

# Load the trained model
model = tf.keras.models.load_model("cifar10_cnn.h5")

### Float16 Quantization ###
converter = tf.lite.TFLiteConverter.from_keras_model(model)

# Set the optimization mode
converter.optimizations = [tf.lite.Optimize.DEFAULT]

# Set float16 as the supported type on the target platform
converter.target_spec.supported_types = [tf.float16]

# Convert and save the Float16 quantized model
tflite_model_fp16 = converter.convert()
with open("model_fp16.tflite", "wb") as f:
f.write(tflite_model_fp16)

print("Float16 Quantized Model Saved: model_fp16.tflite")

### Dynamic Range Quantization ###
converter = tf.lite.TFLiteConverter.from_keras_model(model)

# Set the optimization mode for dynamic range quantization
converter.optimizations = [tf.lite.Optimize.DEFAULT]

# Convert and save the dynamically quantized model
tflite_model_dynamic = converter.convert()
with open("model_dynamic.tflite", "wb") as f:
f.write(tflite_model_dynamic)

print("Dynamic Range Quantized Model Saved: model_dynamic.tflite")
```

5.6.5. Key Differences Between Float16 and Dynamic Range Quantization

Integer quantization Microcontroller devices, Edge TPU performs an integer-based operation. So above generated TFLite model won't compatible with integer-only hardware. To execute the TensorFlow model on integer-only hardware, we need to quantize all model parameters, input and output tensor to an integer.[**tensorflowlite:2025**]

Feature	Float16 Quantization	Dynamic Range Quantization
Weight Precision	16-bit floating point	8-bit integer
Activation Storage	Float32	Float32
Inference Speed	Slightly improved	Faster than Float16
Model Size Reduction	~2x	~4x

Table 5.1.: Comparison between Float16 and Dynamic Range Quantization

The post-training integer quantization is an optimization technique that converts both model's weights and activation outputs from 32-bit floating-point numbers to the nearest 8-bit fixed-point numbers. It also quantizes a model's input/output data. That tends to smaller model size and increased inference speed which is most suitable to deploy TensorFlow model on low-powered devices such as microcontrollers. Sometimes, it is also called full integer quantization as it converts all model parameters such as weights and activations into 8-bit integer numbers.[**tensorflowlite:2025**]
To quantize the variable data such as a model's input/output and intermediates between layers, we need to provide a RepresentativeDataset by supplying a set of input data in a generator function. This enables the converter to estimate a dynamic range for all the variable data.[**tensorflowlite:2025**]
Here, integer-based quantization must be required integer input and output tensor for compatibility. By default, the TensorFlow Lite Converter assign the model input and output tensor in a float. Applying a full integer quantization technique to convert a model into TFLite:

Listing 5.4: Full Integer Quantization for TensorFlow Model

```
import tensorflow as tf
import numpy as np

# Load the trained model
model = tf.keras.models.load_model("cifar10_cnn.h5")

# Load and preprocess CIFAR-10 dataset for representative dataset generation
(x_train, _), (_, _) = tf.keras.datasets.cifar10.load_data()
x_train = x_train.astype(np.float32) / 255.0  # Normalize

# Function to generate a representative dataset
def representative_data_gen():
for input_value in tf.data.Dataset.from_tensor_slices(x_train).batch(1).take(100)
    ↪ :
yield [input_value]

### Full Integer Quantization ###
converter = tf.lite.TFLiteConverter.from_keras_model(model)

# Set optimization mode
converter.optimizations = [tf.lite.Optimize.DEFAULT]

# Pass the representative dataset to help with activation quantization
converter.representative_dataset = representative_data_gen

# Restrict supported operations to INT8
converter.target_spec.supported_ops = [tf.lite.OpsSet.TFLITE_BUILTINS_INT8]

# Ensure input/output tensors are in INT8 format
converter.inference_input_type = tf.int8
converter.inference_output_type = tf.int8

# Convert and save the quantized model
tflite_model_int8 = converter.convert()
with open("model_int8.tflite", "wb") as f:
f.write(tflite_model_int8)

print("Full Integer Quantized Model Saved: model_int8.tflite")
```

Feature	Full Integer Quantization (INT8)
Weight Precision	8-bit integer
Activation Storage	8-bit integer
Inference Speed	Highest (suitable for Edge TPU & microcontrollers)
Model Size Reduction	~4x smaller than FP32

Table 5.2.: Key Benefits of Full Integer Quantization (INT8)

Full Integer Quantization offers significant advantages in terms of reduced model size and faster inference, making it ideal for deployment on low-power microcontrollers and Edge TPU accelerators. This method quantizes all parameters (weights and activations) to 8-bit integers, optimizing the model for edge devices.[**tensorflowlite:2025**]

5.7. TensorFlow Lite Micro (TFLite Micro)

5.7.1. Description

LiteRT for Microcontrollers is designed to run machine learning models on microcontrollers and other devices with only a few kilobytes of memory.[**litert:2023**] The core runtime just fits in 16 KB on an Arm Cortex M3 and can run many basic models. It doesn't require operating system support, any standard C or C++ libraries, or dynamic memory allocation.[**litert:2023**]

5.7.2. Code Example

Listing 5.5: Integrating TensorFlow Lite with LSM9DS1 Sensor for Gesture Detection

```
#include <TensorFlowLite.h>
#include <Wire.h>
#include <LSM9DS1.h>

// Include the generated model C array
extern "C" {
    #include "model.cc"
}

// Initialize objects
LSM9DS1 imu;
tflite::MicroInterpreter* interpreter;
tflite::Model* model;
tflite::MicroAllocator* allocator;

// TensorFlow Lite setup
void setup() {
    Serial.begin(9600);
    Wire.begin();

    if (!imu.begin()) {
        Serial.println("Failed to initialize IMU sensor!");
        while (1);
    }

    model = tflite::GetModel(model_tflite);
    interpreter = tflite::MicroInterpreter(model, allocator, tflite::
        ↪ kTensorArenaSize);

    Serial.println("Setup complete!");
}

void loop() {
    imu.readAccel();
    float input_data[] = {imu.accelX(), imu.accelY(), imu.accelZ()};

    for (int i = 0; i < 3; i++) {
```

```
            interpreter->input(0)->data.f[i] = input_data[i];
        }

        interpreter->Invoke();
        float* output = interpreter->output(0)->data.f;

        Serial.println(output[0] > 0.5 ? "Gesture Class: 1" : "Gesture Class: 0");
        delay(500);
    }
```

5.7.3. Use of TensorFlow Lite Micro (TFLite Micro) in the Magic Wand Project

TensorFlow Lite Micro (TFLite Micro) is designed specifically for running machine learning models on microcontrollers.[**litert:2023**] Its use in the Magic Wand project is detailed below:

- **Real-Time Inference on Arduino Nano 33 BLE Sense:** TFLite Micro enables the deployment of the lightweight TFLite model on the Arduino Nano 33 BLE Sense. The model is converted into a `C` array during deployment and integrated into the Arduino firmware. Real-time sensor data from the LSM9DS1 (accelerometer, gyroscope, and magnetometer) is processed by the model for gesture classification.[**litert:2023**]
- **Memory and Computational Efficiency:** TFLite Micro is optimized for devices with limited memory (e.g., 256 KB RAM) and processing power. It employs static memory allocation and int8 quantization to ensure smooth execution of the model.[**litert:2023**]
- **Gesture Recognition Flow:** The system follows these steps:
    1. **Input:** Real-time accelerometer data is captured in three dimensions (X, Y, Z).
    2. **Preprocessing:** The data is normalized and formatted to match the input requirements of the model.
    3. **Inference:** The TFLite Micro interpreter processes the input through the model, classifying the gesture as *wing*, *ring*, *slope*, or *unknown*.
    4. **Output:** Based on the classification, the Arduino controls LEDs or sends visual feedback via the serial monitor.
- **Interactivity and Deployment:** TFLite Micro ensures real-time interactivity, allowing the Arduino Nano 33 BLE Sense to respond immediately to gestures. The deployment process involves embedding the model and the TFLite Micro runtime into the Arduino program, creating a portable and efficient system.

Supported Platforms
LiteRT for Microcontrollers is written in C++ 17 and requires a 32-bit platform. It has been tested extensively with many processors based on the Arm Cortex-M Series architecture, and has been ported to other architectures including ESP32. The framework is available as an Arduino library. It can also generate projects for development environments such as Mbed. It is open source and can be included in any C++ 17 project.[**litert:2023**]
The following development boards are supported:

- Arduino Nano 33 BLE Sense
- SparkFun Edge

- STM32F746 Discovery kit
- Adafruit EdgeBadge
- Adafruit LiteRT for Microcontrollers Kit
- Adafruit Circuit Playground Bluefruit
- Espressif ESP32-DevKitC
- Espressif ESP-EYE
- Wio Terminal: ATSAMD51
- Himax WE-I Plus EVB Endpoint AI Development Board
- Synopsys DesignWare ARC EM Software Development Platform
- Sony Spresense

Explore the Examples
Each example application is on GitHub and has a `README.md` file that explains how it can be deployed to its supported platforms.[**litert:2023**] Some examples also have end-to-end tutorials using a specific platform, as given below:

- **Hello World** - Demonstrates the absolute basics of using LiteRT for Microcontrollers.
- **Tutorial using any supported device** - *Micro speech* - Captures audio with a microphone to detect the words "yes" and "no".
- **Tutorial using SparkFun Edge**
- **Person detection** - Captures camera data with an image sensor to detect the presence or absence of a person.

Workflow
The following steps are required to deploy and run a TensorFlow model on a microcontroller:

1. **Train a model:** Generate a small TensorFlow model that can fit your target device and contains supported operations.
2. **Convert to a LiteRT model:** Use the LiteRT converter to convert the model.
3. **Convert to a C byte array:** Use standard tools to store the model in a read-only program memory on the device.
4. **Run inference on the device:** Use the C++ library to run inference and process the results.[**litert:2023**]

Limitations
LiteRT for Microcontrollers is designed for the specific constraints of microcontroller development. If you are working on more powerful devices (for example, an embedded Linux device like the Raspberry Pi), the standard LiteRT framework might be easier to integrate.[**litert:2023**]
The following limitations should be considered:

- Support for a limited subset of TensorFlow operations.
- Support for a limited set of devices.
- Low-level C++ API requiring manual memory management.
- On-device training is not supported.

5.7.4. Comparison of TFLite and TFLite Micro in the Project

Aspect	TFLite	TFLite Micro
Purpose	Optimizes models for mobile/edge devices.	Executes models on microcontrollers.
Use Case in Project	Model conversion, quantization, and testing on desktop.	Real-time inference on Arduino Nano 33 BLE Sense.
Key Features	High-performance library with additional overhead.	Extremely lightweight and microcontroller-friendly.
Programming Context	Python for training and testing.	C++ for microcontroller deployment.

Table 5.3.: Comparison of TFLite and TFLite Micro in the Magic Wand Project.

5.8. Gesture Recognition

As already extensively detailed, the project's main objective is to recognize the gestures. To fully understand its implications and usage surroundings, explaining how the project's scope can be translated into multiple and more complex implementations is necessary. Currently, the board will target three gestures which are trained in the data model and give feedback on the response of these gestures.

5.8.1. Software

In the Previous Software description**??**, Installation steps have been explained comprehensively. In the following codes, some important libraries are shown, which will be required to build the software part of the project. The main packages are from the LSM9DS1 for gesture output detection and tensorflow lite for Microcontrollers to run the trained model**??**.

Listing 5.6: C++ Code for Initializing TensorFlow Lite on Arduino

```
#include <Arduino_LSM9DS1.h>
#include <TensorFlowLite.h>
#include "tensorflow/lite/micro/micro_error_reporter.h"
#include "tensorflow/lite/micro/micro_interpreter.h"
#include "tensorflow/lite/micro/micro_mutable_op_resolver.h"
#include "tensorflow/lite/schema/schema_generated.h"
#include "tensorflow/lite/version.h"
```

Here is the recap of the Tensorflow lite Micro packages used in the Arduino:

- micro_mutable_op_resolver.h provides the operations used by the interpreter to run the model.
- micro_error_reporter.h outputs debug information.
- micro_interpreter.h contains code to load and run models.
- schema_generated.h contains the schema for the TensorFlow Lite FlatBuffer model file format.
- version.h provides versioning information for the TensorFlow Lite schema.

Upon uploading the modified sketch and trained model through Tensorflow Lite Micro and the Arduino IDE, the board Arduino Nano 33 BLE Sense should exhibit gesture recognition capabilities. The recognized gestures, including wings, rings, and slope, will be displayed in the Serial Monitor of the Arduino editor during the project stage.

5.9. Configuration

After successfully fitting and uploading the model onto the Arduino Nano 33 BLE Sense board, the device operates under typical environmental conditions. The project's success depends on clearly defining the environment and conditions necessary for proper functionality. Understanding the module's surroundings is crucial. In this application, the Arduino Nano 33 BLE Sense board is connected directly to the computer via a Micro-USB cable, as explained in the Hardware section 3.2.1. This connection powers the device during operation and facilitates data exchange with the user, along with essential calibration parameters for image capturing.

5.9.1. Code Structure in Program

In our program, it consists 12 major files. These files are coded under the Arduino IDE Platform.

5.9.2. Magic_Wand.ino - isMoving()

Listing 5.7: C++ Function to Detect Movement Based on Accelerometer Data

```
bool IsMoving() {
    // Look at the most recent accelerometer values.
    const float* input_data = model_input->data.f;
    const float last_x = input_data[input_length - 3];
    const float last_y = input_data[input_length - 2];
    const float last_z = input_data[input_length - 1];

    // Figure out the total amount of acceleration being felt by the device.
    const float last_x_squared = last_x * last_x;
    const float last_y_squared = last_y * last_y;
    const float last_z_squared = last_z * last_z;
    const float acceleration_magnitude =
    sqrtf(last_x_squared + last_y_squared + last_z_squared);

    // Acceleration is in milli-Gs, so normal gravity is 1,000 units.
    const float gravity = 1000.0f;

    // Subtract out gravity to get the actual movement magnitude.
    const float movement = acceleration_magnitude - gravity;

    // How much acceleration is needed before it's considered movement.
    const float movement_threshold = 40.0f;
    const bool is_moving = (movement > movement_threshold);

    return is_moving;
}
```

Here is one of the main functions in the program main file `Magic_Wand.ino`. The above code determines whether a device is in motion based on accelerometer data. The function extracts the latest accelerometer values for the x, y, and z axes, calculates the total acceleration magnitude, and then subtracts the assumed normal gravity (1,000 units) to obtain the actual movement magnitude. The threshold for considering movement is set at 40.0 units, and the function returns true if the calculated movement magnitude exceeds this threshold, indicating that the device is in motion; otherwise, it returns false. It is essential to note that the threshold value may be application-specific and may need adjustment based on the particular context in which this code is employed.

5.9.3. Magic_Wand.ino - RecognizeGestures()

Listing 5.8: Gesture Recognition Using a Pretrained Model in C++

```
// This is the regular function we run to recognize gestures from a pretrained
```

```
    // model.
    void RecognizeGestures() {
        const bool is_moving = IsMoving();

        // Static state used to control the capturing process.
        static int counter = 0;
        static enum {
            ePendingStillness,
            eInStillness,
            ePendingMovement,
            eRecordingGesture
        } state = ePendingStillness;
        static int still_found_time;
        static int gesture_start_time;
        // State machine that controls gathering user input.
        switch (state) {
            case ePendingStillness: {
                if (!is_moving) {
                    still_found_time = counter;
                    state = eInStillness;
                }
            } break;

            case eInStillness: {
                if (is_moving) {
                    state = ePendingStillness;
                } else {
                    const int duration = counter - still_found_time;
                    if (duration > 25) {
                        state = ePendingMovement;
                    }
                }
            } break;

            case ePendingMovement: {
                if (is_moving) {
                    state = eRecordingGesture;
                    gesture_start_time = counter;
                }
            } break;

            case eRecordingGesture: {
                const int recording_time = 128;
                if ((counter - gesture_start_time) > recording_time) {
                    // Run inference, and report any error.
                    TfLiteStatus invoke_status = interpreter->Invoke();
                    if (invoke_status != kTfLiteOk) {
                        TF_LITE_REPORT_ERROR(error_reporter, "Invoke failed on index:
                            ↪  %d\n",
                        begin_index);
                        return;
                    }

                    const float* prediction_scores = interpreter->output(0)->data.f;
                    const int found_gesture = PredictGesture(prediction_scores);

                    // Produce an output
                    HandleOutput(error_reporter, found_gesture);

                    state = ePendingStillness;
                }
            } break;

            default: {
                TF_LITE_REPORT_ERROR(error_reporter, "Logic error - unknown state");
            } break;
        }

        // Increment the timing counter.
        ++counter;
    }
```

The above section would like to define a function named "RecognizeGestures" that implements a state machine to capture and recognize gestures based on accelerometer data using a pretrained model.

- **Initialization of State and Counter:**

  The function begins by initializing some static variables to control the capturing process. A counter keeps track of the number of times the function has been called, and the state is an enumeration representing the different states in the gesture recognition process.

- **Gesture Recognition State Machine:**

  The core of the function is a state machine that manages the gesture recognition process. It includes the following functions:

  – ePendingStillness: Waits for the device to become still.

  – eInStillness: Monitors the duration of stillness and transitions to the pending movement state if the stillness persists.

  – ePendingMovement: Waits for the device to start moving.

  – eRecordingGesture: Records accelerometer data for a specified duration, runs inference using a TensorFlow Lite model and produces an output based on the recognized gesture.

- **State Transitions:** The function transitions between states based on the current state and the status of motion detection (is_moving):

  – From ePendingStillness to eInStillness if the device becomes still.

  – From eInStillness to ePendingMovement if stillness persists for a specified duration.

  – From ePendingMovement to eRecordingGesture if the device starts moving.

- **Gesture Recording and Inference:** When in the eRecordingGesture state, the function records accelerometer data for a fixed duration (recording_time) and then performs inference using a TensorFlow Lite model (interpreter->Invoke()). The result is processed using the PredictGesture function.

- **Output Handling:** The recognized gesture is processed further using the HandleOutput function, and the state is transitioned back to ePendingStillness to wait for the next gesture.

- **Error Handling:** The code includes error handling, reporting an error if the state machine encounters an unknown state.

- **Counter Increment:** The counter is incremented at the end of each function call to keep track of the overall timing.

5.9.4. Magic_Wand.ino - CaptureGestureData()

Listing 5.9: Gesture Data Capture State Machine in C++

```
void CaptureGestureData() {
    const bool is_moving = IsMoving();

    // Static state used to control the capturing process.
    static int counter = 0;
    static int gesture_count = 0;
    static enum {
        eStarting,
        ePendingStillness,
        eInStillness,
        ePendingMovement,
        eRecordingGesture
    } state = eStarting;
    static int still_found_time;
    static int gesture_start_time;
```

```
        static const char* next_gesture = nullptr;
        // State machine that controls gathering user input.
        switch (state) {
            case eStarting: {
                if (!next_gesture || (strcmp(next_gesture, "other") == 0)) {
                    next_gesture = "wing";
                } else if (strcmp(next_gesture, "wing") == 0) {
                    next_gesture = "ring";
                } else if (strcmp(next_gesture, "ring") == 0) {
                    next_gesture = "slope";
                } else {
                    next_gesture = "other";
                }
                TF_LITE_REPORT_ERROR(error_reporter, "# Hold the wand still");
                state = ePendingStillness;
            } break;
        }
    }
```

The CaptureGestureData() function is to capture and process gesture data. This is crucial for the gesture recognition system within the interactive device. Key to its functionality are static variables and a boolean is_moving variable, which together track the device's motion and gesture states. All these states have all mentioned earlier in the RecognizeGestures() function 5.9.3. The key difference is that it gathers training data and does not predict any gesture or record them.

5.9.5. Magic_Wand.ino - loop()

Listing 5.10: Main Loop for Gesture Recognition or Data Capture

```
        void loop() {
            // Attempt to read new data from the accelerometer.
            bool got_data =
            ReadAccelerometer(error_reporter, model_input->data.f, input_length);
            const bool should_capture_data = false;
            if (should_capture_data) {
                CaptureGestureData();
            } else {
                RecognizeGestures();
            }
        }
```

The loop() function serves as the central operational loop for the program. It reads new data from the accelerometer to perform real-time motion tracking. The boolean got_data indicates the success of this data acquisition. Intriguingly, the function includes a conditional should_capture_data, which, although set to false, suggests a dual-mode operation: one for capturing gesture data CaptureGestureData() and another for recognizing gestures RecognizeGestures(). Setting as false value can use the program in the prediction mode (i.e. RecognizeGestures()).

5.9.6. Arduino_acceleometer_handler

Listing 5.11: IMU Data Sampling and Buffer Management in C++

```
            while (IMU.accelerationAvailable()) {
                float x, y, z;
                // Read each sample, removing it from the device's FIFO buffer
                if (!IMU.readAcceleration(x, y, z)) {
                    TF_LITE_REPORT_ERROR(error_reporter, "Failed to read data");
                    break;
                }
                // Throw away this sample unless it's the nth
                if (sample_skip_counter != sample_every_n) {
                    sample_skip_counter += 1;
                    continue;
                }
                const float norm_x = -z;
```

```
            const float norm_y = y;
            const float norm_z = x;
            save_data[begin_index++] = norm_x * 1000;
            save_data[begin_index++] = norm_y * 1000;
            save_data[begin_index++] = norm_z * 1000;
            // Since we took a sample, reset the skip counter
            sample_skip_counter = 1;
            // If we reached the end of the circular buffer, reset
            if (begin_index >= 600) {
                begin_index = 0;
            }
            new_data = true;
        }

        // Skip this round if data is not ready yet
        if (!new_data) {
            return false;
        }

        // Check if we are ready for prediction or still pending more initial
            ↪ data
        if (pending_initial_data && begin_index >= 200) {
            pending_initial_data = false;
        }

        // Return if we don't have enough data
        if (pending_initial_data) {
            return false;
        }

        // Copy the requested number of bytes to the provided input tensor
        for (int i = 0; i < length; ++i) {
            int ring_array_index = begin_index + i - length;
            if (ring_array_index < 0) {
                ring_array_index += 600;
            }
            input[i] = save_data[ring_array_index];
        }
        return true;
    }
```

The Provided code offers a detailed illustration of efficient data acquisition, filtering, and preparation methodologies. Central to this process is the continuous monitoring of acceleration data availability, as evidenced by the while loop querying IMU.accelerationAvailable(). Upon successful data retrieval, indicated by IMU.readAcceleration(x, y, z), the code employs a strategic sampling approach, selectively processing every nth data point. This method, controlled by sample_skip_counter, effectively manages the data rate, a crucial aspect in scenarios with limited processing capabilities or specific data requirements. The transformation and normalization of the data into a different coordinate system, as seen in the assignment of norm_x, norm_y, and norm_z, further exemplify the tailored preprocessing necessary for subsequent analytical stages.
The implementation of a circular buffer, as indicated by the handling of begin_index and the buffer size of 600, showcases an efficient data storage and management strategy, ensuring continuous data flow without overrunning memory limits. This approach is particularly relevant in real-time systems where older data is periodically replaced by newer inputs. The readiness of the data for further processing is meticulously managed through flags like new_data and pending_initial_data, ensuring that the system only proceeds with sufficient data accumulation.

5.9.7. Gesture_Predictor

Listing 5.12: Gesture Prediction Based on Model Output Scores

```
    // Return the result of the last prediction
    // 0: wing("W"), 1: ring("O"), 2: slope("angle"), 3: unknown
    int PredictGesture(const float* prediction_scores) {
        int max_prediction_index = -1;
```

```
            float max_prediction_score = 0.0f;
            for (int i = 0; i < kGestureCount; i++) {
                const float prediction_score = prediction_scores[i];
                if ((max_prediction_index == -1) ||
                (prediction_score > max_prediction_score)) {
                    max_prediction_score = prediction_score;
                    max_prediction_index = i;
                }
            }

            int found_gesture;
            if ((max_prediction_index == kNoGesture) ||
            (max_prediction_score < kDetectionThreshold)) {
                found_gesture = kNoGesture;
            } else {
                found_gesture = max_prediction_index;
            }

            return found_gesture;
        }
```

The above code analyzes and predicts gestures based on an array of prediction scores. Each score in the array corresponds to a different gesture, and the function's role is to determine which gesture has the highest likelihood of being correct. It operates by iterating through the array of scores, returning numeric values 0, 1, 2 and 3 for Wing, Ring, Slope and unknown respectively and maintaining a record of the highest score encountered and its associated index. This is achieved through a for-loop that compares each score with the current maximum; if a higher score is found, the function updates the maximum score and its corresponding index.
Initially the "prediction scores" obtained from the main is passed on to the function in which a "maxPredictionScore" and "maxPrediction Index". These values above are then compared to "kNoGesture" and "kDetectionThrehsold", then the value of found gesture variable is found which is equal to max prediction index.

5.9.8. Arduino_Output_Handler

Listing 5.13: LED Control Based on Gesture Recognition

```
const int ledPinRed = 22;
const int ledPinGreen = 23;
const int ledPinBlue = 24;

void LightUpRGB(int kind) {
    digitalWrite(ledPinRed, HIGH);
    digitalWrite(ledPinGreen, HIGH);
    digitalWrite(ledPinBlue, HIGH);

    switch (kind) {
        case 0: // W
        digitalWrite(ledPinRed, LOW);
        break;
        case 1: // O
        digitalWrite(ledPinGreen, LOW);
        break;
        case 2: // L
        digitalWrite(ledPinBlue, LOW);
        break;
        default:
        break;
    }
}

void HandleOutput(tflite::ErrorReporter* error_reporter, int kind) {
    // The first time this method runs, set up our LED
    static bool is_initialized = false;
    if (!is_initialized) {
        pinMode(LED_BUILTIN, OUTPUT);
        pinMode(ledPinRed, OUTPUT);
        pinMode(ledPinGreen, OUTPUT);
        pinMode(ledPinBlue, OUTPUT);
```

```
            is_initialized = true;
        }
        // Toggle the LED every time an inference is performed
        static int count = 0;
        ++count;
        if (count & 1) {
            digitalWrite(LED_BUILTIN, HIGH);
        } else {
            digitalWrite(LED_BUILTIN, LOW);
        }

        LightUpRGB(kind);
    }
```

- **Variable Declarations:** "ledPinRed, ledPinGreen, ledPinBlue:" These are constants representing the pin numbers to which the red, green, and blue LEDs are connected. The values 22, 23, and 24 indicate the specific GPIO (General Purpose Input/Output) pins used.

- **Function LightUpRGB:** This function takes an integer kind as a parameter and controls the RGB LEDs based on the value of kind. Initially, it turns all LEDs (red, green, blue) on. The switch statement then determines which LED to turn off based on the value of kind (0 for red, 1 for green, 2 for blue).

- **HandleOutput:** This function is designed to handle the output based on an input kind. It initializes the LED pins only once, as indicated by the is_initialized static variable. It toggles the built-in LED on the microcontroller on each call to indicate that an inference (or some processing) has been performed. Finally, it calls LightUpRGB(kind) to light up the appropriate LED based on the kind value.

Overall, this function serves as a state machine that captures accelerometer data, recognizes gestures using a pretrained TensorFlow Lite model, and handles the output based on the recognized gestures. The specific logic and parameters, such as stillness duration and recording time, can be adjusted based on the requirements of the application.

5.9.9. Upload the code to the board

Lastly, referring back to the Arduino IDE Configuration part**??**, the program will be verified and uploaded to the board through the Arduino IDE.

5.10. Tests

It is clear that after all the previous steps, some parameters need to be defined for the project functionality. This subsection's objective is to present the result from the implementation on one part: SW. It is important to specify this feature to understand how the system will be under corroboration in the coming steps.

5.10.1. Common things to consider during Deployment test

The reliability of the neural network to recognize the digit increases with the increase in the training data. The neural network learns to abstract to the general rule and recognizes various aspects of the gestures, such as the speed of the movement and other factors.

5.10.2. Software tests

After creating Python programs for loading and processing images, it should be tested that the intended results are achieved. Also, after fitting the model, the performance of the CNN needs to be checked to see whether it is returning the desired results. Moreover, some negative test scenarios should be tested with the help of gestures apart from the wings, rings and slope.

5.11. Monitoring stage during deployment

After the test, outputs from the board may not be accurate enough. At this moment, we should decide to modify the data set or add fresh training data. Therefore, we should monitor the model's behavior to observe and record it. Alternatively, whenever a database suffers modifications due to environment changes, it is called Monitoring. The data set for this project is unchanged mainly because it only trains three gestures, and the remaining gestures are marked as unknown if they do not belong to the three trained gestures. In order to ensure that the model is robust enough since its first development, trained models may be changed to create several versions.
The goal of this stage is to ensure that the knowledge extracted from the data is being used to make informed decisions and improve the system's overall performance. During this stage, the following activities are performed:

- **Performance monitoring:** This involves keeping a close eye on the effectiveness of the model created to interpret the data from the meters in order to make sure they are producing precise and pertinent findings.
- **Maintenance:** This entails maintaining the models and the database with fresh data and ensuring their functionality. In this particular case, since the database is "numbers," this is a very unusual case since numbers are universal.
- **Evaluation:** Finding any odd or unexpected trends in the meter data, such as sudden spikes or dips in use, is a part of this process.
- **Feedback:** This involves setting up automated alerts or notifications to let consumers know when particular criteria are satisfied, such as when the use reaches a specific threshold or a leak is discovered.
- **Continuous improvement:** To enhance the performance of the models and the whole KDD process, this entails regularly monitoring the system and making the appropriate adjustments.

The KDD process can be enhanced over time by continuously monitoring the models and the data to spot any flaws, inconsistencies, or outliers in the data. It also gives the system's users valuable insights to make better-educated decisions.

6. Software Tests

The documentation for the magic wand project encompasses detailed explanations of each module, function, and class. It provides insights into the purpose, functionality, and importance of different components. The documentation also includes test classes and methods to ensure the correctness and reliability of the implemented functionalities. The clear organization of the project into modules promotes modularity and code maintainability.
This comprehensive documentation serves as a valuable resource for project understanding, collaboration, and future development. It enhances transparency, allowing developers and stakeholders to grasp the project's architecture and functionality effectively.

6.1. Functions

- **time_wrapping(original_data, factor, num_frames)**

  The time_wrapping function takes in an original_data list, a time-wrapping factor, and the number of resulting num_frames. It performs a time-wrapping operation on the input data, creating a modified dataset with a different temporal structure. This function is utilized in the data augmentation module to introduce temporal variations in the input data, contributing to the robustness of the trained model.

- **augment_data(original_data, original_label)**

  The augment_data function performs data augmentation by generating variations of the input original_data and corresponding original_label. It creates a more diverse dataset by applying operations such as time-wrapping. The augmented data is essential for improving the model's ability to generalize to various input scenarios.

- **setUp()**

  The setUp method is part of the test classes and is executed before each test case. It initializes necessary components for testing, such as data loaders or test-specific variables, ensuring a consistent and controlled environment for testing.

- **test_time_wrapping()**

  The test_time_wrapping method is a unit test for the time_wrapping function. It checks whether the time-wrapping operation produces the expected number of frames and maintains the structure of the input data. This test ensures the correctness of the data augmentation technique.

- **test_augment_data()**

  The test_augment_data method is a unit test for the augment_data function. It verifies that the augmented data has the expected length, type, and relationship with the original data and labels. This test is crucial to ensure the effectiveness of the data augmentation process.

6.2. Classes

- **TestAugmentation**

  The TestAugmentation class is a unittest class containing test methods for the data augmentation module. It tests the functionality of the data augmentation techniques, including time-wrapping and overall data augmentation.

- **TestLoad**

  The TestLoad class is a unittest class for testing the data loading module. It checks the correct initialization of data loaders and ensures that the loaded data has the expected structure and length.

- **TestPrepare**

  The TestPrepare class is a unittest class focusing on the correctness of the dataset preparation process. It tests functions related to preparing and writing data, including generating negative data to enhance dataset diversity.

- **TestSplit**

  The TestSplit class is a unittest class that validates the dataset splitting functionality. It ensures that the dataset is correctly split into training, validation, and testing sets according to specified ratios.

- **TestTrain**

  The TestTrain class is a unittest class dedicated to testing the model training process. It covers loading data, building neural network models (CNN and LSTM), and reshaping input data to ensure compatibility with the models.

6.3. Test Files

6.3.1. Unit Tests

One of the principles of **agile** development (although not exactly being the case for our study) is that testing should be tightly integrated with development, and programmers should write tests for their own code [**Ousterhout:2018**].

- unit tests
  - most often written by developers
  - small and focused
  - are often run in conjunction with a test coverage tool[1]
    - Coverage.py
    - pytest-cov

When writing new code or modifying existing code, it is essential to update the corresponding unit tests to ensure continued code functionality and test coverage [**Ousterhout:2018**].
Unit tests facilitate refactoring [**Ousterhout:2018**]. Without a test suite

- It would be dangerous to make major structural changes to a system.
- bugs will go undetected until the new code is deployed.
  - much more expensive to find and fix

With a good set of tests, developers can be more confident when refactoring because the test suite will find most bugs that are introduced [**Ousterhout:2018**].

[1] ensures that every line of code in the application is tested.

6.3.2. dataaugmentationtest.py

This test code is responsible for verifying the functionality of the data augmentation process. Data augmentation involves creating variations of the existing dataset to enhance the model's ability to generalize. In the context of the magic wand project, it likely applies transformations to the gesture data such as rotation, scaling, or translation to create a more robust training set.
To ensure that the data augmentation techniques are correctly implemented and producing the expected variations in the input data. This helps in improving the model's performance by exposing it to diverse examples of each gesture.
`dataaugmentationtest.py`

- This test module validates the functionality of the data augmentation module used in the magic wand project. It specifically tests the **time_wrapping** and **augment_data** functions.
- **test_time_wrapping**: Tests if the time wrapping function (time_wrapping) produces the expected output dimensions and values.
- **test_augment_data**: Checks if the data augmentation function (**augment_data**) appropriately increases the dataset size and maintains the correct structure.

6.3.3. dataloadtest.py

This test code focuses on the loading of the dataset. It checks whether the dataset, which consists of gesture data (W, L, O), can be successfully loaded into the program. This includes reading image files, processing them, and organizing them for training or testing.
To validate the functionality of the data loading process, ensuring that the code can access and organize the dataset correctly before feeding it into the machine learning model.
`dataloadtest.py`

- This module focuses on testing the data loading functionality of the project using the **DataLoader** class. It ensures that the training, validation, and test datasets are loaded correctly.
- **test_get_data:** Validates the structure and lengths of the loaded training, validation, and test datasets.
- **test_pad:** Tests the padding functionality, ensuring it pads sequences correctly.
- **test_format:** Verifies that the data is formatted as expected, and TensorFlow datasets are created.

6.3.4. datapreparetest.py

The purpose of this test code is to verify the preparation of data before the training process. This may involve tasks such as normalization, resizing, or any other preprocessing steps required to make the data suitable for training.
To confirm that the data is preprocessed correctly and is in the desired format for training the machine learning model.
`datapreparetest.py`

- This test module ensures the correctness of the dataset preparation process, including the creation of negative data and writing data to files.

- **test_prepare_data:** Checks if the original data is prepared correctly and matches the expected format.
- **test_generate_negative:** Verifies the generation of negative data.
- **test_write_data:** Tests if the data is correctly written to files and can be read back successfully.

6.3.5. `datasplittest.py`

This test code is likely responsible for splitting the dataset into training, validation, and testing sets. It checks if the division is done correctly and ensures that each subset contains a representative distribution of the gestures.
To guarantee that the dataset is appropriately divided, which is crucial for assessing the model's performance during training and testing phases.
`datasplittest.py`

- This test module validates the dataset splitting functionality, ensuring that the data is divided into training, validation, and test sets as intended.
- **test_read_data:** Checks if the data is read correctly.
- **test_split_data:** Tests different scenarios of data splitting and verifies the lengths of the resulting datasets.

6.3.6. `traintest.py`

The primary purpose of this test code is to validate the training process of the machine learning model. It includes setting up the model, feeding it with the training data, and monitoring its performance during training epochs.
To ensure that the training process is functioning as expected, allowing the model to learn from the training data and adjust its parameters to make accurate predictions for the gestures.
`traintest.py`

- This module focuses on testing the training process of the machine learning model. It includes building the neural network models, loading data, and reshaping the input.
- test_load_data: Validates the loading of training, validation, and test data as TensorFlow datasets.
- test_build_net: Tests the construction of CNN and LSTM models, ensuring the correct output shapes.

  test_reshape_function: Checks if the reshape function is correctly transforming the input data.

These test codes collectively contribute to the robustness and reliability of the magic wand project by validating different components of the machine learning pipeline.

6.4. Pytest Automation for Magic Wand Project Testing

Pytest is a testing framework that simplifies the process of writing and executing test code in Python. It allows you to create test functions, organize them into test modules, and run tests efficiently. Below is an overview of how you can use Pytest to automate the testing of your Magic Wand project.

6.4.1. Install Pytest

Listing 6.1: Installing Pytest Using pip

```
pip install pytest
```

6.4.2. Folder Structure

Organize your test files into a folder named tests in your project directory. The file names should start with test to be discovered by pytest.

```
|-- magic_wand_project
|-- Modules/
|   |-- dataaugmentation.py
|   |-- dataload.py
|   |-- dataprepare.py
|   |-- datasplit.py
|   |-- train.py
|-- Tests/
|   |-- dataaugmentationtest.py
|   |-- dataloadtest.py
|   |-- datapreparetest.py
|   |-- datasplittest.py
|   |-- traintest.py
|-- handleErrors/
|   |-- errorHandler.py
```

6.4.3. Writing Test Functions

In each of your test files (e.g., dataaugmentationtest.py, dataloadtest.py, etc.), write test functions using the pytest naming convention (starting with test).

Listing 6.2: Example of Test Functions for Data Augmentation

```
# Example: test_dataaugmentation.py

def test_time_wrapping():
# Test logic ...

def test_augment_data():
# Test logic ...
```

6.4.4. Run Tests with Pytest

Open a terminal in the project directory and run:
pytest
pytest will automatically discover and execute all the test functions in the tests directory.

- **Pytest Command-Line Options**
  - Run specific test file:

    Listing 6.3: Running a Test for Data Augmentation with Pytest

    ```
    pytest tests/dataaugmentationtest.py
    ```

  - Run specific test function in a file:

Listing 6.4: Running a Specific Test with Pytest

```
pytest tests/dataaugmentationtest.py::
    ↪ test_time_wrapping
```

– Show detailed output:

Listing 6.5: Running Pytest with Verbose Output

```
pytest -v
```

- **Automation Script**

  To run all the tests together, you can create a simple automation script. For example, create a file named run_tests.sh:

Listing 6.6: Bash Script for Running Pytest in a Specific Directory

```
#!/bin/bash

# Navigate to the project directory
cd Documents/MagicWand/
ML23-06-Magic-Wand-with-an-Arduino-Nano-33-BLE-sense/
Sourcecode/Code/Datatraining

# Activate virtual environment if needed
# source venv/bin/activate

# Run pytest
pytest
```

- **Make the script executable**

Listing 6.7: Setting Executable Permission and Running the Script

```
chmod +x run_tests.sh

# Run the script
./run_tests.sh
```

6.4.5. Execution

The result of running `pytest` in the `Command Prompt` in the directory of the test files is shown in Figure 6.1.

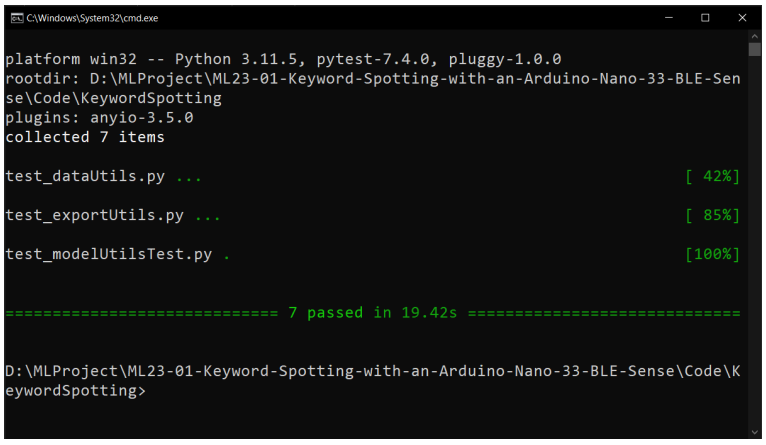


Figure 6.1.: Result of running `pytest` in the direcotry of the test files in `Command Prompt`

```
collected 7 items
```

This line indicates that `pytest` found and collected a total of 7 test items. These items represent individual test files or modules in the project. Note that the name of the files has been changed, starting with "`test_`" so that `pytest` could automatically find them.

```
test_dataUtils.py ...                [ 42%]
test_exportUtils.py ...               [ 85%]
test_modelUtilst.py .                [100%]
```

Each line corresponds to the progress of test execution for a specific test file. Dots (`...` and `.`) represent successful tests. For the case of `...`, three tests were successful, each dot representing a successful test. The percentage values ([42%], [85%], [100%]) indicate the progress through the entire test suite.

```
7 passed in 28.32s
```

This final summary line provides an overview of the test results. It states that out of the 7 tests executed, all 7 passed successfully. The total execution time for all tests was 28.32 seconds. The [100%] success rate indicates that all the tests you ran have passed.

7. Bill of Materials

7.1. Material List and Hardware Bill of Materials

Component list required for the Project are :

- Arduino Nano 33 BLE Lite Sense Board
- USB Cable
- Laptop
- Mouse
- Wand/ Stick
- Sticky Tape

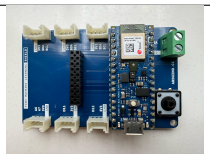
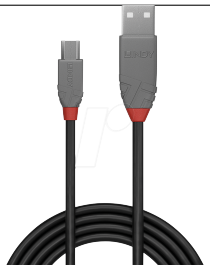
Hardware	Description	Quantity	Link	Price
	Arduino Nano board with 33 BLE Lite support and integrated sensors	1	Arduino	36,60 €
	USB Cable 3.0 fast synchronisation	1	USBCable	2,50 €
	Sticky tape	1	Tape	4,49 €
	Wand	1	Wand	3,18 €
	HP 816F9EA Notebook/Laptop	1	Laptop	629,00 €
	Mouse	1	Mouse	10,95 €

Table 7.1.: Hardware Bill of Materials

The hardware bill of materials table 7.1 outlines the essential components required for the successful execution of the project Magic Wand, each serving a specific purpose in enabling functionality and interaction.

Part IV.

Verification - Evaluation - Conclusion

8. Evaluation and Verification

Once we get the model ready, it is important to test the model with a different dataset. This helps us confirm our model will work fine with gestures from other persons as well. To test the model we need to now call the Keras's model.evaluate() function.[**Warden:2020**]
We can also use a confusion matrix to check the performance of the model. An example of a confusion matrix according to the [**Warden:2020**] is shown below which is calculated by tf.math.confusion_matrix() function:

$$\text{tf.Tensor}\left(\begin{bmatrix} 75 & 3 & 0 & 4 \\ 0 & 69 & 0 & 15 \\ 0 & 0 & 85 & 3 \\ 0 & 0 & 1 & 129 \end{bmatrix}, \text{shape} = (4,4), \text{dtype} = \text{int32}\right)$$

The confusion matrix function assesses the accuracy of the predicted class for each input in the test dataset by comparing it to the actual class. It reveals the model's limitations, pinpointing areas that need improvement. By leveraging this feedback, we can adjust the dataset and retrain the model, leading to continuous performance enhancement as the model learns from its mistakes [**Warden:2020**].

9. Results

After considerable effort in building a comprehensive dataset and training the model, the Magic Wand project, powered by the Arduino Nano BLE 33 sensor, demonstrated promising results during implementation. The gesture recognition system accurately identified and responded to predefined gestures, specifically W (Wing), O (Ring), and L (Slope). This successful integration of TinyML with the Arduino platform highlights the potential for real-time, edge-based gesture recognition in interactive applications, marking a significant step forward in utilizing machine learning for intuitive control systems.

- **Gesture Recognition:**

  The system demonstrated a strong capability in recognizing predefined gestures: W (Wing) 9.1, O (Ring) 9.3, and L (Slope) 9.5. By adding an "Unknown" category for unrecognized movements, the project gained greater flexibility and adaptability to different gestures.

- **LED Feedback:**

  Real-time visual feedback through LED lights in red **??**, green **??**, blue **??**, and white **??** reinforced the accuracy of gesture recognition. This feedback not only improved the user experience but also acted as a clear indicator of system performance.

- **Challenges Overcome:**

  The development journey wasn't without its hurdles. Initially, the model struggled with precise gesture recognition. However, these issues were addressed by consistently refining data collection methods, improving model training processes, and optimizing the code for better performance.

- **Iterative Refinement:**

  The project's success can largely be attributed to its iterative approach. Through repeated adjustments to the dataset, careful model tuning, and code improvements, the project achieved significant progress in gesture recognition accuracy.

- **Accomplishments:**

  Despite facing early challenges, the Magic Wand project is now a successful example of integrating hardware with machine learning to enable gesture-based interactions. The project demonstrates the potential for real-time, accurate gesture recognition using TinyML.

- **Visual Representation:**

  Enclosed are images that capture moments when recognized gestures are triggered, providing a visual insight into the successful implementation of the project.

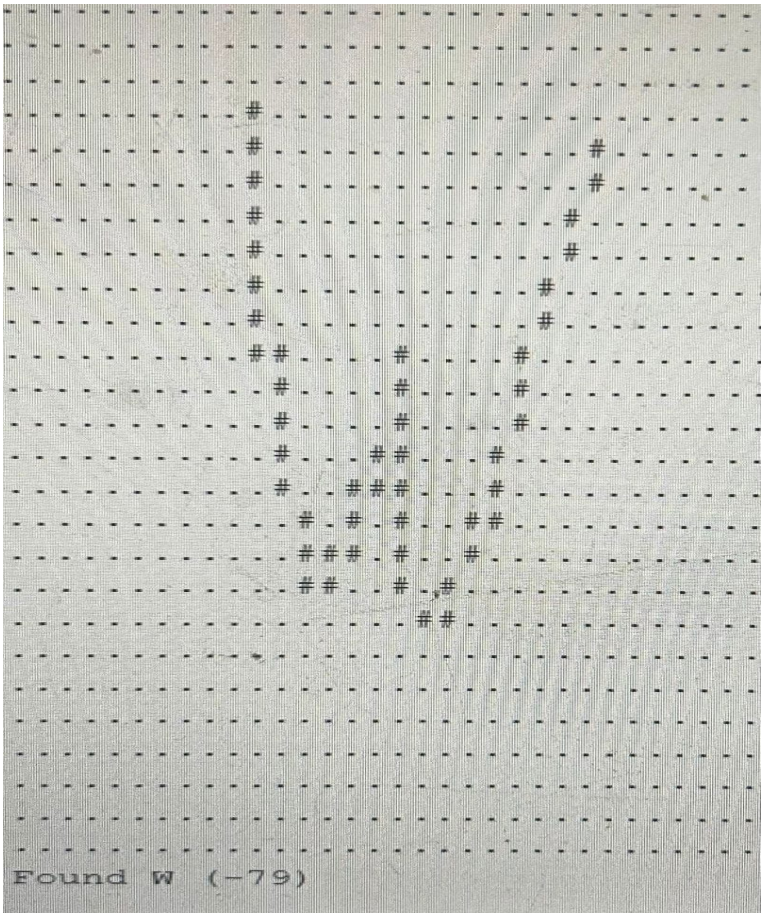


Figure 9.1.: Gesture Output for WING **W** on Output Terminal

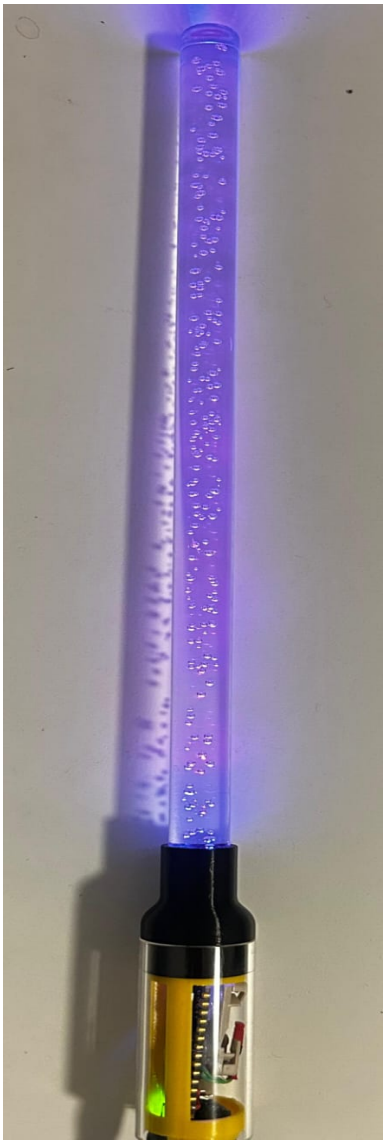


Figure 9.2.: RED Colour Output for WING **W** on Magic Wand

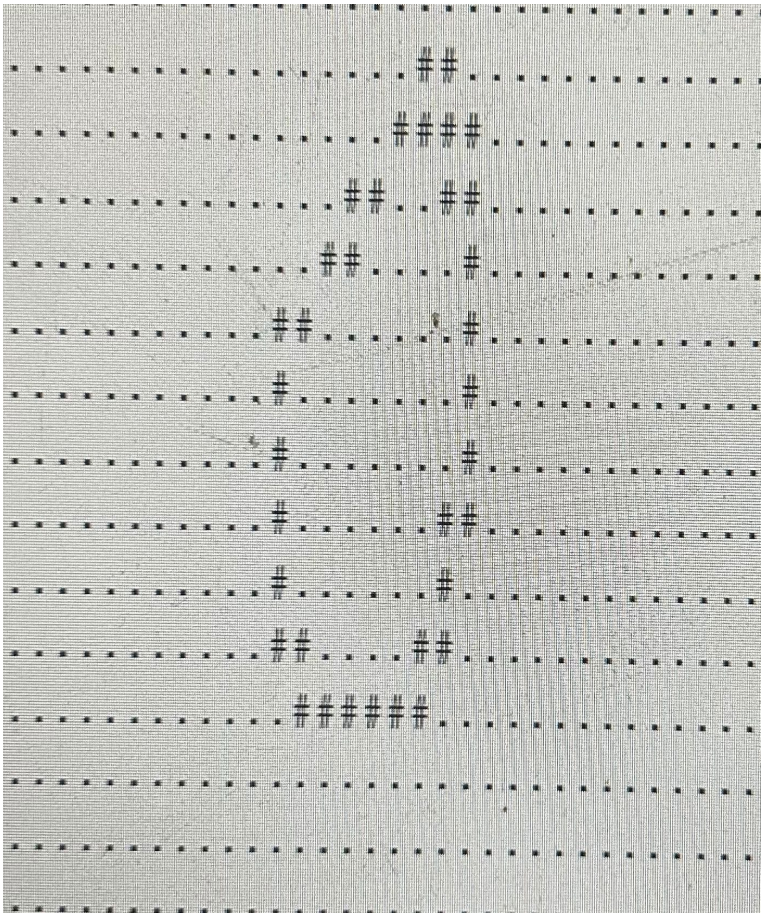


Figure 9.3.: Gesture Output for RING **O** on Output Terminal

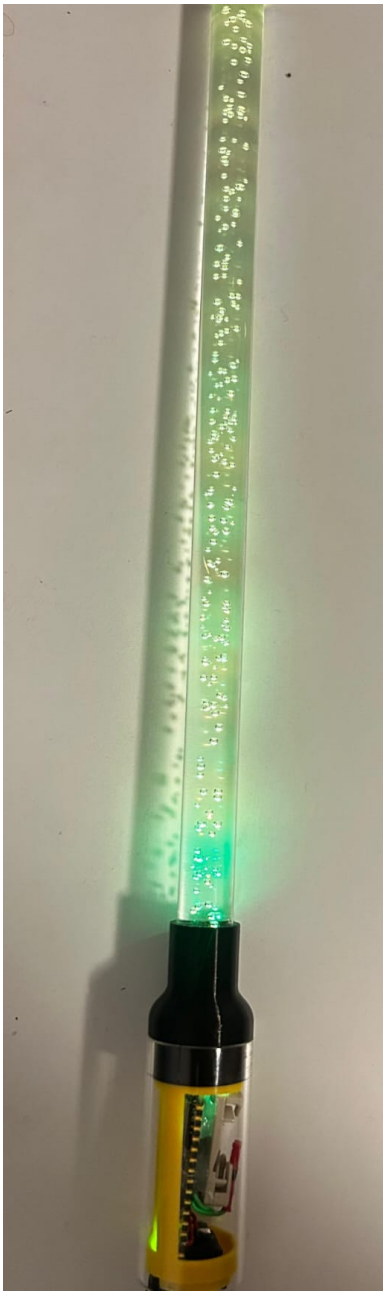


Figure 9.4.: GREEN Colour Output for RING **O** on Magic Wand

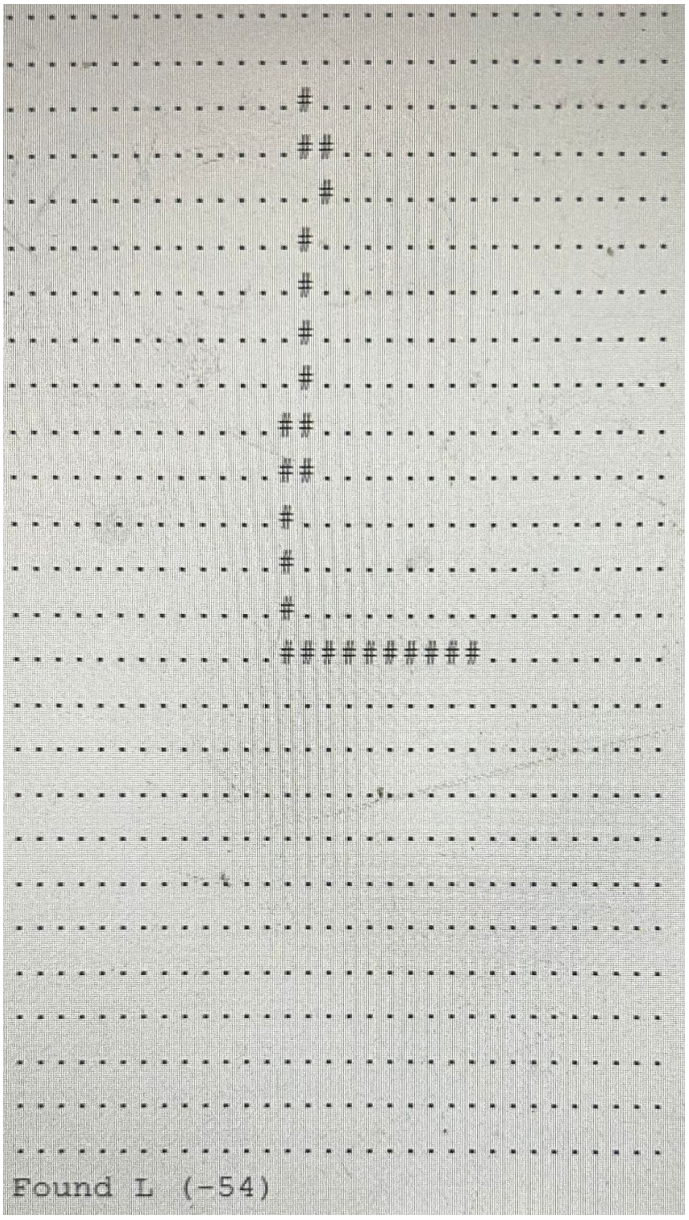


Figure 9.5.: Gesture Output for SLOPE **L** on Output Terminal

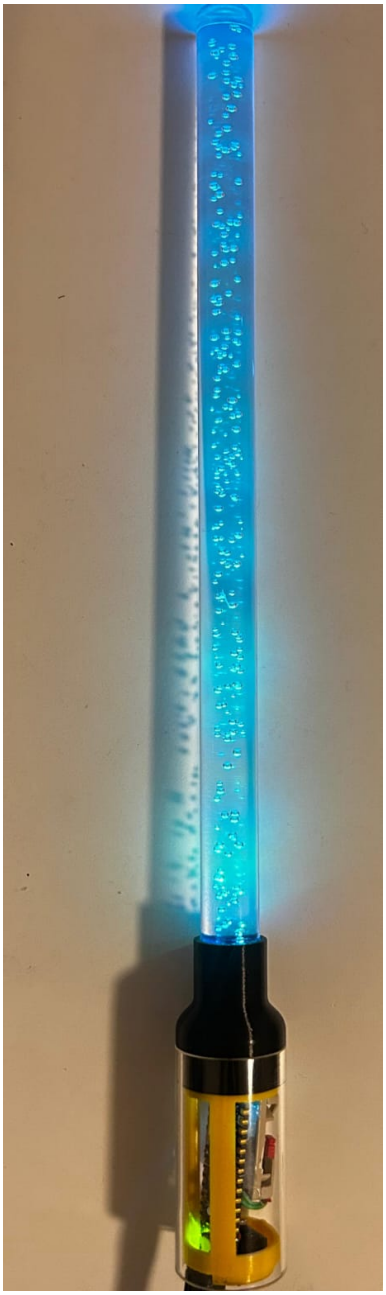


Figure 9.6.: BLUE Colour Output for SLOPE **L** on Magic Wand

10. Conclusion

In conclusion, the integration of Tiny Machine Learning (TinyML) with the Arduino Nano 33 BLE Sense has paved the way for the development of a unique and interactive "Magic Wand." This project signifies the synergy between machine learning advancements and the versatile capabilities of edge devices in the realm of the Internet of Things (IoT). The ability to process data locally, demonstrated through gesture recognition on the Arduino board, showcases the potential of edge computing in responding instantly to user inputs.
The implementation involves TensorFlow Lite, which contributes to the success of gesture recognition by utilizing a Neural Network model. The incorporation of a heuristic function adds an element of customization, allowing users to define and refine gesture knowledge without extensive programming skills. The visual output of recognized gestures and the LED response underscore the real-time interaction achieved through this integration.
Despite the challenges encountered during the planning process, such as managing the data model size within the Arduino Nano 33 BLE Sense's memory limitations and training for accuracy, the project's success demonstrates the feasibility of deploying machine learning applications on resource-constrained platforms.

1. **Accuracy Enhancement:** Focus on refining the algorithms or machine learning models to improve the accuracy and reliability of gesture recognition, minimizing both false positives and negatives. Furthermore, the script has been updated to integrate an LED output as an additional result of the gesture recognition, providing an alternate output view alongside the Arduino IDE serial output monitor.

2. **Enrichment of the testing procedures:** The testing process now specifies the exact aspects being tested (e.g., accuracy of motion detection and responsiveness of LEDs). We have documented the hardware setup, including all necessary connections and the test environment. Clear, step-by-step instructions for conducting the tests are provided, including relevant code snippets or commands. Expected outcomes are clarified, with specific values or behaviors that indicate successful testing.

3. **Optimized the code listing:** The complete code used for testing has been significantly improved, with key sections explained in detail beneath the code. This enhances readability and provides comprehensive guidance for understanding the experiments, ensuring that the process is accessible and well-documented for future reference.

4. **Enhancement of the test results:** The results from the hardware are documented thoroughly, including images of the serial monitor outputs and visual indicators used during testing to demonstrate that the code is functioning as expected. The experiments are interpreted, confirming the reliability of the tested components and providing insights into the overall system performance.

5. **Software Description Enrichment:** A sensor calibration code has been added to the software suite. Calibration is critical for ensuring accurate sensor readings by correcting any inherent hardware deviations or inaccuracies. This step not only enhances the precision of our data collection but also establishes a higher

standard for data integrity and reliability, which is essential for subsequent analyses and applications.

10.1. Additions

1. **User Feedback:** Enhance the user interaction by incorporating auditory or haptic feedback during the gesture recognition process. This improvement will provide users with a more immersive and responsive experience, making the system more intuitive to use.
2. **Expansion of Gesture Library:** Extend the current gesture library to include more complex and diverse motions, enabling the recognition of a wider range of gestures. This expansion will allow users to perform more nuanced actions, enhancing the versatility and functionality of the system.
3. **Energy Efficiency Improvements:** Introduce energy-saving features to optimize the device's battery life, which is especially critical for portable or wearable applications. This includes techniques such as power-down modes and efficient processing to reduce overall power consumption.
4. **Environmental Adaptability:** Increase the system's robustness to varying environmental conditions, such as different lighting, backgrounds, and noise levels, to maintain consistent gesture recognition performance. This will make the device more reliable in real-world settings where conditions often change.
5. **Community Engagement:** Create an online platform where users can share custom gestures and application ideas. This will foster a collaborative community around the project, enabling others to contribute, improve, and personalize their gesture-based experiences.

10.2. To-Do List

For the next phase of the Magic Wand project using the Arduino Nano 33 BLE Sense, we plan to tackle the following tasks:

1. **Design of Enclosure:**
   - Design and develop a custom enclosure to house the Arduino Nano 33 BLE Sense and the associated components.
   - Ensure the enclosure is ergonomic and facilitates easy interaction with the wand while providing sufficient protection for the internal components.
2. **Power Supply Management:**
   - Calculate the power requirements of the entire system and select an appropriate power supply that meets those needs.
   - Evaluate the possibility of making the project battery-powered and implement an efficient power management system to prolong battery life during use.
3. **Stability of Connections:**
   - Test the connection stability between the Arduino Nano 33 BLE Sense and any peripheral components to ensure reliable communication throughout the system.
   - Resolve any issues related to unstable or loose connections, ensuring smooth and uninterrupted functionality.

4. **Physical Attachment and Comfort:**
    - Securely attach the Arduino Nano 33 BLE Sense to the wand, considering factors such as weight distribution, balance, and ease of use.
    - Test the attachment to ensure that it can withstand typical handling and use without any risk of detachment or damage.
5. **Testing and Calibration:**
    - Calibrate all sensors and components to ensure that the gesture recognition system is accurate and reliable.
    - Conduct comprehensive testing of the entire hardware setup to identify and address any potential issues before final deployment.
6. **Comprehensive Documentation:**
    - Develop thorough documentation for the hardware setup, including assembly instructions, troubleshooting guides, and maintenance tips to ensure ease of use.
    - Include clear instructions on how to replace batteries (if applicable) and perform other necessary maintenance tasks to keep the system running smoothly.
7. **Safety and Compliance:**
    - Ensure that the project complies with relevant safety standards and industry regulations.
    - Introduce safety features, such as emergency shutdown procedures or safeguards, if applicable, to ensure user safety during operation.
8. **Packaging and Presentation:**
    - Design attractive and protective packaging for the Magic Wand that enhances its presentation and ensures it remains secure during shipping.
    - Consider branding and labeling options to create a professional and polished look for the final product.

10.3. Unanswered Points

1. **Device Durability:** There are still questions surrounding the long-term durability and reliability of the device under continuous use. The impact of regular usage on its components, especially over time, needs further investigation.
2. **User Customization:** The degree of customization available to users for personal preferences, such as customizing gestures and feedback mechanisms, is yet to be fully defined. For instance, our data model has been optimized for users who can perform gestures steadily and at a regular speed, but the system's performance may be impacted by users who have more erratic hand movements. This needs further analysis to address the recognition accuracy for such cases.
3. **Data Privacy and Security:** There is a need for clarification on how user data, particularly gesture inputs, are handled and protected against unauthorized access. This includes defining the measures in place to secure sensitive information.
4. **Cross-Platform Compatibility:** Uncertainties remain regarding the system's compatibility with various operating systems and devices. Further testing is required to ensure smooth integration without encountering compatibility issues across multiple platforms.

5. **Real-World Usability:** We need to explore the system's effectiveness in a variety of real-world scenarios and environments. Its performance in different lighting conditions, user settings, and diverse interaction styles has yet to be fully tested.

6. **Expansion of Gesture Recognition:** While additional gestures are being considered for inclusion in the data model to enhance recognition capabilities, the potential challenges around the model's capacity to handle diverse and complex gestures need further investigation. This includes examining the limits of gesture resistance and whether the model can effectively handle additional types.

10.4. Next Steps

In the next phase, we aim to address the unanswered points and expand the scope of the project. Key actions include gathering diverse user data and testing new, more complex gestures for broader use cases.

1. **Field Testing:** Conduct extensive field testing to assess the system's performance in real-world environments, gathering valuable user feedback to guide further improvements.

2. **Model Refinement:** Refine the machine learning models using real-world data collected from testing, aiming to enhance the accuracy and robustness of gesture recognition.

3. **Expand Application Ecosystem:** Develop and integrate new applications that take full advantage of the Magic Wand's gesture recognition capabilities, exploring a wider array of use cases and possibilities.

4. **Partnerships and Collaborations:** Seek potential partnerships with technology companies and academic institutions to explore innovative technologies and broaden the project's impact and reach.

5. **Sustainability Initiatives:** Implement strategies to ensure the sustainability of the project, such as incorporating energy-efficient design practices and using recyclable or eco-friendly materials.

10.5. Future Work

Although our current model is trained to recognize only three distinct gestures, there is considerable potential for expanding its functionality to recognize a broader range of gestures, based on evolving requirements.
Future developments of the project may include the following:

1. **LED Blinking for Specific Gestures:** Implement a feature where the LED attached to the Magic Wand blinks in response to a specific gesture. For instance, a particular gesture could trigger a distinct blinking pattern (e.g., slow blinking for "Wing" gesture, fast blinking for "Ring" gesture). This visual feedback can enhance the user experience, making the system's response more intuitive and noticeable. The LED blinking can also serve as a form of confirmation, ensuring the user that the gesture was recognized.

2. **Integration of External AI Software:** We plan to explore the possibility of using advanced AI software platforms to further train the model, thereby improving the accuracy of gesture recognition. Additionally, these platforms could help optimize the size of the data model, making it more efficient while retaining high performance.

3. **Wireless Communication Using BLE:** Leveraging the Bluetooth Low Energy (BLE) capability of the Arduino Nano 33 BLE Sense, we aim to enable wireless communication between the Magic Wand and a computer. This would enhance the system's flexibility, allowing for remote control or interaction without the need for physical connections.

4. **Advancements in TinyML:** As TinyML technologies continue to evolve, we anticipate greater advancements that can significantly improve the performance of our system. With future improvements in TinyML hardware, such as larger local memory storage and even smaller physical sizes, the capabilities of the Magic Wand could be further enhanced, enabling more sophisticated operations with minimal power consumption.

Part V.

Packages

11. Libraries/Packages List

11.1. Introduction

Data science has become a crucial component across industries such as finance and healthcare, transforming how businesses operate. The rapid increase in data generation has created a demand for tools capable of managing large and complex datasets effectively. This has driven the creation of numerous data science packages that support data manipulation, visualization, and machine learning.
These tools are indispensable for data scientists, enabling efficient data analysis, model building, and deriving insights from complex datasets. This report highlights some of the most widely used and popular packages, including **pandas**, **numpy**, **matplotlib**, **datetime**, **Sklearn**, **keras**, **lightgbm**, and **tensorflow**. These open-source tools are user-friendly, extensively documented, and accessible to both beginners and experienced professionals.

11.2. Numpy

NumPy is a versatile Python library that forms the backbone of scientific computing and data analysis. It introduces a multidimensional array object and provides a wide range of functions and tools for array operations. NumPy's efficient array structure enables the storage and manipulation of large datasets, facilitating fast numerical computations and mathematical operations.
With features like linear algebra, Fourier transforms, and random number generation, NumPy is a critical tool for scientific and data-intensive tasks [**McKinney:2012**]. Its ease of use, high performance, and seamless integration with other scientific libraries make it a preferred choice for professionals and researchers in disciplines such as physics, engineering, finance, and machine learning.

Listing 11.1: Example of importing NumPy

```
import numpy as np
```

11.3. Pandas

The name "pandas" is derived from the econometrics term "panel data," which refers to datasets that contain observations over multiple time periods for the same individuals. Additionally, the name is a creative acronym that combines "Python data analysis" with the concept of panel data [[**McKinney:2012**]].
Pandas is a highly powerful and flexible library for data manipulation and analysis, specifically designed for working with structured data, such as tabular datasets. It provides a wide array of functions to clean, merge, transform, and analyze data efficiently. Built on top of the numpy library, pandas offers two primary data structures: Series (for one-dimensional data) and DataFrame (for two-dimensional data). These structures allow for intuitive and fast data manipulation, making pandas an essential tool for anyone working with data in Python, whether for exploratory analysis, data preprocessing, or complex data transformations.

Listing 11.2: Commands to update or install a specific version of NumPy using conda

```
conda update numpy
conda install numpy=1.21.0
```

11.4. Keras

Keras is a high-level, user-friendly Python library designed to streamline the development of deep learning models. It is built to work seamlessly with back-end frameworks such as TensorFlow, Theano, or CNTK, providing a simple and efficient interface for building complex neural networks. Keras abstracts away the intricate mathematical details of tensors and their operations, allowing developers to focus on the design and architecture of neural network layers without getting bogged down in low-level implementation.
Keras offers two main APIs for model creation: the **Sequential API** and the **Functional API**. The Sequential API is particularly popular due to its simplicity, enabling the construction of a linear stack of layers where each layer is added one after the other. This approach is especially beneficial for beginners or for projects where a straightforward architecture is sufficient. On the other hand, the Functional API provides more flexibility and is ideal for creating more complex models, such as those with multiple inputs, outputs, or shared layers. Overall, Keras simplifies the development of deep learning applications, allowing users to harness the power of deep learning without dealing directly with the complexities of the underlying back-end frameworks.

11.5. TensorFlow

TensorFlow is an open-source machine learning framework developed by the Google Brain team, designed to facilitate the development and deployment of machine learning models. Keras, on the other hand, is an open-source, high-level neural network API written in Python, which runs on top of TensorFlow. While TensorFlow serves as the primary machine learning framework, Keras provides a user-friendly, high-level interface that simplifies the process of building, training, and evaluating neural networks. The Keras layers module is particularly crucial, as it allows for the easy definition and stacking of layers within a neural network, enabling efficient model design and experimentation [**TensorFlow:2023**].

Listing 11.3: Importing TensorFlow and Keras modules

```
import tensorflow as tf
from tensorflow import keras
from tensorflow.keras import layers
```

11.6. Matplotlib

Matplotlib is a versatile and powerful Python library designed for creating high-quality 2D plots and visualizations. It allows users to generate a wide variety of graph types, including line plots, bar charts, pie charts, histograms, and more. Whether for interactive or static plotting, Matplotlib offers both options and can export visualizations to multiple file formats such as PNG, PDF, and SVG. The library's extensive customization capabilities enable users to fine-tune elements such as labels, colors, and axis scales, ensuring that graphs meet specific presentation requirements. Known for its flexibility and ease of use, Matplotlib is widely adopted in both academic and professional environments, making it a standard tool for data visualization in Python [**Hunter:2007**].

Listing 11.4: Importing the Matplotlib library for plotting

```
import matplotlib.pyplot as plt
```

11.7. Sklearn

Scikit-learn, commonly referred to as sklearn, is a popular and highly regarded open-source Python library for machine learning. It offers a comprehensive suite of tools and algorithms that facilitate a wide range of tasks including data analysis, preprocessing, feature extraction, and model selection. Built on top of core scientific libraries such as NumPy, SciPy, and matplotlib, scikit-learn provides an easy-to-use and efficient interface for implementing machine learning workflows. The library includes a broad spectrum of algorithms for tasks like regression, classification, clustering, and dimensionality reduction, along with capabilities for model evaluation and selection. Thanks to its simplicity, versatility, and strong community support, scikit-learn has become a go-to choice in both academic research and industry applications, making it one of the most widely adopted and powerful tools in the Python ecosystem for machine learning.

11.8. Imagedataset

A utility function in TensorFlow/Keras designed to load and preprocess image datasets directly from a specified directory. This function automates the process of reading images, performing necessary preprocessing steps like resizing, normalization, and data augmentation, to ensure the images are in the right format for training a machine learning model. By using this function, users can efficiently prepare their image data, reducing manual effort and minimizing potential errors. It streamlines the setup process, allowing for smoother integration into the training pipeline and ensuring the data is optimized for model performance.

11.9. google.colab and IPython

Google Colaboratory, often referred to as Colab, is a cloud-based platform that enables users to write and execute Python code in an interactive environment directly from their browser. It is especially useful for tasks involving machine learning, data analysis, and other computational activities, offering free access to powerful resources like GPUs and TPUs. On the other hand, IPython (Interactive Python) is an enhanced interactive shell that allows users to execute Python code and display outputs in a command-line interface. While Colab serves as a collaborative space for running code in the cloud, IPython is commonly used within notebooks to enable advanced features like inline visualization, such as displaying images or interactive plots, enhancing the user experience during data exploration or analysis.

Listing 11.5: Importing modules from Google Colab and IPython for file handling and displaying images

```
from google.colab import files
from IPython.display import Image, display
```

12. Numpy

12.1. Introduction

NumPy extends Python's functionality by adding support for large, multidimensional arrays and matrix operations. This library enables the creation and manipulation of complex structures such as masked arrays and matrices, while offering a comprehensive set of mathematical functions for efficient array processing. These functions support various operations, including algebraic and logical computations, array reshaping, sorting, selection, input/output handling, discrete Fourier transforms, basic linear algebra, statistical functions, and random number generation. [**Harris:2023**]
Developed by Travis Oliphant in 2005, NumPy (short for Numerical Python) is an essential open-source library that plays a significant role in scientific computing and data analysis. Its primary feature, the ndarray (N-dimensional array), simplifies the management and processing of large, homogeneous datasets. Additionally, NumPy provides vital tools for linear algebra, Fourier transforms, and matrix operations, making it an integral component of many scientific computing frameworks and applications. [**NumPy:2024**]

12.2. Description

As a prominent library in the Python ecosystem, NumPy delivers strong support for large-scale, multidimensional arrays and matrices. It enhances Python by enabling efficient element-wise operations through broadcasting and offers a wide array of functions for essential mathematical tasks like linear algebra. Additionally, NumPy provides interfaces that allow code written in languages such as C, C++, and Fortran to be integrated, which is critical for performance-sensitive computing applications. [**Harris:2023**]
The power of NumPy lies in its optimized management of numerical data and its ability to execute various mathematical operations. The library excels in linear algebra and integrates smoothly with other computational libraries, offering substantial performance gains when processing large datasets. These features make NumPy an essential tool for scientific computing, data analysis, and machine learning projects.
Key Capabilities of NumPy:

1. **ndarray** NumPy overcomes the limitations of Python lists with its ndarray object, which improves data storage efficiency. While lists can store multiple data types, ndarrays ensure that each column contains a single data type, which enhances computational performance [**NumPy:2024**].

2. **Array Creation and Manipulation** NumPy provides users with the ability to create arrays of varying dimensions and predefined values using utility functions such as numpy.zeros(), numpy.ones(), and numpy.random. It also supports a wide range of array manipulation techniques, including slicing, indexing, merging, and segmentation, making complex data operations easier [**GeeksforGeeks:2017**].

3. **Mathematical Functions and Operations** The library includes an extensive set of mathematical functions optimized for array operations. These functions

cover essential arithmetic, trigonometric, exponential, and logarithmic operations, enabling efficient calculations across arrays without needing to use loops [**GeeksforGeeks:2017**].

4. **Linear Algebra** A key feature of NumPy is its robust support for linear algebra, offering operations for vectors and matrices, including matrix multiplication (numpy.dot()), inversion (numpy.linalg.inv()), eigenvalue calculation (numpy.linalg.eig()), and singular value decomposition (numpy.linalg.svd) [**GeeksforGeeks:2017**].

5. **Random Number Generation** The numpy.random module provides advanced functionality for generating random numbers and arrays that follow specific probability distributions, which is essential for statistical modeling and simulations [**GeeksforGeeks:2017**].

6. **Integration with Other Libraries** NumPy acts as the core component for many other scientific computing libraries, such as SciPy, Matplotlib, pandas, and scikit-learn, facilitating seamless data interoperability and supporting efficient analysis and visualization workflows [**Raj:2019**].

12.3. Manual

12.3.1. Installation Instructions

NumPy is compatible with both Python 2 and Python 3 versions, though support varies by version.

- For Python 2, it is advisable to use NumPy versions up to 1.16.5 as these versions maintain compatibility with this Python version.
- For Python 3, specifically versions 3.5 and later, users should employ NumPy 1.17.0 or newer versions to ensure full compatibility and access to the latest features.
- To verify the installation of Python on your machine, you can execute the following command in your terminal or command prompt:

Listing 12.1: Example command to check Python version

```
# Check the Python version
python --version
```

Installing NumPy via pip
NumPy can be installed using pip, the Python package manager, which is included by default with most Python installations. To install NumPy, run the following command:

Listing 12.2: Example command to install NumPy using pip

```
# Install NumPy using pip
pip install numpy
```

This command retrieves the latest version of NumPy from the Python Package Index (PyPI) and installs it.

Installing NumPy via Conda
If you are using the Conda package manager, it is recommended to install NumPy within a separate environment to avoid conflicts with other packages. You can follow these steps:

Listing 12.3: Example commands to create a Conda environment and install NumPy

```
# Creating a new environment named 'my-env'
conda create -n my-env

# Activating the environment (use the appropriate command based on your operating
    ↪ system)
conda activate my-env  # Windows
source activate my-env # Linux and macOS

# Optionally add Conda-forge as a channel
conda config --env --add channels conda-forge

# Install NumPy in the active environment
conda install numpy
```

12.3.2. Verifying Installation: Import NumPy

To confirm that NumPy has been installed properly, open a Python interpreter or create a new Python script and input the following commands. This will import NumPy and display the installed version, verifying that it is ready for use:

Listing 12.4: Example code to print the installed NumPy version

```
import numpy as np
print("NumPy version:", np.__version__)
```

This script checks the version of NumPy currently installed on your system, such as **numpy - 1.24.1**, and displays it.

12.3.3. Key Attributes of NumPy

NumPy's utility is defined by several core attributes that facilitate efficient data management and operations [**NumPy:2024**]:

- **shape**: This attribute provides the dimensions of an array, indicating the size along each axis as a tuple.
- **dtype**: This describes the data type of the array's elements, supporting types like integers, floats, or booleans, which aids in the consistent handling of data.
- **ndim**: This denotes the number of dimensions or axes of the array.
- **size**: It reflects the total count of elements in the array, computed as the product of the array's dimensions.
- **itemsize**: This shows the memory size in bytes for each element in the array, illustrating the storage efficiency.
- **nbytes**: This calculates the total amount of memory (in bytes) the array uses, which is the product of the size and the itemsize.
- **data**: Represents a buffer containing the actual data of the array, allowing direct access to the array's raw data.

12.3.4. Practical Examples of Attributes

Listing 12.5: Example code demonstrating NumPy array operations

```
import numpy as np

# Defining data types for structured data
dtypes = {'store_nbr': np.dtype('int64'),
    'item_nbr': np.dtype('int64'),
```

```
        'unit_sales': np.dtype('float64'),
        'onpromotion': np.dtype('O')}

    # Check for unique data
    unique_data = np.unique(data['variable'], return_counts=True)

    # Reshape and manage data
    data_array = np.arange(15).reshape(3, 5)
    reshaped_array = data_array.reshape(data_array.shape[0], -1)

    # General array creation
    array_example = np.array([[0, 1, 2, 3, 4],
    [5, 6, 7, 8, 9],
    [10, 11, 12, 13, 14]])

    # Accessing array attributes
    print("Shape:", data_array.shape)
    print("Number of dimensions:", data_array.ndim)
    print("Data type:", data_array.dtype.name)
    print("Size of each element:", data_array.itemsize)
    print("Total size of array:", data_array.size)

    # Example of array operation
    unique_values, counts = np.unique(data_array, return_counts=True)
    print("Unique values and counts:", np.array((unique_values, counts)))
```

12.4. Examples

12.4.1. Linear Algebra Fundamentals

NumPy is equipped with a vast selection of functions designed for high-level linear algebra operations. The following example illustrates how to solve a system of linear equations represented by Ax = B:

Listing 12.6: Example code to solve a system of linear equations using NumPy

```
import numpy as np

# Defining coefficients matrix A and constant vector B
A = np.array([[3, 1, -2], [1, -1, 4], [2, 0, 3]])
B = np.array([5, 6, 4])

# Computing solution vector x
x = np.linalg.solve(A, B)

print("Solution vector x:", x)
```

12.4.2. Advanced Array Operations

NumPy excels in performing vectorized operations across arrays efficiently and swiftly, surpassing traditional Python techniques.

Element-wise Operations

Operations such as addition and multiplication can be applied element-wise between arrays of identical shapes by using (a + b and a * b) formula, which is critical for mathematical and scientific computations.

Scalar Operations

NumPy enables operations between arrays and scalars seamlessly for example (5 * a). This functionality is integral for modifying data scales and normalizations efficiently.

```
    # Setup two example NumPy arrays
    a = np.array([1, 2, 3, 4])
    b = np.array([10, 20, 30, 40])

    # Demonstrating various operations
    print("Element-wise addition:", a + b)
    print("Element-wise multiplication:", a * b)
    print("Scalar multiplication:", 5 * a)
    print("Elements greater than 5:", a > 5)
    print("Sum of elements in a:", np.sum(a))

```

Listing 12.1.: Example code demonstrating basic NumPy array operations

Boolean Operations

NumPy facilitates direct array comparisons, producing boolean arrays ideal for conditional filtering. For example, a>5 checks if elements in array a are greater than 5.

Summation Functions

The library provides methods like np.sum() to perform summations across arrays rapidly, an essential tool in data analysis.

12.4.3. Advanced Broadcasting Techniques

NumPy's broadcasting feature is pivotal for performing operations on arrays of different sizes, by extending the smaller array across the larger one to align their shapes. This capability is essential for efficiently applying scalar operations across entire arrays or combining arrays of differing dimensions without the explicit replication of data.

Listing 12.7: Example code demonstrating broadcasting in NumPy

```
import numpy as np

# Example with a scalar and an array
a = np.array([1, 2, 3])
b = 2
print("Broadcasting with scalar addition:", a + b)

# Example with 2D and 1D arrays
A = np.array([[1, 2, 3], [4, 5, 6]])
b = np.array([1, 0, 1])
print("Complex broadcasting with 2D and 1D arrays:\n", A + b)
```

12.4.4. Reshaping and Flattening

Manipulating the structure of arrays is a common task in data processing, especially in contexts like machine learning where data shape directly impacts model behavior. NumPy offers several functions to reshape and flatten arrays, facilitating the reorganization of data structures.

Reshaping Arrays

Changing the shape of an array to fit specific requirements without altering the data is performed with the reshape method.

Flattening Arrays

Converting a multi-dimensional array into a 1D array is frequently necessary for processes that require a linear sequence of elements.

Listing 12.8: Example code demonstrating array reshaping and flattening in NumPy

```
import numpy as np

# Reshaping an array into a 3x3 matrix
a = np.arange(9).reshape(3, 3)
print("Reshaped to 3x3 array:\n", a)

# Flattening the array
flat_a = a.flatten()
print("Flattened array:", flat_a)

# Reshaping to 1x9 for a different view
reshaped_a = a.reshape(1, 9)
print("Reshaped to 1x9 array:\n", reshaped_a)
```

12.4.5. Stacking and Splitting Arrays

NumPy simplifies the process of combining multiple arrays into one and dividing a single array into multiple parts, crucial for data preparation and segmentation tasks.

Stacking Arrays

Both vertical and horizontal stacking are common operations that combine different datasets into a single array.

Splitting Arrays

Dividing data into manageable or required portions is often necessary in cross-validation workflows or during data analysis.

Listing 12.9: Example code demonstrating array stacking and splitting in NumPy

```
import numpy as np

# Vertical and horizontal stacking examples
a = np.array([1, 2, 3])
b = np.array([4, 5, 6])
v_stack = np.vstack((a, b))
h_stack = np.hstack((a, b))
print("Vertically stacked:\n", v_stack)
print("Horizontally stacked:", h_stack)

# Splitting an array into three parts
c = np.arange(9)
split_c = np.split(c, 3)
print("Equally split array:", split_c)
```

12.4.6. Managing NumPy Versions

Understanding the compatibility between NumPy and Python versions is crucial for maintaining stable applications. Here's an overview of how NumPy aligns with different Python environments:

- For Python 2 users, it's recommended to use NumPy versions up to 1.16.5, as these are the last releases to support Python 2.
- For Python 3, especially versions 3.5 and later, NumPy 1.17.0 or newer should be used to ensure full functionality and support.

- To verify the Python version on your machine, execute the following in your command prompt or terminal:

Listing 12.10: Command to check the installed Python version

```
python --version
```

To keep NumPy up-to-date or to install a specific version, use the package manager that was originally used for installation, typically pip for Python installations or conda for Anaconda distributions:

12.4.7. Upgrading NumPy

Utilize pip, Python's package installer, to update NumPy to the most recent version available from the Python Package Index (PyPI) or to install a specific version as needed.

Listing 12.11: Commands to upgrade or install a specific version of NumPy using pip

```
pip install --upgrade numpy
pip install numpy==1.21.0
```

Upgrading NumPy with conda
If you are using Anaconda, manage your NumPy installation using the conda command line. This approach is beneficial for handling dependencies more effectively than pip.

Listing 12.12: Commands to update or install a specific version of NumPy using conda

```
conda update numpy
conda install numpy=1.21.0
```

Maintaining compatibility with your project's dependencies during upgrades is vital. Consider the following:

- **Maintaining a Requirements File**

  Keep a 'requirements.txt' file updated to manage package versions within your project environment. After any upgrades, generate a new requirements file and thoroughly test your project to ensure all components function as expected without conflicts or deprecated issues.

Listing 12.13: Command to export installed packages to a requirements.txt file

```
pip freeze > requirements.txt
```

12.4.8. File Interaction with NumPy

NumPy offers robust functionality for file operations, allowing arrays to be saved to and loaded from the disk efficiently.
Storing and Retrieving Arrays
NumPy utilizes ".npy" as the standard file format for saving single arrays, which optimizes space and preserves data integrity. When dealing with multiple arrays, the ".npz" format is preferable as it encapsulates several arrays into one file without loss of information.

Listing 12.14: Example code to save and load arrays in .npy and .npz formats

```
# Saving a single array to a .npy file
import numpy as np
a = np.array([1, 2, 3, 4, 5])
np.save('my_array.npy', a)
```

```
# Loading the array from a .npy file
b = np.load('my_array.npy')
print("Loaded array:", b)

# Saving multiple arrays into a .npz file
a = np.array([[1, 2, 3], [4, 5, 6]])
b = np.array([7, 8, 9])
np.savez('my_arrays.npz', array1=a, array2=b)

# Retrieving arrays from a .npz file
data = np.load('my_arrays.npz')
print("First array:", data['array1'])
print("Second array:", data['array2'])
```

Handling Custom File Formats
NumPy's 'genfromtxt' function provides a versatile solution for importing data from structured text files, like CSVs. This function is especially useful for data that may contain missing values, requiring conversion between different data types or extraction based on specific criteria.

Listing 12.15: Example code to import and export data using NumPy with CSV files

```
# Importing data from a CSV file
import numpy as np
data = np.genfromtxt('my_data.csv', delimiter=',')

# Exporting an array to a CSV file
np.savetxt('array.csv', data, delimiter=',')
```

12.4.9. Error Handling in NumPy

NumPy provides robust tools for managing errors and exceptions that arise during numerical computations, particularly with floating-point operations. Here's how NumPy handles some common numerical errors:

12.4.10. Handling Floating-Point Errors

Overflow Error in Exponential Functions
Calculating the exponential of a large value with 'np.exp(1000)' triggers an overflow error because the result exceeds the range that can be represented by the floating-point type in NumPy.
Invalid Operation Error
An invalid operation error occurs with 'np.sqrt(-1)', as taking the square root of a negative number is not valid in the realm of real numbers, thus raising a FloatingPointError.
Division by Zero Error
Attempting to divide by zero using 'np.divide()' where the denominator is zero leads to a divide by zero error, which NumPy handles by raising a FloatingPointError.
Managing Runtime Warnings
When configured to 'raise' errors for invalid operations, such as division by zero, NumPy will elevate these issues to FloatingPointError exceptions. This is crucial for debugging and ensuring that numerical calculations adhere to expected standards of accuracy and reliability.

Listing 12.16: Example code demonstrating handling floating-point errors in NumPy

```
import numpy as np
np.seterr(all='raise')  # Set to raise exceptions for all types of floating-
    ↪ point errors

try:
# Attempting an operation that might cause an error
np.divide(1, 0)
except FloatingPointError as e:
```

```
        print("Caught an exception:", e)
```

12.5. Example - files

NumPy can interact with files primarily through loading and saving arrays to and from disk.

Saving and Loading Array

The standard file formats that can save array using the ".npy" format to store the array. If we would like to save multiple arrays in one file, ".npz" will be used instead of ".npy". For a single array, we can use the "save" function to store the data. For multiple arrays, we use the "savez" function to store the multiple data.

Listing 12.17: Example code for saving and loading NumPy arrays in .npy and .npz formats

```
# For single array, ".npy" format would be applied
# Example 1:
import numpy as np
a = np.array([1, 2, 3, 4, 5])
np.save('my_array.npy', a)

# Loading an array from a ".npy" file:
b = np.load('my_array.npy')
print(b)

# Example 2:
# Saving multiple arrays into a single file in NumPy .npz format:
a = np.array([[1, 2, 3], [4, 5, 6]])
b = np.array([7, 8, 9])
np.savez('my_arrays.npz', array1=a, array2=b)

# Loading multiple arrays from a .npz file:
data = np.load('my_arrays.npz')
print(data['array1'])
print(data['array2'])
```

Saving and Loading for custom file formats

NumPy's genfromtxt function is a flexible and powerful way to import data from text files, like CSV, into NumPy arrays. It allows for handling missing values, converting data types, and selecting specific columns.

Listing 12.18: Example code for loading and saving data using NumPy

```
import numpy as np
# Basic usage to load data from a CSV file
data = np.genfromtxt('my_data.csv', delimiter=',')

# Save an array to a text file, such as CSV
np.savetxt('array.csv', data, delimiter=',')
```

12.6. Further Reading

Diving deeper into NumPy and enhancing your numerical computing skills can be significantly aided by engaging with diverse educational materials. The following resources have been carefully selected to provide comprehensive learning opportunities:

12.6.1. Neural Network for Beginners: Build Deep Neural Networks and Develop Strong Fundamentals using Python's NumPy, and Matplotlib

Authored by Sebastian Klaas, this introductory guide focuses on the foundational principles and hands-on experience necessary for building neural networks. Utilizing

NumPy for computations and Matplotlib for visualization, it provides a clear path to understanding the mathematical foundations of neural networks, including key concepts like forward propagation, backpropagation, and optimization[**Sewak:2018**].

12.6.2. Effective Computation in Physics: Field Guide to Research with Python

Targeted not just at physicists but any researchers using computational methods, this book provides a robust introduction to scientific computing with Python. Covering key libraries such as NumPy, SciPy, and Matplotlib, it extends into areas like parallel computing and database management. The practical, example-driven approach makes it an excellent resource for applying Python to solve real-world scientific problems [**McKinney:2011**].

12.6.3. Python for Data Analysis

Written by Wes McKinney, the creator of pandas, this book is invaluable for those looking to understand how NumPy fits into the larger picture of data analysis with Python. Covering everything from data preparation to complex analyses, it provides a solid foundation in using NumPy and pandas together to handle, process, and analyze data effectively.

12.6.4. NumPy Official Website

As the primary resource for all things NumPy, the official documentation offers exhaustive material suitable for learners at all levels. From beginner tutorials to advanced feature guides and release notes, it's the definitive source for up-to-date and accurate information on how to utilize NumPy to its fullest potential [**NumPy:2024**].

12.6.5. NumPy Beginner's Guide - Third Edition

Ideal for newcomers to NumPy or scientific computing in general, Ivan Idris's guide is updated with the latest features of NumPy and offers a hands-on learning approach. Through detailed examples and exercises, readers can explore basic to advanced array operations, learn efficient data manipulation techniques, and apply NumPy in practical scenarios such as image processing and financial modeling [**NumPy:2024**].
Each of these resources provides unique insights and practical knowledge that can help both novices and seasoned users of NumPy expand their skills and understanding of numerical computing with Python.

13. Matplotlib

13.1. Introduction

Matplotlib is a powerful and widely used Python library for creating visualizations and plots. It provides a comprehensive set of tools for generating a wide range of static, animated, and interactive visual representations of data. Matplotlib is highly flexible and customizable, allowing users to create visually appealing and informative plots for various purposes, such as data analysis, scientific research, presentations, and more. It is built on top of NumPy, another popular Python library for numerical computing, which makes it compatible with other scientific computing libraries in the Python ecosystem.
The version of the Matplotlib package used here is 3.5.2. All these functions of the Matplotlib package are explained in the upcoming sections.

13.2. Description

Matplotlib is a fundamental tool in the data science and visualization toolkit of Python, along with libraries like Pandas and NumPy. With Matplotlib, users can create a wide range of plots, including line plots, scatter plots, bar plots, histograms, pie charts, 3D plots, and more. It provides fine-grained control over plot elements such as axes, labels, colors, markers, and legends. Additionally, Matplotlib integrates seamlessly with Jupyter notebooks, making it a popular choice for interactive data exploration and visualization.
Key features and functions of Matplotlib function:

1. **Plotting Functions:** Matplotlib provides various plotting functions to create different types of plots, such as line plots, scatter plots, bar plots, histograms, pie charts, box plots, and more. These functions allow users to visualize data in different formats and representations.

2. **Customization Options:** Matplotlib offers extensive customization options to control every aspect of a plot. Users can modify plot elements like axes, labels, titles, colors, markers, line styles, legends, and annotations. This flexibility allows users to tailor the appearance of the plots to their specific requirements.

3. **Multiple Subplots:** Matplotlib enables the creation of multiple subplots within a single figure, allowing users to display and compare different plots side by side. This feature is particularly useful when visualizing multiple datasets or different aspects of a dataset simultaneously.

4. **3D Plotting:** Matplotlib includes capabilities for creating 3D plots and visualizations. Users can generate 3D scatter plots, surface plots, wireframe plots, and other three-dimensional representations of data. These plots are helpful when working with spatial or volumetric data.

5. **Colormaps:** Matplotlib provides a wide range of built-in colormaps for mapping data values to colors. Colormaps allow users to add depth and meaning to their plots, emphasizing patterns or variations in the data. Matplotlib also supports custom colormaps, giving users flexibility in choosing the color scheme that best suits their needs.

13.2.1. Data Suitability

When it comes to data suitability, Matplotlib is a versatile library that can handle a wide range of data types and formats.

- **Numerical Data:** Matplotlib is particularly well-suited for visualizing numerical data. Whether it's a simple line plot, scatter plot, histogram, or more complex plots like contour plots or heatmaps, Matplotlib provides the necessary tools to effectively represent and analyze numerical data. It offers customization options to highlight trends, patterns, and relationships within the data.

- **Categorical Data:** While Matplotlib is primarily designed for numerical data, it also supports visualizations for categorical data. Bar plots, pie charts, and stacked bar plots are useful for representing distributions, proportions, or comparisons among different categories. Matplotlib allows for customization of colors, labels, and other visual elements to enhance the understanding of categorical data.

- **Time Series Data:** Matplotlib is commonly used to visualize time series data. It provides various plot types suitable for displaying temporal trends, such as line plots, area plots, or candlestick plots. With Matplotlib, users can easily add date and time axes, format tick labels, and annotate events or significant time points. This makes it a valuable tool for analyzing and presenting data that changes over time.

- **Spatial Data:** Matplotlib also supports the visualization of spatial data. By leveraging its 3D plotting capabilities or using specialized modules like Basemap or Cartopy, Matplotlib can create maps, contour plots, and geospatial visualizations. It allows users to plot points, lines, polygons, and other geometric shapes on maps, making it suitable for analyzing and displaying spatial relationships.

- **Data Size:** Matplotlib can handle data of varying sizes, from small to large datasets. While it performs well with smaller datasets, it may face limitations in terms of performance and interactivity with extremely large datasets. In such cases, users may need to optimize the code, use data sampling techniques, or explore other specialized libraries for big data visualization.

- **Data Preprocessing:** Before visualizing data with Matplotlib, it is often necessary to preprocess and format the data appropriately. Matplotlib expects data to be in a suitable format, such as NumPy arrays or Pandas data structures. Therefore, users may need to apply data manipulation techniques using libraries like NumPy and Pandas to transform the data into the desired format before plotting.

13.3. Installation

13.3.1. Installing an official release

Matplotlib and its dependencies are available as wheel packages for macOS, Windows and Linux distributions:

Listing 13.1: Installing/upgrading pip and matplotlib

```
python -m pip install -U pip
python -m pip install -U matplotlib
```

13.3.2. Installing with Anaconda

Matplotlib is available both via the anaconda main channel

Listing 13.2: Installing matplotlib using conda

```
conda install matplotlib
```

as well as via the conda-forge community channel

Listing 13.3: Installing matplotlib from conda-forge

```
conda install -c conda-forge matplotlib
```

13.3.3. Required dependencies

Matplotlib requires the following dependencies:

Table 13.1.: Python Library Dependencies

Library	Minimum Version Required
Python	$\geq$ 3.6
FreeType	$\geq$ 2.3
libpng	$\geq$ 1.2
NumPy	$\geq$ 1.11
setuptools	–
cycler	$\geq$ 0.10.0
dateutil	$\geq$ 2.1
kiwisolver	$\geq$ 1.0.0
pyparsing	–

13.4. Example - Manual

13.4.1. General Concepts

"matplotlib" has an extensive codebase that can be daunting to many new users. However, most of matplotlib can be understood with a fairly simple conceptual framework and knowledge of a few important points.
Plotting requires action on a range of levels, from the most general (e.g., 'contour this 2-D array') to the most specific (e.g., 'color this screen pixel red'). The purpose of a plotting package is to assist you in visualizing your data as easily as possible, with all the necessary control – that is, by using relatively high-level commands most of the time, and still have the ability to use the low-level commands when needed.
Therefore, everything in matplotlib is organized in a hierarchy. At the top of the hierarchy is the matplotlib "state-machine environment" which is provided by the matplotlib.pyplot module. At this level, simple functions are used to add plot elements (lines, images, text, etc.) to the current axes in the current figure.
The next level down in the hierarchy is the first level of the object-oriented interface, in which pyplot is used only for a few functions such as figure creation, and the user explicitly creates and keeps track of the figure and axes objects. At this level, the user uses pyplot to create figures, and through those figures, one or more axes objects can be created. These axes objects are then used for most plotting actions.
For even more control – which is essential for things like embedding matplotlib plots in GUI applications – the pyplot level may be dropped completely, leaving a purely object-oriented approach.

How to import

To import functions from the Matplotlib library, you can use the import statement in Python. Here's how you can *import* Matplotlib functions:

Listing 13.4: Importing matplotlib for plotting

```
import matplotlib.pyplot as plt
```

Figure

The whole figure. The figure keeps track of all the child Axes, a smattering of 'special' artists (titles, figure legends, etc), and the canvas. (Don't worry too much about the canvas, it is crucial as it is the object that actually does the drawing to get you your plot, but as the user it is more-or-less invisible to you). A figure can have any number of Axes, but to be useful should have at least one.
The easiest way to create a new figure is with pyplot:

Listing 13.5: Creating a figure with matplotlib

```
fig = plt.figure()  # an empty figure with no axes
fig.suptitle('No axes on this figure')  # Add a title so we know which it is

fig, ax_lst = plt.subplots(2, 2)  # a figure with a 2x2 grid of Axes
```

Axes

This is what you think of as 'a plot', it is the region of the image with the data space. A given figure can contain many Axes, but a given Axes object can only be in one Figure. The Axes contains two (or three in the case of 3D) Axis objects (be aware of the difference between Axes and Axis) which take care of the data limits (the data limits can also be controlled via set via the set_xlim() and set_ylim() Axes methods). Each Axes has a title (set via set_title()), an x-label (set via set_xlabel()), and a y-label set via set_ylabel()).
The Axes class and its member functions are the primary entry point to working with the OO interface.

Axis

These are the number-line-like objects. They take care of setting the graph limits and generating the ticks (the marks on the axis) and ticklabels (strings labeling the ticks). The location of the ticks is determined by a Locator object and the ticklabel strings are formatted by a Formatter. The combination of the correct Locator and Formatter gives very fine control over the tick locations and labels.

Artist

Basically everything you can see on the figure is an artist (even the Figure, Axes, and Axis objects). This includes Text objects, Line2D objects, collection objects, Patch objects ... (you get the idea). When the figure is rendered, all of the artists are drawn to the canvas. Most Artists are tied to an Axes; such an Artist cannot be shared by multiple Axes, or moved from one to another.

13.4.2. Types of inputs to plotting functions

All of plotting functions expect `np.array` or `np.ma.masked_array` as input. Classes that are 'array-like' such as `pandas` data objects and `np.matrix` may or may not work as intended. It is best to convert these to `np.array` objects prior to plotting.
For example, to convert a **`pandas.DataFrame`**

Listing 13.6: Creating a DataFrame and converting it to a NumPy array

```
a = pandas.DataFrame(np.random.rand(4,5), columns = list('abcde'))
a_asarray = a.values
```

and to convert a `np.matrix`

Listing 13.7: Converting a NumPy matrix to an array

```
b = np.matrix([[1,2],[3,4]])
b_asarray = np.asarray(b)
```

13.4.3. Example

Listing 13.8: Example of a simple line plot using Matplotlib

```
import matplotlib.pyplot as plt

# Sample data
x = [1, 2, 3, 4, 5]
y = [2, 4, 6, 8, 10]

# Create a figure and axis
fig, ax = plt.subplots()

# Plot the data
ax.plot(x, y, marker='o', linestyle='-', color='b', label='Data')

# Set labels and title
ax.set_xlabel('X-axis')
ax.set_ylabel('Y-axis')
ax.set_title('Line Plot')

# Add gridlines
ax.grid(True)

# Add legend
ax.legend()

# Display the plot
plt.show()
```

1. In this code example, we import the necessary modules from Matplotlib ('pyplot') and create sample data ('x' and 'y').

2. Create a figure and axis:

   Listing 13.9: Creating a subplot with Matplotlib

   ```
   fig, ax = plt.subplots()
   ```

   This creates a figure object ('fig') and an axis object ('ax'). The figure is the overall window or canvas, while the axis is the area within the figure where the plot is drawn.

3. Plot the data:

   Listing 13.10: Plotting data with a line plot

   ```
   ax.plot(x, y, marker='o', linestyle='-', color='b', label='Data')
   ```

   The 'plot()' function is used to create a line plot. Here, we specify the x and y data, marker style, line style, color, and label for the plot.

4. Set labels and title:

Listing 13.11: Setting labels and title for the plot

```
ax.set_xlabel('X-axis')
ax.set_ylabel('Y-axis')
ax.set_title('Line Plot')
```

The set_xlabel(), set_ylabel(), and set_title() functions are used to set the labels for the x-axis, y-axis, and the plot title, respectively.

5. Add legend:

Listing 13.12: Adding legend to the plot

```
ax.legend()
```

The 'legend()' function is used to add a legend to the plot, which shows labels for different elements of the plot.

6. Display the plot:

Listing 13.13: Displaying the plot

```
plt.show()
```

The 'show()' function is used to display the plot.

13.4.4. Matplotlib Error Handling

Error handling in Matplotlib, like in any other Python library, involves using try-except blocks to catch and handle potential errors that may occur during the execution of your code. Here's an example of how one can handle errors in Matplotlib:

Listing 13.14: Line Plot Example with Error Handling

```
import matplotlib.pyplot as plt

# Sample data
x = [1, 2, 3, 4]
y = [2, 4, 6]

try:
# Attempt to create a line plot
plt.plot(x, y)
plt.xlabel('X-axis')
plt.ylabel('Y-axis')
plt.title('Line Plot')
plt.show()

except ValueError as ve:
# Handle a specific type of error (ValueError in this case)
print(f"ValueError: {ve}")
# Take appropriate action to handle the error

except Exception as e:
# Handle any other type of exception
print(f"An error occurred: {e}")
# Take appropriate action to handle the error
```

In this example, a try-except block is used to handle potential errors that may occur during the creation and display of the line plot. Here's how the error handling works:

1. The code within the try block attempts to create a line plot using 'plt.plot()' and adds labels, title, and other plot elements.

2. If an error occurs during this process, the corresponding exception will be raised. In this example, we handle two types of exceptions:

a) **'ValueError':** This is a specific type of error that may occur if the input data for the plot is invalid or incompatible. We use the 'ValueError' as an example here, but you can handle other specific exceptions as needed.

b) **'Exception':** This is a generic exception that can handle any other type of exception that may occur during the execution of the code.

3. Inside each except block, you can define specific actions to be taken when the corresponding exception is encountered. In this example, we print out a customized error message and handle the error appropriately. You can replace the print statements with your desired error handling logic, such as logging the error, displaying a user-friendly message, or taking corrective actions.

13.5. Further Reading

13.5.1. Matplotlib Official Documentation

The official documentation is an extensive resource that provides comprehensive information about the Matplotlib library. It includes a user guide, API reference, tutorials, and examples. You can access it at:
`https://matplotlib.org/stable/users/index.html`

13.5.2. Mastering matplotlib

"Mastering Matplotlib" by [**McGreggor:2015**] is a comprehensive guide that empowers readers to become proficient in creating professional-grade plots and visualizations using the Matplotlib library in Python. This book covers fundamental concepts, advanced techniques, and practical aspects to enhance data visualization skills. It can be accessed through: `https://www.packtpub.com/product/mastering-matplotlib/9781783987542`

13.5.3. Scientific Visualization: Python + Matplotlib

"Scientific Visualization: Python + Matplotlib" by [**Rougier:2021**] is a practical guide that demonstrates the use of Python and Matplotlib for scientific data visualization, catering to scientists, researchers, and data analysts. It offers insights into visualization techniques and customization to effectively present scientific data. It can be accessed through: `https://inria.hal.science/hal-03427242/`

13.5.4. Python for Data Analysis

This book, written by Wes McKinney (the creator of Pandas) [**Mckinney:python**], is a valuable resource for learning Pandas. It covers various aspects of data manipulation, analysis, and visualization using Pandas. The book also explores practical examples and real-world use cases. Find it here:
`https://www.oreilly.com/library/view/python-for-data/9781491957653/`

Index