WoodwindMania

Judenique Auguste, Vivian Beaudoin, Angel Peprah

Department of Electrical and Computer Engineering, Carnegie Mellon University

***Abstract*—A digital flute input device (flute controller) that mimics the functionality of a real flute, accompanied by an application that allows beginners to learn flute in a cheap and effective way. The controller and app allow the user to learn fingerings, breath control, and posture while also learning scales, notes, and basic music theory with the web application that communicates wirelessly with the controller and gives live feedback.**

***Index Terms*—Arduino, Bluetooth Packet, Breath Detection, Django, Music, Python, Raspberry Pi, Sonic Pi, Web Application**

I. Introduction

Learning an instrument can be daunting, inconvenient, and expensive for beginners. It can easily cost thousands to gain access to a good quality instrument and consistent high-quality lessons. This can be a discouraging aspect of learning how to play a new instrument, especially if the user simply wants to try out the craft or is not sure about committing for a long period of time.

To help solve this problem, we invented WoodwindMania, a digital way to learn flute that is more cost-effective and easier to use for beginners. This product allows beginners to interact with a flute controller that is similar to an actual flute. Additionally, it allows for a seamless transition to a real flute in the case that the user decides that they would like to pursue the instrument more seriously after mastering the basic skills.

The flute controller has the same dimensions as a real flute, with buttons located where they normally are. In addition, the user has to blow into the device using the correct technique to create a sound. Lastly, the device tracks its position, so the user must be holding it correctly. This ensures that the user starts to learn the correct skills associated with playing the flute outside of playing the correct notes.

The application displays feedback to the user and allows them to learn the correct fingerings of notes from E4 to Db6. The application also teaches the user nine scales and tests the user on them. There is also a free play mode in which the user can play without feedback.

In terms of competing technologies, there are no direct products that are aimed at beginners. There are electronic wind controllers that output Musical Instrument Digital Interface (MIDI) that are aimed at musicians who want to add wind instruments to their projects, but they do not teach the user anything about the instrument or give feedback to the user about what is being played.

The main goal of this project is a budget-friendly system that allows beginners to learn how to play flute in a comprehensive way. Users are able to learn this new instrument conveniently and at their own pace.

II. Use-Case Requirements

A. Accuracy

The user is looking to use the flute controller to replace a traditional instrument. Therefore, we expect the accuracy of our system to be reasonably high. The accuracy of feedback for fingerings, breath control, and orientation must be greater than 90%.

B. Speed

The input from the user is sent wirelessly from the Arduino Nano on the flute controller to a Raspberry Pi that processes the user input and hosts the web app locally. The combined latency of Arduino processing, Bluetooth communication, and RPi processing between the user input and displaying feedback in the application must not be longer than 500 milliseconds on average. This corresponds to a playing speed of 120 beats per minute, which is the upper-end speed of what we expect a beginner would like to play.

For the same reasons, we also require that the latency between user input and audio feedback on the RPi is no longer than 500 milliseconds on average.

C. User Experience

The dimensions of the flute controller must be as close to a real flute as possible. This means it must measure 26 inches in length and have a 1-inch diameter. It also must be around 1.13 pounds. The controller should also be wireless.

Secondly, user satisfaction (collected in the form of a survey) must be greater than an average score of 4/5, or 80%, as between 75% and 85% user satisfaction is generally considered a good rate [9]. Additionally, the device and application should be very beginner-friendly, with no assumed knowledge of the flute or how to play one expected from the users of the device.

Lastly, the device should save the user money. This means the total price of the device and web application should be less than the expected $500 for a beginner flute plus a month of lessons.

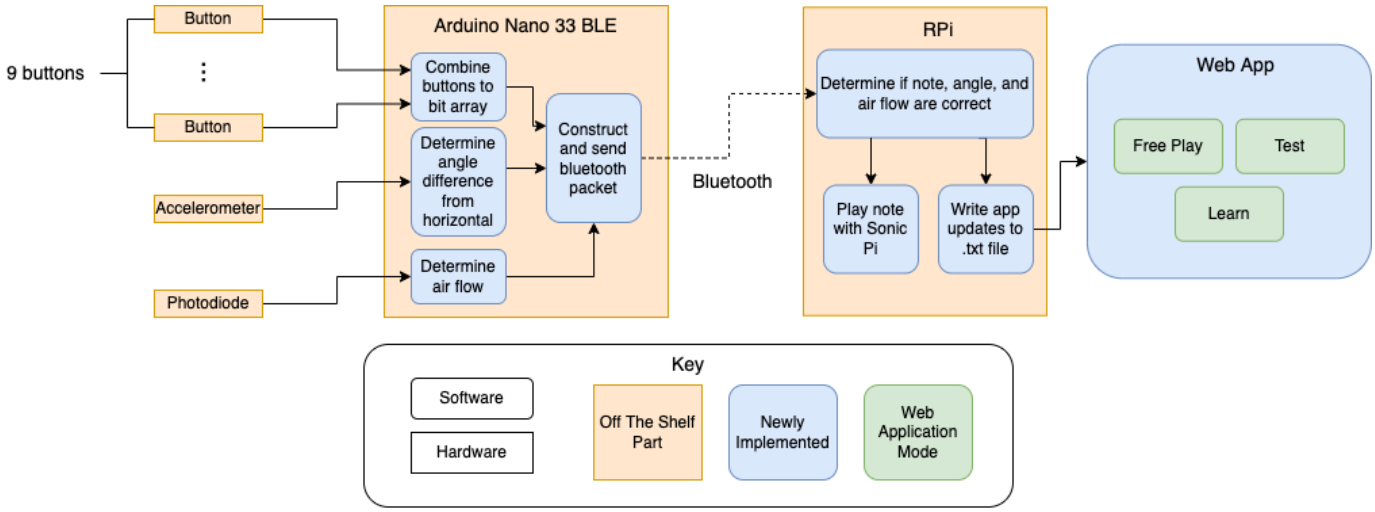


Fig. 1: System Overview

III. Architecture and/or Principle of Operation

There are three main systems in the design of WoodwindMania: the physical controller, the wireless communication, and the web application. The block diagram in Fig. 1 represents the overall system architecture. The controller captures the user input of the fingerings, orientation, and breath control and then sends this data wirelessly to a Raspberry Pi using BLE communication. The RPi receives this data and plays the corresponding note based on the fingering and breath control amplitude. It then sends the data to the web application where it displays the data for feedback.

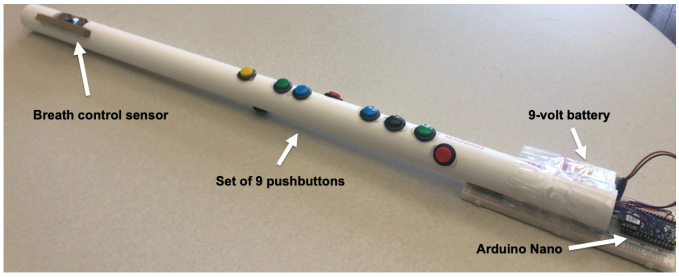


Fig. 2: Physical controller

A. Physical Controller

Our physical controller, shown in Fig. 2, consists of a collection of sensors encased in a PVC pipe. We have nine tactile push buttons mounted in the position of the keys on a flute, a breath control sensor, our Arduino Nano that also contains an accelerometer for orientation reading, and a nine-volt battery. The buttons determine the note fingering the user is playing. The accelerometer in the Arduino determines the angle at which the user is holding the device. The breath sensor determines whether the user is providing enough airflow to produce the note and indicates note onset and offset time. The Arduino mounted at the end of the controller processes all of these sensor readings and constructs a packet to be sent to the RPi via Bluetooth.

Since the design report, the design of the physical controller has only been modified slightly. We decided to use the Inertial Measurement Unit (IMU) mounted on the Arduino Nano 33 BLE to limit the number of wires needed, and we decided to use an accelerometer reading instead of a gyroscope reading. We found that the accelerometer was more appropriate in determining angle changes from the horizontal, as it measures acceleration instead of velocity. We also decided to mount the Arduino and battery outside the controller on a breadboard. This ensures that the Arduino is always flat with respect to the controller (which makes the angle reading more accurate), it ensures that we can easily change wires if any of our connections break, and it allows us to easily upload code changes to the Arduino.

B. Communication

Using BLE, the Arduino Nano sends packets to the RPi using characteristics for each type of data. This packet includes the buttons pressed, angle difference from the horizontal, and breath sensor reading. Once the RPi receives the data, it does some data processing, plays the corresponding note through Sonic Pi, and writes to a text file for the web application.

No changes have been made to our communication design since the design report.

C. User Interface

The web application reads the sensor data from this text file and displays feedback depending on the mode the user chooses to interact with. This sensor data is processed in the three modes on the website: Learn, Test, and Free Play. In the Learn mode, the web app provides the user with information about various notes and scales and teaches the user how to play them. In the Test mode, the user is prompted to play scales without assistance. Lastly, in the Free Play mode, the user can simply just play various notes and see what note they are playing.

The web application also includes a Simple mode, where the user can choose to opt-out of breath control feedback for any of the three modes. This allows users to test out our product without placing their mouths on our device, which is essential for our public demo.

The only main change to the web application since the design report was the addition of the Simple mode.

IV. Design Requirements

A. Accuracy

Our use-case requires that the feedback the user receives from the project is 90% accurate. This can be broken down into the accuracy of our sensors and the accuracy of our wireless communication.

Our data must be sent wirelessly via Bluetooth with 95% accuracy. This is to compensate for any inaccuracies in our sensor readings. We can calculate the accuracy we have in our Bluetooth communication with the following equation:

$$accuracy = \frac{total\ packets\ sent - dropped\ packets}{total\ packets\ sent} \quad (1)$$

We need this accuracy to hold from a reasonable distance between the controller and the RPi. To be comfortable for the user, we need this accuracy in wireless communication to hold within a distance of five feet between the controller and the RPi.

To ensure that the device senses note fingerings with a 90% accuracy, our push buttons should be 99% accurate. We can calculate the accuracy of our push buttons with the following equation:

$$accuracy = \frac{tests\ where\ push\ button\ presses\ were\ detected}{total\ tests} \quad (2)$$

This should not be a difficult metric to achieve, as push buttons are mechanical devices that are rarely inaccurate. Combined with a 95% accuracy of wireless communication, our note fingering detection should hit our 90% accuracy requirement.

To ensure that our project senses correct playing posture with 90% accuracy, our gyroscope should sense a change in 1° with respect to the horizontal. This corresponds to a 95% accuracy, which combined with the accuracy of the wireless communication, should yield an overall accuracy of 90%.

Our breath control sensor should also determine the octave of the note with 95% accuracy. The flute can produce notes from three octaves, but our project only supports notes from two out of those three octaves. This is because the very low and very high notes the flute can play are very difficult, and a beginner would be unlikely to play them on a real flute. Therefore, our breath sensor only needs to differentiate between two breath speeds for each note. Combined with the 95% accuracy of our wireless communication, our breath control feedback should be 90% accurate.

B. Speed

The time between input change into our flute controller and receiving feedback from that change should be no longer than 500 milliseconds. This means that our feedback will only feel significantly delayed if the user is playing at a speed faster than 120 beats per minute, which we believe is the fastest reasonable pace for a beginner to play at.

The longest contributor to our overall latency is the latency of the wireless communication between our Arduino and RPi. We wish to have a latency of Bluetooth communication of no longer than 400 milliseconds. This leaves the remaining 100 milliseconds for processing data on the Arduino, processing data on the RPi, and sending updates to the web application. To split up these 100 milliseconds, we should have the latency of our Arduino and RPi signal processing be less than ten milliseconds each, and the latency of updates to the web application from the RPi (reading from a text file) should be less than 80 milliseconds. Combining all these latencies yields an overall latency of 500 milliseconds from a user input change to a feedback display.

C. User Experience

Our project should be easy to use and feel similar to a real flute. We break down our user experience requirements into dimensions and battery life.

The length, width, and weight of the flute controller must be close to 26”, 1”, and 1.13 lbs. respectively. More specifically, the length of our controller should be between 25” and 27”, the width of our controller should be between 0.7” and 1.3”, and the weight of our controller should be between 1.0 lbs. and 1.2 lbs. This is in order to best match the feel of a real flute. For the same reason, the buttons on the controller must feel similar to the buttons on a real flute. Our tactile buttons should be within 10% of the diameter of the buttons on a real flute. Additionally, the position of the mouthpiece must be comparable to the actual instrument. The position of the mouthpiece should land within 5% of the position of a real flute. These constraints for the controller are crucial in order to aid the user in transferring their learned skills to an actual flute and increase the efficacy of our device.

The battery life of the flute controller should last at least three hours. We chose this timeframe as the upper bound of a continuous playing session for a beginner. Therefore, the user does not have to replace the battery in the middle of a practice session.

No changes were made to our design requirements since the design report.

V. Design Trade Studies

Throughout the design phase of our project, we had to make many design tradeoff decisions to best fit our use-case. These design decisions can be divided into three main categories: the choice of our breath detection sensor, the nature of our communication between our physical controller and web application, and the nature of our communication between the RPi and our web application.

A. Breath Detection

Beginners learning to play the flute often feel that learning the correct breath control is the most difficult aspect. As a result, it is important for our project that our breath control sensor is accurate for learning breath control on a real flute. Throughout our ideation process, we determined three important requirements for our breath control sensor; accuracy, ability to fit inside the controller, and ease of connection to the Arduino. We examined these three requirements for each breath control sensor we considered.

1) Microphone

The first sensor we considered was a microphone mounted inside the controller. To use this sensor, we would need to convert the microphone reading to a breath speed measurement. This would involve filtering out noise and mapping the volume measurement to the blow speed. This approach would require signal processing, which no one in our group has much experience in. We would also need to distinguish between correct and incorrect blowing forms. We would need to ignore readings where the user is talking or shouting, as well as

readings when the user is blowing across or near the microphone without blowing down into the device.

Due to these complications, we predicted that the microphone would be the least accurate sensor for breath control. To determine whether the user was blowing instead of talking or shouting, our hardware would have to analyze the microphone signal in the frequency domain and determine if the sound spectrum matches that typically formed by a user blowing instead of talking. Since blowing would look similar to noise in the frequency domain, we would be looking for an equal band of high amplitude at many frequencies. However, we might not have a way to determine whether the user was blowing downward into the device or simply blowing nearby. If we used the volume reading from the microphone to determine the distance to the controller, we might inaccurately ignore breath control readings where the user is blowing softly into the mouthpiece.

Our second consideration for this sensor is the ability to fit inside our controller. We plan on using a PVC pipe with a 1" internal diameter to contain all our sensors. Therefore, the width and height of the sensor we use should be less than 1". Table 1 gives a breakdown of the dimensions of the sensors we considered using for breath control. Both the width and height of the microphone we considered using are less than an inch, so we would be able to comfortably fit this sensor inside our controller.

TABLE 1: BREATH SENSOR DIMENSIONS, SOURCE [3], [4]

Sensor	Dimensions (in)		
	Width	***Height***	***Length***
Microphone	0.62	0.18	0.95
Physical Fan	1.06	1.06	1
LED + Photodiode	0.38	0.23	0.23

Our final consideration is the ease that the sensor connects to the Arduino, which is the microcontroller we are using to process our sensor data and send it to the RPi. The connection between the microphone and Arduino is simple, as the microphone only has pins for power, ground, and analog out. Therefore, the microphone would only take up one pin on the Arduino in addition to powering the sensor.

2) Physical Fan

The second sensor we considered was a physical fan the user blew on. This fan would be mounted on top of the mouthpiece of the controller and would rotate when the user blew downwards on it. To sense the airspeed, the Arduino would connect to an IR diode and receiver mounted on each side of the fan. Blowing on the fan would break the beam between the IR diode and receiver, and we could convert the rate at which the beam is interrupted to the speed the user is blowing.

This sensor would have better accuracy than the microphone, as no signal processing is necessary to convert the signal to a speed reading. The user would not be able to move the fan when talking or shouting, but they might be able to move the fan when blowing at an incorrect angle (such as across the mouthpiece instead of down). To determine the speed the user is blowing, we would be able to use a more accurate reading from the interruption of the IR beam, instead of using the volume reading from the microphone.

The difficulty with this sensor is the size. If we created a paper windmill out of a small piece of paper (0.75" x 0.75"), the pinwheel would be 1.06" in diameter, which is larger than what would fit on the inside of the PVC pipe. We could shrink the paper fan more, but it would be hard to work with a piece of paper smaller than 0.75" x 0.75".

Similar to the microphone, the IR sensor would only require an analog input pin on the Arduino, as well as power and ground. The circuit required would be a bit more complicated than the microphone, because both the IR LED and the IR sensor need to be connected to resistors to ensure the correct current is being drawn.

3) LED and Photodiode

The final sensor we considered, and the sensor we plan to use in our project, combines the small, portable size of the microphone and the accuracy of a physical fan. This sensor consists of an LED and a photodiode mounted across from each other on different sides of the mouthpiece. A latex barrier is mounted overtop of the LED. When the user blows downward, the latex barrier blocks the beam of light between the LED and the photodiode, allowing the controller to sense the airspeed of the blow.

This sensor would be of similar accuracy to the physical fan. The user would not be able to trigger the sensor while talking or shouting. The user also would not be able to blow incorrectly into the sensor to trigger it, as the latex barrier is mounted horizontally and would need airflow downwards to break the beam. We did a proof of concept with an LED and a photoresistor, and we found it to be able to differentiate between blowing softly into the mouthpiece and blowing harder.

The LED and photodiode required for this sensor are both quite small and would be able to fit inside a mouthpiece-sized hole inside the PVC pipe. Table 1 gives the dimensions of this sensor.

Finally, the connection to the Arduino would be simple. The LED requires a connection to digital output and ground, and the photodiode requires a connection to analog input and ground. This would only take an additional two pins off the Arduino.

B. Bluetooth vs. Wired vs. Wi-Fi Communication

There are many factors to consider when looking at how to facilitate the communication between the flute controller and the RPi. There were many options available to us, each with its own pros and cons, which we will discuss here. In order for the flute controller to feel the most natural and allow the most range of motion for the user, we initially only looked at wireless options.

Bluetooth and Wi-Fi communication were both feasible options for this project, as the Arduino platform is compatible with many external modules that enable these types of communication. Additionally, there are Arduino boards with built-in BLE and Wi-Fi as well.

1) BLE

When looking specifically at BLE, we see that it fulfills many of our requirements. One, it is very energy efficient, especially compared to regular Bluetooth. This is achieved via a combination of longer sleep times and shorter transmission

bursts (from seconds/minutes to milliseconds). This helps us achieve our goal of a 3-hour battery life on a single 9V battery. Since we are not constantly streaming large amounts of data (such as music), the data transmission rate for BLE –which maxes out around 1 Mbit/s—satisfies the design requirements, as we are only sending a couple of bytes twenty times or so a second. The BLE connection is also a direct wireless connection to the RPi.

For cons, there is always a chance of dropped packets or lost data in wireless communication, which could be better managed by recommending the user stay within five feet of the RPi for the smoothest experience.

2) Wi-Fi

We also considered using a Wi-Fi connection and ended up not choosing it as our preferred wireless option. For one, this would require the user to connect the flute controller to their local Wi-Fi connection and depend on that network to transmit data to the RPi. We felt that this was not necessary especially since we are currently planning on hosting the application locally on the RPi. Additionally, more power and data would have to be used/sent compared to BLE, which would send 12 bytes at a time.

3) Wired Connection

We are also considering a wired connection between the flute controller and the RPi as our mitigation for not getting either of the wireless connections working, and this has its own pros and cons to consider. The biggest downside of this option is the potential to affect the use-case for the feel of the controller, as the user's motion would potentially be greatly inhibited by the presence of a cable attached to the end of the flute controller. However, this could be slightly improved by using a longer cable that is a couple of feet long.

In terms of the pros, a wired connection would be the most stable and fastest communication option since it would be the most direct connection to the RPi. This would ensure that our accuracy and latency goals were met in our mitigation case.

After looking at these three options, we decided to go with BLE communication between the flute controller and RPi for the energy benefits and direct connection, with the wired being our backup for reliability.

C. Communication between RPi and Web Application

Sensor data from the controller is first communicated to the RPi and then the web application. This is an essential communication line which means the design choices made are important. One such choice was between hosting on AWS versus hosting locally.

AWS would provide a platform for user interaction, as well as more security and scalability. We ultimately decided not to use AWS and host locally instead. User interaction would add a social media feel to our web application. Registered users would be able to communicate with each other and see others' learning progress. For a first implementation website, we did not feel that user interaction was needed. It is something that we can consider for the future. Similar to user interaction, scalability is not necessary for the initial implementation and would be a valid reason to switch when we add more instruments. AWS's best benefit is security for data-driven projects. While our project is data-driven, we are able to enforce the safety of our users' learning data, private logins, and progress with Django's built-in protections.

Overall, the main reason for choosing hosting locally over AWS is the latency. Our latency requirement is 500 milliseconds, so the user receives audio and visual feedback in a reasonable amount of time. Hosting on AWS would add an additional latency of 500 milliseconds varying by region. This along with the latency from the Arduino to RPi communication would not be efficient for our system.

VI. System Implementation

Our system implementation, briefly described in section III, is broken down into three sections: physical controller, communication, and user interface. In this section, we describe these subsystems in greater detail.

A. Physical Controller

Our physical controller consists of a collection of sensors encased in or at the end of a PVC pipe.

To sense the note the user is trying to play, the controller has nine push buttons mounted on the device. These nine buttons are arranged in the same locations as the buttons and keys on a real flute. The Arduino mounted at the end of the controller reads the button values and combines them into a 9-bit array to eventually send to the RPi.

To determine whether the user is holding the flute controller at the correct angle, we originally planned for the controller to have a gyroscope module mounted inside. However, when we decided on using the Arduino Nano 33 BLE as our final microcontroller, we decided to utilize the IMU on the board, so we were not wasting pins. The IMU contains a gyroscope, accelerometer, and magnetometer, and we decided to use the accelerometer instead of the gyroscope. The gyroscope gives angular velocity readings while the accelerometer gives angular acceleration readings, meaning the accelerometer is more appropriate to determine how the device is oriented in space, while the gyroscope is more appropriate for determining the direction of force on the board. The accelerometer reading is what the physical controller uses to determine how many degrees the user is holding the controller from the correct position (parallel to the ground) [8].

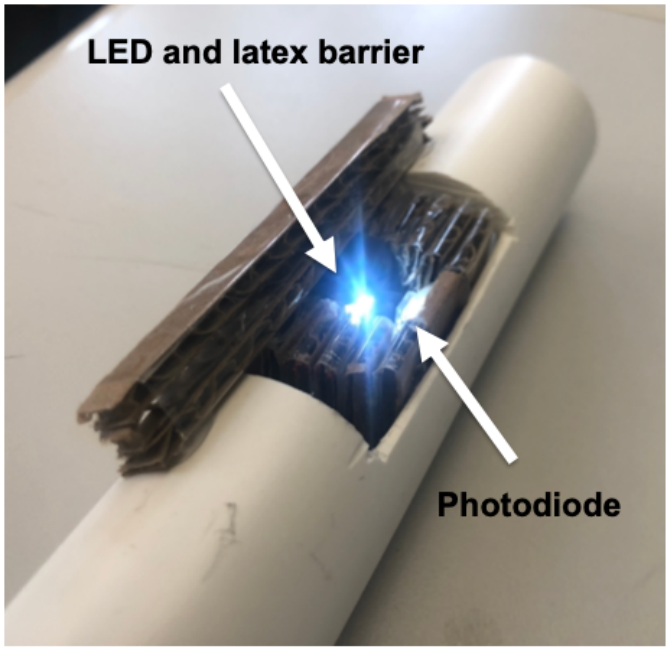


Fig. 3: Breath control sensor

To determine whether the user is using the correct embouchure to play the given note and is blowing with sufficient speed to produce the note, we have a breath sensor also mounted inside the controller. Our breath sensor consists of an LED and a photodiode mounted across the mouthpiece from each other, as well as a latex barrier mounted on top of the LED (Fig. 3). When the user blows downward into the mouthpiece, the latex barrier curls down and obstructs some of the light entering the photodiode. The analog output from the photodiode is sent to the Arduino, where the Arduino determines the airspeed that the user is blowing with.

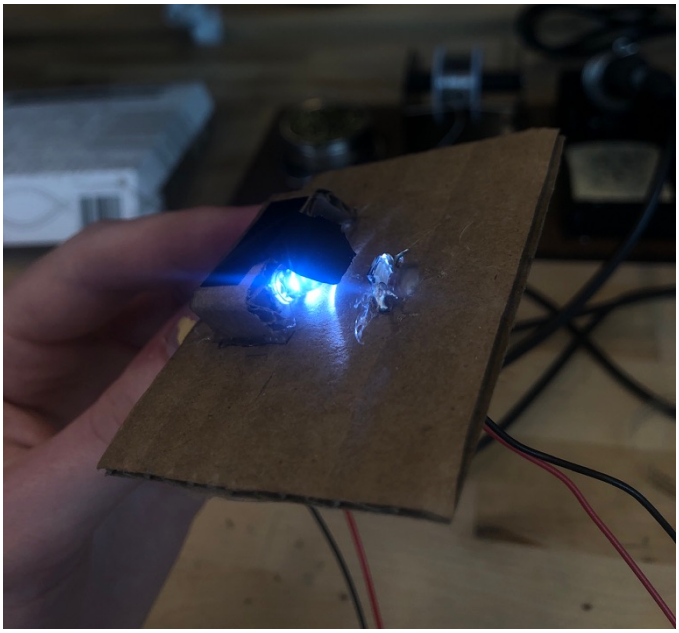


Fig. 4: First breath control prototype

Throughout the development process for the breath sensor, we had two major prototypes before landing on our final design. The first prototype, as shown in Fig. 4, consisted of the same LED, photodiode, and latex barrier, but was mounted quite differently. Both the LED and photodiode were glued on a flat piece of cardboard across from each other. A cardboard ceiling is placed on top of the LED for the latex to be taped onto.

This prototype was a good start but had a few issues. First, the dimensions of this sensor made it impossible to mount inside the controller because the flat piece of cardboard was larger than the 1" inner diameter of the PVC pipe. Secondly, the components of the sensor were not very well secured. The LED and photodiode moved around a lot, making the reading from the photodiode very inconsistent.

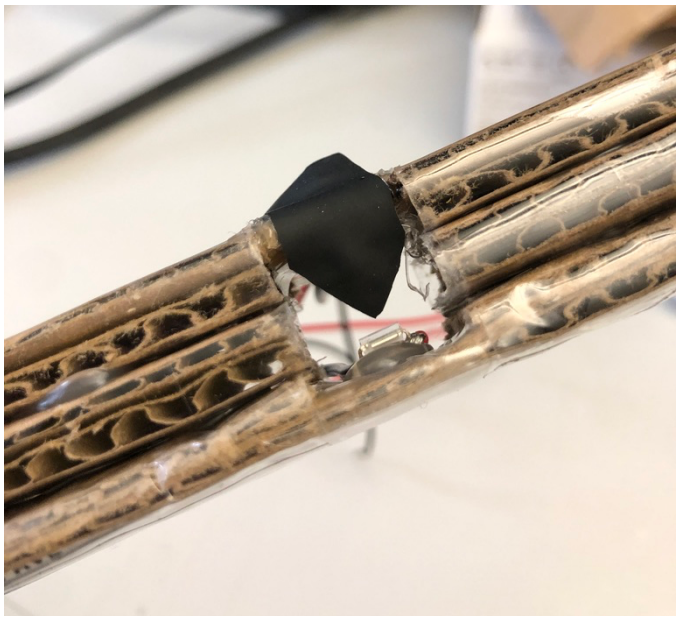


Fig. 5: Second breath control prototype

We improved on both of these issues in the second prototype of the breath sensor (Fig. 5). In this sensor, the LED and photodiode were attached flat to pieces of cardboard, with more pieces of cardboard in between to separate the two components. The latex barrier was mounted on the back of the cardboard behind the LED, resting over the top of it.

While this sensor was an improvement on the first prototype, it still had a few issues. The main problem was too much moisture was contacting the latex barrier and causing it to curl permanently. This meant that all of our breath sensor readings were too low, and we had to continuously replace the latex, which took a long time. We were able to solve this issue in our final breath sensor by using a thicker piece of latex and putting more space between the user's mouth and the breath sensor with some cardboard pieces on the PVC.

Finally, the Arduino constructs a packet to send to the RPi via Bluetooth. This packet consists of the 9-bit array containing the buttons currently depressed, the angle with respect to the horizontal that the user is holding the controller, and the airspeed with which the user is blowing.

B. Communication

Once the Arduino Nano collects the sensor data representing the flute fingerings, breath control, and position it broadcasts this data as a peripheral device in the BLE scheme, with the RPi acting as the central device. In BLE, members act as either of

these types of devices to either generate and post data using services and characteristics or to read/write to said data [11]. That is, the central devices interact with the peripheral devices that generate the data that they want. A diagram of central devices and peripheral devices showing their relationship graphically is located below (Fig. 6).

On the Arduino Nano, we defined a new service associated with the flute controller, and three characteristics for each type of data that we wanted to send (fingerings, breath control, orientation). All three of the characteristics for the fingerings, breath control, and orientation were represented as integers.

Once the Arduino updates a characteristic in the flute controller service, the RPi is able to quickly read the updated value when it changes (in a couple of milliseconds). Additionally, the baud rate was set to 9600 bits/s for the final product, but could easily be scaled up to 115200 bits/s.

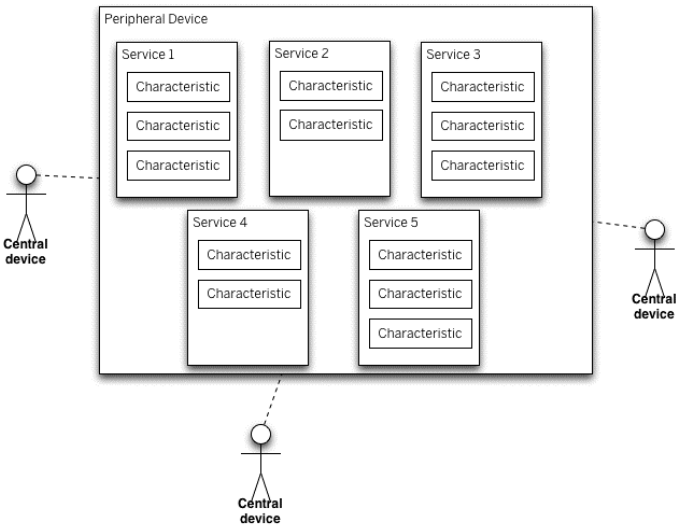


Fig 6: BLE Diagram, Source: [5]

Once the RPi reads data from one of the three characteristics, it writes the updated values of the sensor data to a text file for the purpose of communicating with the application. The RPi also determines which note is being played based on the fingerings and breath control amplitude. A note mapping implemented via a dictionary that is prepopulated with the fingering combinations is used to quickly translate fingerings to note values. If the fingering is invalid, no note is be played. After determining the note (in the case that it is correct), it then sends a signal via OSC (Open Sound Control) to Sonic Pi denoting the note value, whether a new note is being played, and a "stop" value denoting if any sound should be played at all. Once Sonic Pi receives the signal, it plays the corresponding flute sample [10] or stops playing the current sample, depending on the message sent. This is possible through the live_loop functionality in Sonic Pi that allows a loop to play that can dynamically run and react to signals and function calls. This direct connection between the Bluetooth program and Sonic Pi allowed the user to get auditory feedback as quickly as possible after receiving the sensor data.

The communication of the project was worked on incrementally with each portion being figured out as parallelly as possible— specifically, figuring out how to send sensor data via Bluetooth to an RPi and how to play flute samples automatically in response to a program. After being recommended Sonic Pi by a fellow student, it was relatively straightforward to use live_loops and OSC to play notes from a separate python script. The more difficult process was establishing a Bluetooth connection between an Arduino and the Raspberry Pi. We tried out many different Bluetooth modules such as the HC-10 (BLE) and HC-5, but none of them would connect to the RPi. After doing some research, we discovered the Arduino Nano BLE has built-in BLE capabilities. This chip allowed us to connect with the RPi with the help of another library on the RPi. We also utilized Bluepy on the RPi to utilize its BLE capabilities. This library allows easier control over the BLE ports on devices, and hence we were able to use it to connect specifically with the Arduino's MAC address in a program. Bluepy also contained defined classes that allowed us to read and write to specific characteristics and services of devices it was connected to, so we were able to send data to the RPi using the newly defined flute controller services and characteristics associated with the sensor data.

C. User Interface

The web application is the main view of the user interface. We use Django [13], a high-level Python web framework. This framework consists of several languages including HTML, CSS, and JavaScript. The Model-View-Controller system in Django allows the construction of four different modes for our system: "Practice", "Learn", "Test", and "Resource". The web application is styled using Bootstrap [12]. Bootstrap added functionality for the layout and a more appealing UI.

The Learn mode is the core of our web application as referenced in Fig. 7. This mode is similar to a private flute lesson. The user is not able to start if the controller is not enabled. On this page, the user first chooses one of the eight major scales plus a chromatic scale to play. Each scale goes through the notes with displayed flute fingerings. The UI consists of visuals of the flute fingering with the note on the staff, a visual of the fingering for the current note, and a description of the fingering. When the data from the user is received, this mode provides user feedback. These are displayed as red keys for incorrect hand positions and green keys for the correct hand positions. Additionally, feedback about the overall flute position such as the angle and breath control are shown. This page has the most computations and is done by reading a text file with sensor information from the physical controller. Specifically, the Controller in Django's MVC system has the functions for processing the sensors: the accelerometer, buttons, and breath sensor. The Controller communicates with the JavaScript component of Django to update the page every 200ms. When the user completes a note, a short popup appears giving the user feedback that they are correct. This continues until the user reaches the starting note again where the scale is marked as complete, and the user is redirected to the select scale menu.

The Test mode is where the user has a chance to show

mastery over the nine scales. The physical controller is required in this mode. The user first chooses the scale to get tested on. The UI is very similar to the Learn mode except the fingerings for the notes are not shown. Passing the scale means finishing all the notes. In the end, they receive an accuracy score. If the user has the correct breath control but does not have the correct fingering, the accuracy score decreases. The priority in the Test mode is breath control, correct position, and finally, overall flute position. On the processing side, the user inputs from the constantly updating text file are compared with the correct fingering in the scale. The test results are saved inside the user's database.

The Resource mode is the simplest mode. It does not require the flute controller to be connected. On this page, there are options to view and listen to the seven major scales that are available to learn. A static flute fingering chart of all the notes is also displayed. And finally, there are links to resources for additional education.

The purpose of the Practice mode is to give the user a space to practice what they learned. It is similar to the experience of playing the flute for fun or practicing a musical piece. The UI for this page is more relaxed, and there is user feedback as well as a fingering chart to practice. It ensures that the controller is active before starting.

All of the modes, with the exception of the Resource mode, have an option called "Simple". With Simple enabled, the web application disregards the breath control sensor. The web application passes information to the RPi to produce the sound. This is done by the Controller writing to a text file that the RPi reads. In the Practice mode, the note the user is playing is only written if they have the correct fingering for a note. Since there are similar fingerings for multiple notes, only one octave is available. In the Test and Learn mode, the note that is written to the text file is the current note in the scale.

Every user of the web app has a profile page. The main information shown is the progress for each scale in Learn mode and the completion for each scale in Test mode. The user also has the option to reset all their scale data.

The current web application is run locally. A database is needed to store user information, like usernames, passwords, and progress. There are many databases Django supports, and we started with MySQL due to familiarity. The final web application uses PostgreSQL [14] since it is much faster.

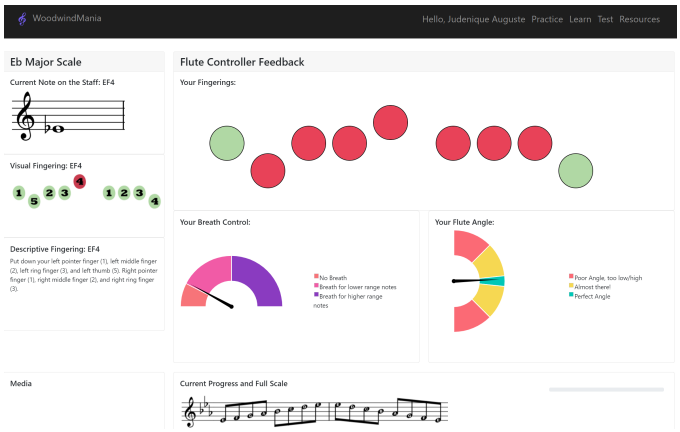


Fig. 7: Learn Mode in action

VII. Test, Verification, and Validation

We separated our use-case requirements into three categories: accuracy, speed, and user experience. Below we break down our testing procedures and results for each requirement.

A. Accuracy

We aimed to have 90% accuracy in the feedback from our project. This accuracy relates to the feedback provided for note fingerings, instrument posture, and breath control.

To test the accuracy of our note feedback, we provided both correct and incorrect note fingerings when prompted. For each of the 14 notes in a scale, we tested both an incorrect note fingering and a correct one and verified the feedback given from the web application. We then averaged our accuracy over 28 trials.

Similarly, to test the accuracy of our posture feedback, we held the controller at the correct angle (parallel to the ground) and a series of incorrect angles. We then tested that the web application correctly determined if the user was playing in the correct posture and averaged that result over ten trials.

Finally, we tested the accuracy of our breath control sensor. First, we tested that our sensor does not interpret talking or blowing with incorrect form (such as blowing across the mouthpiece instead of blowing downwards) as an attempt to play a note. Second, we tested that the breath sensor sensed the octave the user was playing correctly. We tested this by doing ten trials of blowing softly to produce the lower octave note and ten trials of blowing hard to produce the higher octave note. We calibrated the speed necessary to blow soft/hard by using a real flute. We then determined if the web app played the correct octave, averaging over 20 trials.

In our design report, we also included a design requirement of 95% accuracy for Bluetooth communication. This is because we anticipated that we would occasionally drop a packet, and that would affect our overall feedback accuracy. As a result, we aimed for higher than 90% accuracy of our sensors, so that when averaged with our Bluetooth accuracy we would hit our overall requirement of 90%. However, we found in our testing that we rarely ever dropped a Bluetooth packet. As a result, we were able to lower the accuracy we could tolerate from our sensors to still hit our overall 90% feedback accuracy.

TABLE 2: TEST RESULTS FOR ACCURACY REQUIREMENTS, USE-CASE REQUIREMENTS ARE BOLDED

Requirement	Metric	Result
Accurate fingering feedback	**>= 90%**	**100%**
Bluetooth accuracy	>= 95%	100%
Button accuracy	>= 99%	100%
Accurate orientation feedback	**>= 90%**	**90%**
Accelerometer accuracy	>= 95%	90%
Accurate breath control feedback	**>= 90%**	**90%**
Breath sensor accuracy	>= 95%	90%

Table 2 above shows a summary of our test results for our accuracy requirement. For note fingering feedback, we had an overall accuracy of 100%. As mentioned previously, we had a Bluetooth communication accuracy of 100%, in addition to a button accuracy of 100%. This was expected because push buttons are mechanical sensors that are rarely inaccurate.

For orientation feedback, we had an overall accuracy of 90%. Specifically, when we tested our orientation feedback, we only found one trial out of ten to give inaccurate feedback. While we did not hit our desired accelerometer accuracy of 95%, because Bluetooth accuracy was so high, we still hit our use-case requirement.

Similarly, for breath control, we hit our overall accuracy requirement of 90%. We found our breath sensor to only be 90% accurate instead of 95%, but we are still satisfied because we hit our use-case requirement. Specifically, we completed 20 trials of testing for the breath sensor, and we only found two trials where the breath control feedback was wrong. Additionally, we found our breath sensor to require the correct technique to get readings. The user must blow instead of talking or shouting, and specifically, the user must blow downward into the controller to trigger the sensor.

B. Speed

We aimed to have an average latency of less than 500ms between the user producing a note and our project providing feedback. In our design requirements, we split this up into the two largest contributors: the latency from Bluetooth communication and the latency from the communication between the RPi and the web app. For Bluetooth latency, we aimed to have an average of 400ms and for web app latency we aimed to have an average of 100ms.

We tested the Bluetooth latency by writing a python program to ping the Arduino and wait for a response back. The code records the start time and the end time and then subtracts the two times to determine the total time taken for a round trip via Bluetooth. The Arduino code simply waited for a message and immediately sent a message back. The Bluetooth latency is then calculated as:

$$latency = \frac{end\ time - start\ time}{2} \quad (3)$$

We tested the web app latency by writing a python program that continuously wrote the current time to a text file. Then, in the web app, we wrote some JavaScript code to read the text file, subtract the given time from the current time, and print the latency in milliseconds.

TABLE 3: TEST RESULTS FOR SPEED REQUIREMENTS, USE-CASE REQUIREMENTS ARE BOLDED

Requirement	Metric	Result
Total latency	**<= 500ms**	**369ms**
Bluetooth latency	<= 400ms	96.95ms
Web app latency	<= 100ms	269ms

Table 3 above shows a summary of our test results for our accuracy requirement. For our Bluetooth latency, we found that we were way under our 400ms bound, and our latency was around 97ms on average. For our web app latency, however, we were over our predicted latency bound, with an average of 269ms. While this was more than we aimed for, our overall latency was still within our 500ms bound, with an average latency of 369ms.

TABLE 4: FIVE TRIALS FOR LATENCY TESTS

Requirement	1	2	3	4	5
Bluetooth latency	97ms	97.05ms	96.9ms	96.9ms	97.1ms
Web app latency	18ms	234ms	127ms	20ms	317ms

Table 4 above shows five trials of both our Bluetooth latency tests and our web app latency tests. As you can see from these trials, the Bluetooth latency is very consistent and close to the average latency. The web app latency, however, is quite inconsistent and fluctuates quite far above and below the average.

One tradeoff we identified when testing our latency requirement was the tradeoff between audio latency and having a wireless controller. The average reasonable audio latency is considered to be around 20ms [6], and our audio latency is currently around 100ms due to our Bluetooth communication. Our Bluetooth latency is actually around the average Bluetooth latency (between 100ms and 300ms) [7], but this is higher than desired for general audio latency. We decided that for a beginner learning to play the flute, it was more important that the controller be wireless than to have a small delay between user input and audio feedback.

C. User Experience

Because users would be using our project to replace a real flute, it was important that we extensively tested our user experience. This came in two major areas: controller dimensions and user satisfaction. We tested our controller dimensions by measuring the length, width, and weight of our physical controller. We aimed for our controller to be between

25” and 27” in length, between 0.7” and 1.3” in width, and 1.0 and 1.2lbs in weight. We tested the user satisfaction of our device in the form of a survey. We surveyed 10 beginners (and a few advanced flute players) on the comfort of our controller, comparison to a real flute, and overall satisfaction with the device. We aimed to have an overall user satisfaction rate of 80%.

TABLE 5: TEST RESULTS FOR USER EXPERIENCE REQUIREMENTS, USE-CASE REQUIREMENTS ARE BOLDED

Requirement	Metric	Result
Length	**25"-27"**	**29"**
Width	**0.7"-1.3"**	**1.375"**
Weight	**1.0-1.2lbs**	**1.13lbs**
Portability	**Controller is wireless**	**Controller is wireless**
User satisfaction	**>= 80%**	**92%**
Cost	**<= $500**	**$249.49**
Size of buttons	13.5mm-16.5mm	13.5mm
Place of breath sensor	1.7"-4.3"	2.75"
Battery life	>= 3hrs	8hrs
Bluetooth range	>= 5ft	30ft

Table 5 above shows a summary of our test results for our user experience requirements. For our dimension measurements (length, width, and weight of the controller), the only requirement we hit accurately was the weight requirement. We made some design tradeoff decisions in relation to both of these requirements, which we will discuss shortly.

However, all our other use-case and design requirements fell within the range we were aiming for. Firstly, we created a wireless controller that is significantly less expensive than purchasing a beginner flute and a month of lessons (around $500). Secondly, our buttons and breath sensor are positioned correctly in relation to where they fall on a real flute, and our battery life and Bluetooth range are comfortable for practicing. Finally, our overall user satisfaction rate was 92%.

When we did our user satisfaction survey, we were able to gain a bit more insight than just overall user satisfaction. The main complaint we received was that the buttons were a bit difficult to press, and therefore somewhat uncomfortable to hold for long periods of time. However, we also received some helpful positive feedback. Most users found breath control to be comfortable to use and found in general that the feel of the device was pretty similar to a real flute.

We made quite a few design tradeoff decisions in relation to the user experience requirements. Firstly, our physical controller is currently 29” in length because our board is mounted at the end on a breadboard. This ensures that our Arduino is always flat with respect to the device for accelerometer readings and allows us to replace broken wires and upload code easily. We determined that this increase in length did not affect the user much, as the buttons were still comparable to their spots on the flute and the user’s hands did not contact the breadboard or battery at the end.

Secondly, the width of our controller is currently 1 3/8”. The outer diameter of a real flute is 1”, so our device is slightly wider than a real flute. While this is a bit of a difference the user might notice, most users we surveyed did not appear bothered by the slight increase in width. We chose a PVC pipe with a slightly larger outer diameter than 1” to ensure we could fit all our electronics inside. Our electronics currently fit quite snugly in the device, so if we had chosen a smaller PVC pipe, we likely would have had a lot of wires sticking out of the controller. We felt that having the controller wireless was more important than this small difference in width.

Finally, we wanted to make sure our buttons were of similar size to the buttons on a real flute. Because the buttons on the flute are quite large for push buttons, there were not a lot of options we could order. In the end, the best buttons we could find were a bit hard to press sometimes (as evidenced by the feedback in our user survey), but we felt that it affected the user’s experience more to have similarly sized buttons to that on the flute.

VIII. PROJECT MANAGEMENT

A. *Schedule*

Our schedule remained mostly the same between the design report and the final demo. We still split our project into three main sections, each taken on by a specific team member, and worked on them largely in parallel. In our design report, we had about two weeks of slack at the end of the project, and we ended up using all of that for testing and adjustments before the demo. We extended some of our beginning tasks for the communication section of the project because getting Bluetooth communication between the Arduino and the RPi ended up being more challenging than we anticipated. Similarly, we extended tasks to work on the breath control sensor because that also took longer than anticipated. Additionally, we removed and shortened integration tasks for the communication between the RPi and the web app because we ended up just communicating via a text file instead of by Wi-Fi. Finally, we added tasks to complete our demo mode for our final demo, because we did not anticipate needing that in the design report.

Refer to Appendix A for our Gantt chart.

B. *Team Member Responsibilities*

Angel Peprah was primarily responsible for the communication chain in the system. Her responsibilities included sending sensor information from the Arduino with BLE characteristics to the RPi, transferring that processed information from the RPi to the web application, and determining the audio of the note through Sonic Pi. Angel worked with Vivian to determine what note was played based on fingerings and breath input from the flute controller.

Judenique Auguste was primarily responsible for the construction of the web application and user interface. Her responsibilities included designing the layout of the website, the

scales, flute fingerings, the three modes, and the processing of user feedback. Judenique collaborated with Angel to send/process the sensor information in the communication chain between the RPi and the web application and adjusted data as needed.

Vivian Beaudoin was primarily responsible for the construction of the flute controller. Her responsibilities included crafting the PVC pipe to size, wiring and soldering the buttons, fitting all components inside the pipe, and combining the breath control components. As stated previously, Vivian worked with Angel for note determination.

Our team member responsibilities did not change after the design report.

C. Bill of Materials and Budget

Please see Appendix B for a full list of the equipment we used and their costs, with indications of items we discovered we no longer needed and items we realized we needed after the design report.

D. Risk Management

The largest risk we faced in this project was implementing wireless communication between the RPi and the Arduino. If this communication had too high of a latency, users could begin to feel frustrated using our project. If the communication was unstable, the accuracy of feedback would be greatly affected. It was also vital that our controller remained wireless to make it easier to use and similar to a real flute.

To combat this risk, we began researching and prototyping this area of the project very early, and we dedicated a group member's responsibility solely to this communication. We also had many backup plans to fall back on. We originally planned on attempting communication first with Bluetooth, then with Wi-Fi, and finally, with USB wired communication as the last resort. As getting Bluetooth to work initially seemed daunting, we reached out to our TA, since she had previously used BLE in her capstone project. With her guidance and recommendations on libraries to use, we felt confident that we would be able to get it working with BLE if regular Bluetooth was too difficult to get working. However, in the case that Bluetooth did not work, we had a deadline in our initial schedule where we would cease work on Bluetooth and move on to the next options so integration would still start on time. Thankfully, we were able to get Bluetooth working before this deadline and did not have to switch to our backup plans. We also were able to use our extra budget to buy many different Bluetooth modules to test out until we were successful.

Another risk we faced in this project was having inaccurate or inconsistent readings from our sensors. Our physical controller combines a lot of different analog sensors, and therefore we needed to do a lot of calibration and testing to find sensors that were accurate and reasonable for our use-case. The three sensors we have inside our flute controller are push buttons, an accelerometer, and a breath control sensor. Because a large use-case requirement for our project is the accuracy of our feedback, it was vital that our sensors were accurate.

Our mitigation plan for this risk was to again begin prototyping early and dedicate a group member to focus solely on this area. We tested a variety of different sensors and selected the most accurate and consistent sensor for our final product. As we expected, the push buttons and gyroscope were the easiest to get accurate, so we did not spend too much time on those sensors once we got working ones. Our hardest sensor was definitely the breath control sensor because we mainly created this sensor from scratch and breath control is an important part of playing the flute. To combat this, Vivian researched this sensor very early and made a proof of concept before the design report, so we felt confident that we would be able to reach a working product. Vivian also made two early prototypes before landing on the final sensor. These sensors were of various sizes and used various materials until eventually we found a consistent sensor. We researched and tested various different materials for the sensor, including different kinds of latex and acrylic or thin wood instead of cardboard. We also considered mounting the breath sensor outside the controller if we needed to make it larger or more stable.

The last risk we identified for this project was ensuring the feel of our controller was like the feel of a real flute. This is important to our use-case as our project is meant to be a replacement for purchasing a real instrument and receiving lessons. If the controller was too heavy or clunky to hold, users would feel uncomfortable practicing on it.

To manage this risk, we started building the physical controller as early as possible. We also calculated rough weights and dimensions for components before we purchased them, making sure we only got sensors that would match the feel of a real flute and fit inside our controller. Because we had a lot of money in our budget, we were able to purchase a lot of different components and test them individually for feel and size, instead of getting stuck on one item. We also always had a flute in the lab to use as a reference, which was vital for verifying that our controller felt similar to the real thing. We specifically chose the flute for our project because Judenique is a flute player and had an extra flute on hand to keep in the lab. Finally, we made a backup plan to switch to wired communication and take the heavy sensors out of the controller if the controller was unreasonably heavy and hard to hold.

IX. Ethical Issues

In terms of ethical issues, there are a few things to discuss. For one, the use of a web application opens opportunities for malicious parties to access the data associated with users on the site. Some examples of data that could possibly be stolen are emails, passwords, or even payment information if that was added to the web application. Another thing that malicious parties could determine is when users are practicing or using the app. This data could be used to track what the users are doing and help piece together their schedule for nefarious purposes. Malicious users could also hack the Bluetooth connection between the flute controller and RPi, sending unintended data to the web application or causing the user to think that the controller is not working. To mitigate this, we could use secure communication between the different parts of the project and make sure to encrypt all communications.

The device, as it currently is, could not be accessible to certain users due to some requirements in the web application.

People who are unable to hold the flute controller horizontal to the ground would not progress in the Learn or Test modes. This could be fixed by simply notifying the user that their posture is not flat, instead of requiring it to progress. Another way that the controller could not be accessible is the difficulty of pressing the buttons. Compared to a real flute, the controller's buttons are a lot harder to press. Additionally, the added size from the thickness of the PVC pipe also makes the device harder to use for people who have smaller hands. These issues could be fixed by making the controller smaller with custom PCBs to minimize the wiring and by making custom buttons that are easier to press down, making it easier to use.

In terms of the web application not being accessible, it currently uses colors to distinguish between buttons being pressed or not. This could be difficult for users who are color blind. Additionally, the only audio feedback provided in our project is when the user puts in a correct fingering. This could be difficult for users who mainly rely on audio feedback for day-to-day tasks that could still learn an instrument with the help of a more traditional teacher. These could all be fixed by tweaking the web application to be more accessible and give feedback in a larger variety of options.

X. Related Work

While researching sensors and requirements for our project, we discovered that there does not currently exist a woodwind controller that accurately mimics a flute. The current flute controller on the market [2] is modeled to be a generic woodwind instrument, so the breath control required is closer to a clarinet or saxophone than a flute. This controller supports output with headphones, a USB cable, or wirelessly to an optional device. This instrument retails for 800€ (~$888) and does not offer any feedback on note playing, posture, or breath control.

Our design is more specialized for flute playing, including a more accurate breath control sensor, and requires the user to hold the controller at the correct angle. We output sound from the RPi wirelessly, and our controller is significantly less expensive.

The closest project we have found to our design is a YouTube project [1]. In his design, he uses conductive paint instead of buttons and uses specific buttons to change the octave the flute is currently playing. Our breath control sensor is modeled off his breath sensor, but his flute controller is connected to his laptop via a USB cable and does not provide any feedback.

Our design uses tactile buttons and a wireless controller to resemble the feel of a real flute. We modeled our breath control sensor off this sensor, but we use the reading of the breath speed to test the user's octave instead of simply changing the note volume.

XI. Summary

Our goal was to create an effective woodwind learning tool for beginners to start learning the flute at a low cost. Overall, we feel that we were able to accomplish this when looking at the use-case specifications. Our goals for latency and accuracy were all met as determined through testing, and the flute controller itself was in the same range of the weight and size dimensions of a real flute. In terms of performance, we are currently limited by two things, latency and feel, which future groups could address.

For latency, we would like it to be lowered to allow for a smoother experience. The user could receive feedback quicker, put in a faster series of inputs, and practice at a faster speed. This could be accomplished with the use of a different device to host the web application or by hosting the web app on the cloud. We also found that our web app latency was pretty inconsistent and think that hosting the web app differently would also address this issue.

As stated in the user feedback, the buttons were hard to press, so this aspect of the controller also limits the experience. This is for two reasons. First, this contrasts with a real flute, so the pressing the buttons does not feel like pressing the valves, negatively affecting our goal for feel. Second, the buttons being hard to press negatively affects the user experience leading to quicker fatigue during practice, and harder use overall. This was an extremely important portion of the feedback that we feel could reasonably be addressed with more time, as we were limited in our button options since we focused more on the size similarity to an actual flute.

We learned the importance of leaving slack in our schedule. Components took more time to develop than we initially thought they would, which ate into the slack that we built into our timeline. Without the conscious addition of slack, we most likely would not have been able to reach all of our goals for the project.

Another thing we learned throughout this semester is the importance of being flexible and willing to try different things. When trying to figure out Bluetooth, we had to go through five different modules and boards before we found one that would work with our design. Additionally, the breath sensor went through many prototypes between establishing the proof of concept and the final product that performed reliably.

Overall, we feel this project was a success, and will be excited to see the work of any future groups that choose to continue this.

Glossary of Acronyms

AWS — Amazon Web Services
BLE — Bluetooth Low Energy
IMU — Inertial Measurement Unit
IR — Infrared
LED — Light Emitting Diode
RPi — Raspberry Pi
UI — User Interface

REFERENCES

[1] KontinuumLab. "KontrolFreak DIY cardboard MIDI flute. Worlds first flute mouthpiece emulator?" *YouTube*, Jun. 29, 2019 [Video file]. Available: https://www.youtube.com/watch?v=HkaP1IJqq98. [Accessed: Mar. 4, 2022].

[2] "Buy the Sylphyo – electronic musical instrument – Aodyo." [Online]. Available: https://www.aodyo.com/sylphyo-produit-157.html. [Accessed: Mar. 4, 2022].

[3] "Electret Microphone Amplifier - MAX4466 with Adjustable Gain." [Online]. Available: https://www.adafruit.com/product/1063?gclid=CjwKCAiAo4OQBhBBEiwA5KWu_2fry14Pa8Jg-PhUIAVv299KjdJ6InrgVb7P4s4mAHMuBUoFaqL4AxoCKR0QAvD_BwE. [Accessed: Mar. 4, 2022]

[4] "LED Size Chart: Types & Dimensions," 2022, [Online]. Available: https://evandesigns.com/pages/information-about-led-sizes. [Accessed: Mar. 4, 2022]

[5] SM. 'ble-bulletin-board-model', 2019 [Online Image] Arduino. Available: https://www.arduino.cc/en/Reference/ArduinoBLE?_gl=1*z9dbj6*_ga*MTQyOTAyOTg5Ni4xNjQ0MDIzODk5*_ga_NEXN8H46L5*MTY0NjI3MDU0Ni4zMC4xLjE2NDYyNzA1NDguMA. [Accessed: Mar 4, 2022]

[6] Sweetwater, "Better Latency Than Never: Latency Tips and Hints:" inSync, 06-Jan-2005. [Online]. Available: https://www.sweetwater.com/insync/better-latency-than-never-latency-tips-hints/. [Accessed: 05-May-2022]

[7] C. Editor, "How to Fix Sound Delay in Bluetooth Headphones," Headphonesty, 07-Oct-2021. [Online]. Available: https://www.headphonesty.com/2020/07/fix-sound-delay-bluetooth-headphones/. [Accessed: 05-May-2022]

[8] N. Alushi, "Accessing Accelerometer Data on Nano 33 BLE," Arduino Documentation | Arduino Documentation, 03-May-2022. [Online]. Available: https://docs.arduino.cc/tutorials/nano-33-ble/imu_accelerometer. [Accessed: 07-May-2022]

[9] A. Birkett, "What is Customer Satisfaction Score (CSAT)?," HubSpot Blog, 16-Jun-2021. [Online]. Available: https://blog.hubspot.com/service/customer-satisfaction-score#:~:text=While%20CSAT%20scores%20vary%20by,a%20negative%20or%20neutral%20one. [Accessed: 07-May-2022]

[10] Robert, "Adding a new sample based flute voice for sonic pi," rbnrpi,wordpress.com, 13-Oct-2014. [Online]. Available: https://rbnrpi.wordpress.com/project-list/adding-a-new-sample-based-flute-voice-for-sonic-pi/. [Accessed: 07-May-2022]

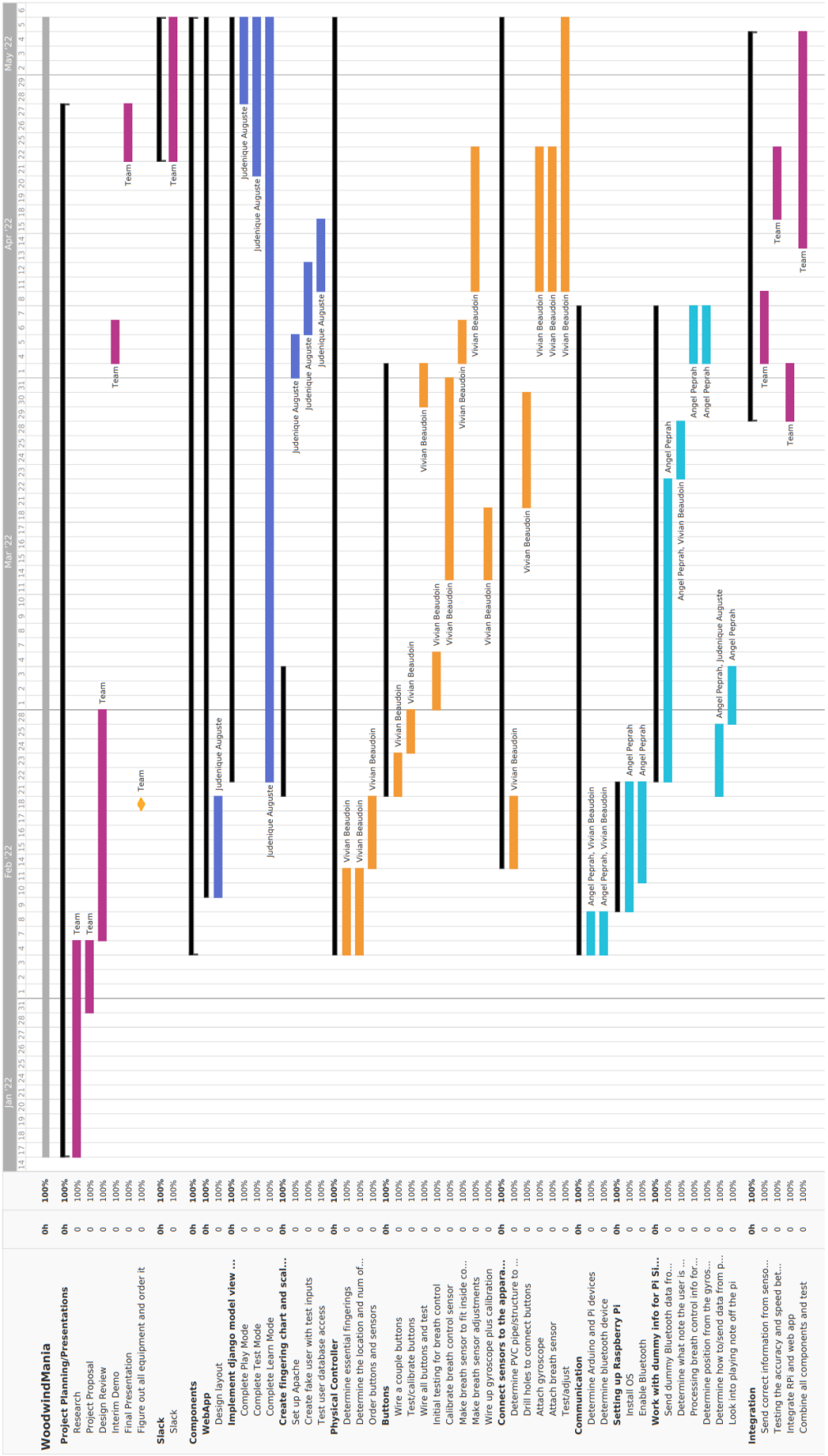
[11] "Arduinoble - blecharacteristic.written()," ArduinoBLE - bleCharacteristic.written() - Arduino Reference. [Online]. Available: https://www.arduino.cc/reference/en/libraries/arduinoble/blecharacteristic.written/. [Accessed: 07-May-2022]

[12] Mark Otto, Jacob Thornton. "Bootstrap." Bootstrap · The Most Popular HTML, CSS, and JS Library in the World. [Online] Available: https://getbootstrap.com/. [Accessed: Mar 4, 2022]

[13] "The Web Framework for Perfectionists with Deadlines | Django." Django Project [Online] Available: www.djangoproject.com. [Accessed: Feb 20, 2022]

[14] Group, PostgreSQL Global Development. PostgreSQL, 8 May 2022, [Online] Available: https://www.postgresql.org/. [Accessed: April 20, 2022]

Appendix A: Gantt Chart



Appendix B: Bill of Materials

Description	**Model #**	**Manufacturer**	**Quantity**	**Amt Used in Final Product**	**Notes**	**Cost @**	**Total**
Raspberry Pi 4	SC15184	Labist	1	1		$159	$159
Bluetooth 2.0 Module	HC-06	DSD Tech	1	0	Did not need	$8.49	$0
3-Axis Accelerometer Gyroscope Module	B01DK83ZYQ	HiLetGo	1	0	Did not need	$6.29	$0
16GB Micro SD	SDSQUNC-016G-GN6MA	SanDisk	1	1		$7.89	$7.89
Arduino Nano 33 BLE	ABX00034	Arduino	3	1	Ordered extras	$22.50	$22.50
Bluetooth Serial Pass-through Module	HC-05	DSD Tech	1	0	Did not need	$9.99	$0
Bluetooth 4.0 BLE Beacon	ML-HM-10	DSD Tech	1	0	Did not need	$10.99	$0
3ft PVC Pipe (1 in diameter)		Ventral	2	1	Ordered extras	$12.59	$12.59
16mm Momentary Pushbutton (set of 10)	B07SVTQ7B9	Twidec	2	1	Ordered extras	$7.99	$7.99
Silicon PIN Photodiode (set of 5)	BPW34	Comimark	1	1		$6.49	$6.49
9V Battery (8 pack)		Amazon Basics	1	1	Ordered post-design report	$11.39	$11.39
9V Battery clip connector (8 pack)		Meebok	1	1	Ordered post-design report	$6.99	$6.99
Low Voltage Solderless Wire Connectors (12 pack)	B017FT5G4I	Brightfour	1	0	Did not need	$10.99	$0
Wago (2 port, 3 port, 5 port) pack	B07NKSHVF6	ECTY	1	1	Ordered post-design report	$14.65	$14.65
							$249.49