RoboTune

By: Adam Davis, India Hutson, Tobi DeRuiter, and Ryan Ellison

Purpose:

One of the most important qualities to an ensemble is ensuring that each instrument is tuned to the correct pitch. Slight variations in pitch across several instruments can cause a drastic degradation in sound quality. Imagine 10 people signing the same note, but each of them are slightly off pitch. As you can probably guess, it wouldn't sound very nice. The RoboTuner aims to correct this issue by continuously tuning a person's instrument throughout their performance to ensure a better sound quality.

Traditionally, musicians have tuned by using clip-on tuners or by tuning to a given note. Both of these techniques have flaws. By tuning to a given note, musicians are limited by the precision of their ear. Even the most skilled musicians cannot perfectly match pitch, and therefore it is likely that this method often results in inaccurate tuning. Additionally, by tuning in this way, musicians only tune a singular note. While tuning that one note may draw the other notes closer to the correct pitch, it won't tune them all to a performer's desired level of accuracy. The only way for each note to be perfectly in tune is to adjust the tuning slide individually for each note. This effect is often more noticeable when using a clip-on tuner because they display the pitch quality of each note as it is being played. However, even with the clip-on tuner the only way of adjusting the tuning of an instrument while it is being played is by using your embuscher, or mouth muscles, and this method comes with human error.

The RoboTune device fixes all of these issues. It measures the pitch of each note while it is being played and autonomously makes adjustments to the tuning slide. By using a computer, the effects of human error are removed and the device allows each note to be tuned individually which improves the overall pitch quality of the instrument. RoboTune can also display the pitch quality live and show the pitch tendencies of the player to provide the player with information that they can use to improve their playing independent of the auto-tuner.

Functional Requirements

Tuning

1. As a user, I would like my brass instrument to tune itself before a performance.
2. As a user, I would like my brass instrument to tune itself between songs.
3. As a user, I would like my brass instrument to tune itself while playing, if needed.
4. As a user, I would like my brass instrument to continuously tune itself while playing.
5. As a user, I would like to know the note the tuner reads. (Desktop App)
6. As a user, I would like to be able to visualize the tuning analysis in a graphical format that shows me how far off pitch I am. (Desktop App)
7. As a user, I would like to have a mode that does not automatically correct the tuning, but will play a reference pitch for the note that I'm trying to play instead (if time allows.)

Hardware Control

8. As a user, I would like to be able to regulate the speed that the tuning slide is adjusted. (Desktop App)
9. As a user, I would like an auto-tuning device that can be compatible with as many brass instruments as possible via software and hardware configurations.
10. As a user, I would like the tuner to turn off after a desired period of inactivity.
11. As a user, I would like to define limits on the tuning slide's range for certain performances (if time allows.) (Desktop App)
12. As a user, I would like to save sets of hardware configurations for easier instrument swapping. (Desktop App)
13. As a user, I would like the tuner to enter sleep mode after the instrument has not been played for a desired period (if time allows.)

Damage Reduction

14. As a user, I would like to input the length of the tuning slide so that the tuner doesn't push it farther than it can go. (Desktop App)
15. As a user, I would like the tuner to stop moving if it encounters significant forces that oppose the movement of the tuning slide.
16. As a user, I would like the hardware to not apply any damage to the instrument.
17. As a user, I would like the tuner to be able to withstand damage.
18. As a user, I would like an indicator that there is too much resistance.
19. As a user, I would like the tuner to be water resistant (if time allows.)

Configuration

20. As a user, I would like a desktop application that I can use to configure the device to my specific needs.
21. I would like to be able to reinstall the software on my tuner via the desktop application if the software becomes corrupted (if time allows.)
22. As a user, I would like to be able to easily find the specifications of the hardware. (Desktop App)
23. As a user, I would like a way to export my saved settings. (Desktop App)
24. As a user, I would like a way to import saved settings. (Desktop App)
25. As a user, I would like to have multiple settings that allow me to configure the precision that the device will attempt to tune with. (Desktop App)
26. As a user, I would like to have a manual that explains how to use the device. (Desktop App)

Desktop App Displays

27. As a user, I would like to see my tuning tendencies for specific notes over a given playing period. (For example: B3 is 7 cents sharp on average when played.) (Desktop App)
28. As a user, I would like to be able to access the CAD designs of any custom built parts. (Desktop App)

Hardware

29. As a user, I would like the hardware to be removable.
30. As a user, I would like the hardware to be easy to attach to the instrument.
31. As a user, I would like the hardware to fit reasonably within the dimensions of the instrument.
32. As a user, I don't want a lot of hanging wires that could interfere with my performance.
33. As a user, I would like to be able to turn the auto tuner on and off easily.
34. As a user, I would like the motor to not stall during play.
35. As a user, I would like an indicator that battery life is low.

Non-functional Requirements

Speed

1. As a developer, I would like the software to interpret the pitch in well under 1 second.
2. As a developer, I would like the tuning algorithm to run fast and accurately.

Accuracy

3. As a developer, I would like to be able to choose between different tuning algorithms for a configurable amount of precision.
4. As a developer, I would like the product to bring the instrument within 5 cents of the desired pitch.

Desktop App

5. As a developer, I would like the user interface to be intuitive and easy to use.
6. As a developer, I would like the user interface to run on Windows.
7. As a developer, I would like there to be different tabs for viewing the tuning analysis, configuration settings, and user manual.
8. As a developer, I would like the product to connect to the desktop app via USB.

Damage Reduction

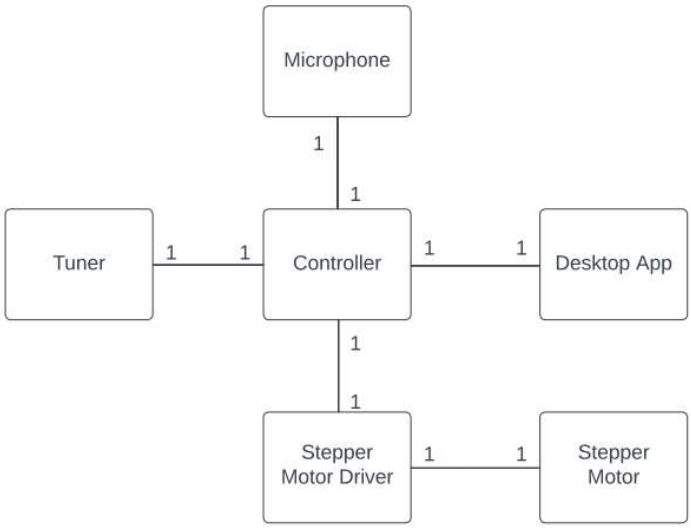
9. As a developer, I would like the software to detect when there is no pitch change when there should be, so it can stop any hardware movement, preventing damage to the instrument.

Hardware Details

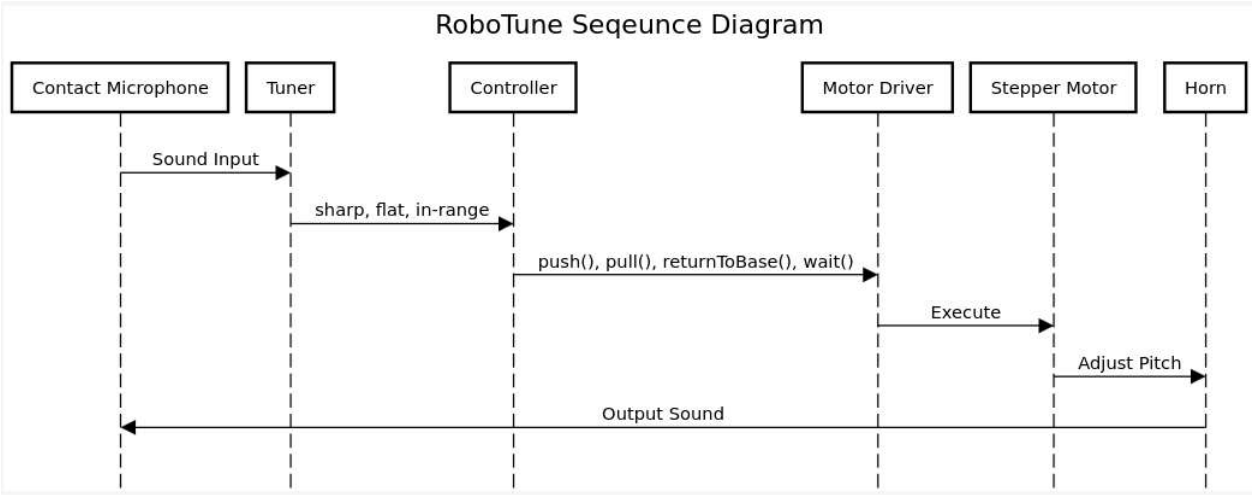
10. As a developer, I would like the hardware to be minimal enough to not interfere with the user's ability to play.
11. As a developer, I would like the hardware to not exceed 1kg.
12. As a developer, I would like the hardware to not be visually distracting.
13. As a developer, I would like the hardware to be easily attached and removed without risk of damaging the horn.
14. As a developer, I would like the battery for the product to last at least 3 hours so it can make it through most performances.

Design Outline

High Level Overview



High Level Sequence Diagram



RoboTune will be constantly working in a cycle as it is being used in order to react as quickly as possible for adjusting the tuning slide according to the user's needs as they play a brass instrument. This cycle will begin with a contact microphone that will be attached to the end of the instrument to pick up the most accurate sound without outside noise like a normal microphone would pick up. The microphone will send the sound data to the tuner, which will

determine the pitch of the instrument being played so we can find if said instrument is out of tune, and if so, in what way (i.e. flat or sharp). Once the tuner determines how the instrument is tuned, the controller can then send commands to the motor driver to push the tuning slide out to flatten the pitch, pull it in to sharpen, or turn off the stepper motor (make it wait) if the pitch is within the acceptable range. As the stepper motor adjusts the tuning slide to fix the pitch of the instrument, the cycle will begin again to continuously adjust the instrument pitch.

The RoboTune device will also interface with a desktop app in order to change instrument settings on the device, load recorded songs, and display recorded songs on a graphical interface to show the difference between the actual and target pitches throughout the song. This desktop app allows the tuner to be used across a variety of instruments, as it allows the user to custom fit their instrument's dimensions.

The controller that we are using is the RaspberryPi 3B. The stepper motor, microphone, and battery will all connect to the controller at defined ports which will then allow the controller to send data to or receive data from specific components when necessary. The controller will be connected to the desktop app by USB and will transfer data through this format. Both the controller and the desktop app will be constructed using Python. This is because a majority of robotics work is written in Python and this allows us to integrate libraries, such as a tuner library, easier.

Design Issues

Functional Issues

1. If the hardware fails how does the software react
    a. Option 1: Shuts down and deletes any currently recorded data
    b. Option 2: Terminates power to the motor and stores the available data
    c. Option 3: Terminates power to the motor, stores the data, and displays an error message

    Choice: Option 3
    Justification: If the hardware fails it must shut off immediately to eliminate risk of damage to both the hardware and the instrument itself. Otherwise, the data that was received until the failure should be saved so it can be analyzed properly later to help determine the cause of the failure. Lastly, an error message or indicator light should be shown/turned on to immediately inform the user of the failure so they stop playing and can fix the issue.

2. If the user selects incorrect inputs
    a. Option 1: Create a quick tutorial at the start of the windows app
    b. Option 2: Make preset inputs for the various instruments

    Choice: Option 2
    Justification: Making preset inputs for the various brass instruments this device could be used on will make it much easier both for the user and in development. This way, the user can simply choose the instrument they are playing in the desktop app, rather than learning how to manually set each input themselves. This also prevents the user from needing to measure out the dimensions of their instrument if they don't already know them. Making a tutorial on how to manually set the inputs could be implemented if time allows.

3. If the user assembles the hardware incorrectly
    a. Option 1: Show simple instructions as part of the initial setup of the windows app
    b. Option 2: Make assembling the hardware more robust (i.e. there's only one way to attach it to ensure it is assembled correctly)

    Choice: Both
    Justification: It will be simple enough to make instructions for how to attach the device to an instrument by using the 3D model we have of the device. For example, we could take screenshots of the model of each step to attach it, then draw and write over the screenshots to make the instructions more clear. Given the design for the device we are thinking of implementing, there should only be

one way to attach the device as well, which should make the assembly even easier.

4. The hardware could damage the instrument and/or the instrument's finish
    a. Option 1: Add rubber or cloth pieces where the hardware attaches to the instrument as a protective barrier
    b. Option 2: Use 3D printed material that will be unable to bend/damage the instrument or its finish

    Choice: Option 2
    Justification: Using 3D printed material for the device will not only make the device harmless to the instrument, it will also be easier to make designs and iterations of the device during development.

5. If the hardware needs to move large distances it will sound off as it can only move so fast
    a. Option 1: Use a fast stepper motor to eliminate this issue
    b. Option 2: Make limits in the software to prevent any movement that is too large and will decrease performance quality

    Choice: Both
    Justification: The use of a fast stepper motor will eliminate a high majority of this issue, but realistically not all of it. Thus, we will need to make limits to the length of movement the stepper motor can make to ensure this issue is resolved fully.

6. If the force needed to overcome the friction on the tuning slide is too much for the stepper motor
    a. Option 1: Stop moving stepper motor
    b. Option 2: Light an indicator

    Choice: Both
    Justification: The stepper motor must stop moving in order to prevent damage to the device and potential damage to the instrument, and an indicator should be lit in order to immediately notify the user of the problem so it can be fixed by them lubricating the tuning slide.

7. If the tuner overcorrects
    a. Option 1: Make the adjustments progressively smaller as it gets closer to the target
    b. Option 2: Use a PID formula to make the adjustment smooth and accurate

    Choice: Option 1

Justification: While using a PID formula would be a better solution that will ensure the tuning will be fast and smooth, it could also slow the overall computation too much to be able to make the adjustments as fast as needed. We can instead simply make the stepper motor move slower once the tuning gets within a certain amount of cents of the target to resolve this issue.

Non-Functional Issues

1. What language will be used for the tuner
    a. Option 1: Python
    b. Option 2: C

    Choice: Option 1
    Justification: Using Python will allow us to use packages that can parse the sound data and give us accurate readings for pitch. While this may be possible with C as well, using Python will allow us to expedite the development process, getting us closer to a functioning prototype by sprint 2 for testing, validation, and improvement.

2. What language will be used for the desktop app
    a. Option 1: Java
    b. Option 2: C
    c. Option 3: Python

    Choice: Option 3
    Justification: By using a common language across all components, it increases ease of compatibility by a large margin. Additionally, Python is the language the team is most familiar with, providing us with the quickest pathway to development rather than learning unfamiliar technology.

3. How will the tuner and desktop app interface
    a. Option 1: Create a common format such that the data is sent from the tuner as a file which is parsed by the software
    b. Option 2: Have the tuner first send the type of data then the data itself so the software

    Choice: Option 1
    Justification: Compiling the data into a .csv file, or something similar, will be the fastest and most efficient method of storing and sending the data. Otherwise the data would be sent in smaller buffers, which would ultimately make the software slower.

4. How will the tuner prevent constant adjustments
    a. Option 1: Control algorithm

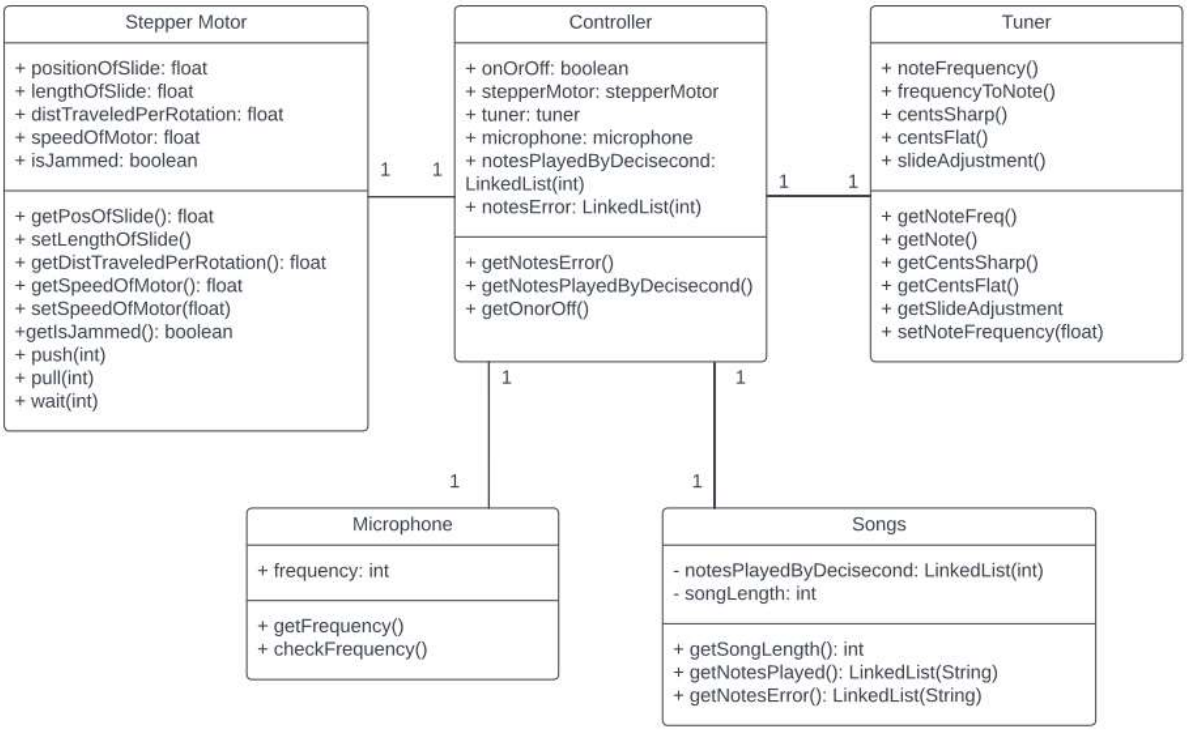
b. Option 2: Don’t move slide unless the adjustment necessary exceeds a certain amount

   Choice: Option 2
   Justification: While a control algorithm would help prevent constant adjustments, the software will be simpler and faster if adjustments are only made once the tuning would exceed a certain amount of cents. This also helps preserve power as the hardware won’t waste energy making micro-movements that won’t be noticed by the ear.

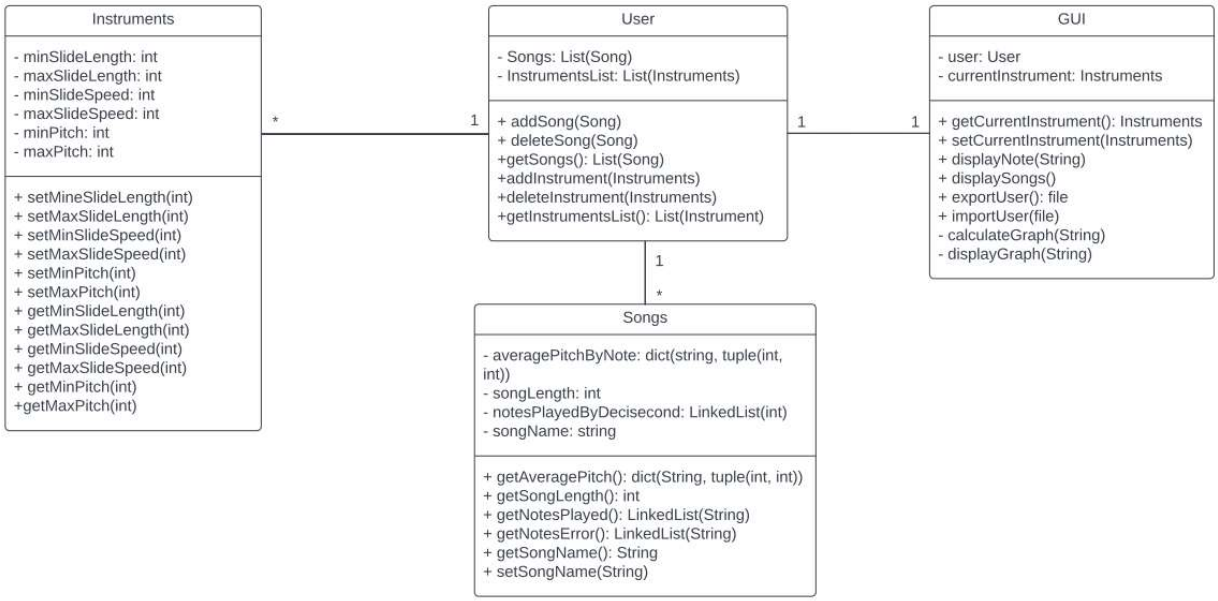
Design Details

Class Design

Controller:



Desktop App:



Descriptions of Classes and Interaction between Classes:

The classes are modeled around objects in the application. In the controller, they are based around physical components and in the desktop app they are based around a feature.

Controller

1. Controller
    a. Represents the raspberry pi controller
    b. The controller sends and receives all information from the desktop app
    c. The controller can be toggled on and off
    d. The controller controls the actions of the tuner by taking information from classes and using it to command other classes.
2. Stepper Motor
    a. The stepper motor object contains the length of the tuning slide and will stop if the tuning slide is close to being separated from the instrument.
    b. The stepper motor can be moved forward and back by using the push(), pull(), and wait commands.
    c. The stepper motor knows the distance one rotation of the lead screw moves the tuning slide.
    d. The stepper motor can detect if there is too much friction in the movement of the motor and will turn off if that is the case.
    e.

3. Tuner
    a. The tuner takes a frequency and finds the note that the frequency is equivalent to.
    b. The tuner finds how sharp or flat the pitch is.
    c. The tuner calculates how far the tuning slide needs to be adjusted and in what direction
4. Microphone
    a. The microphone records the frequency.
    b. The microphone sends the frequency to the controller.
5. Songs
    a. The songs class records all of the notes played.
    b. The song class records how sharp or flat each note played is.
    c. A song is automatically created every time the tuner is started.
    d. The song is sent to the controller when the tuner is stopped.

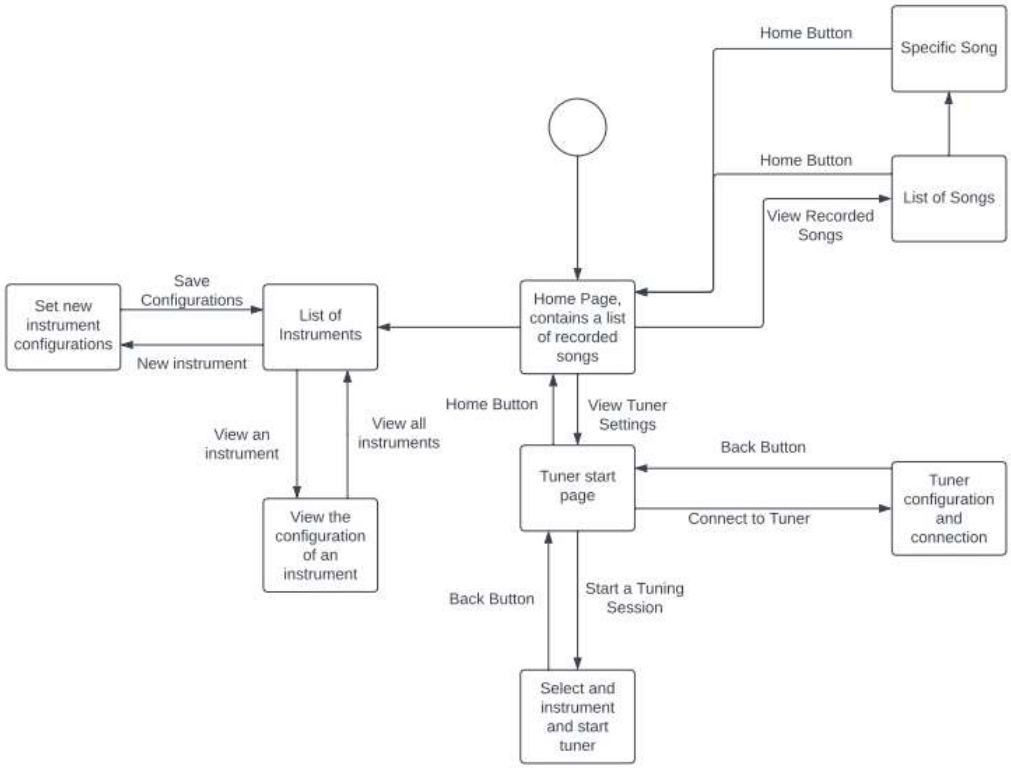
Desktop app:

1. User
    a. The user class keeps all of the songs that the tuner has recorded.
    b. The user class contains all of the different instrument configurations the user has created.
2. GUI
    a. The GUI contains the user class and the current instrument being viewed.
    b. The GUI will access information from the user class to create graphical representations of songs.
    c. The GUI will display statistics on the pitch quality of different songs.
3. Instruments
    a. The users will specify the minimum length of the slide, the maximum length of the slide, the minimum and maximum speed the tuning slide can be moved, and the pitch range.
    b. This information can be accessed so the tuner can act appropriately while being used on different instruments.
4. Songs
    a. The user will be able to name the songs
    b. The song class will contain the notes played during each song
    c. The song class will contain the pitch quality of each note played during the song.

Navigation Flow Map

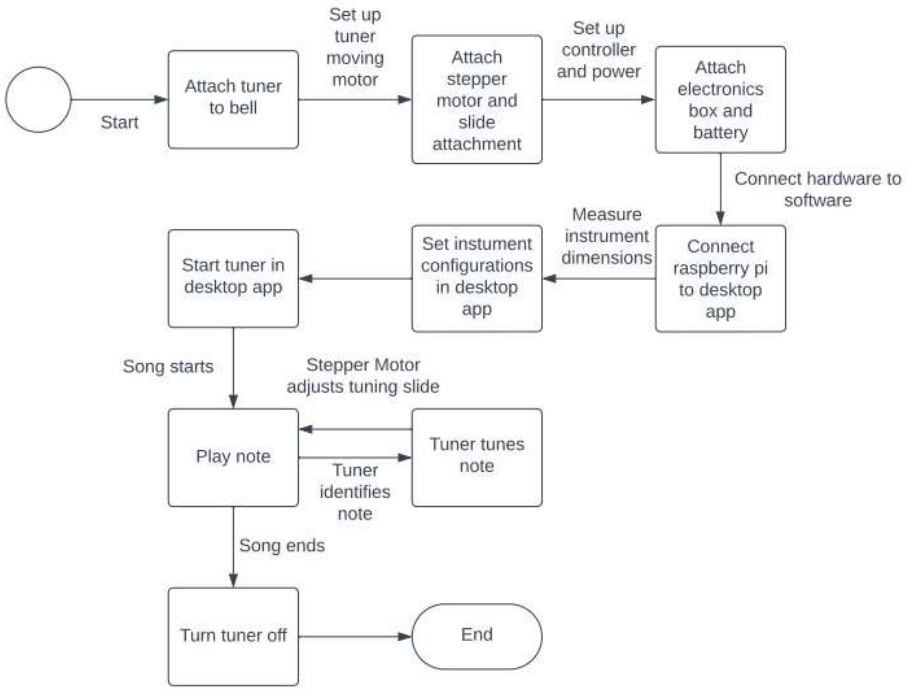
Software Functions:

When the desktop app is opened the user will see a homepage that contains a list of previously recorded songs. Additionally, it will hold tabs to take the user to a full list of songs, the tuner/controller configuration settings, the tuner start page, and a full list of instruments. When the songs tab is selected, the user will be taken to a page that lists all of the songs by their name. The user will be able to delete and rename their songs in this window. From here a song can be selected and a page will be pulled up that shows a graphical representation of the song and statistics regarding the pitch of the user throughout the song. From the main home screen the user can also access the list of instruments. Here a list of all instruments and their names can be found. Instruments can also be created or deleted on this page. If an instrument is created the user will be taken to a page to enter configuration information about the instrument. If the user selects an already existing instrument, the user can view and edit the configuration. Returning to the home page, the user can also start the tuner. This will allow the user to select the instrument and ensure the tuner is configured correctly before starting the tuner. The user will also be able to stop the tuner when the song is over.

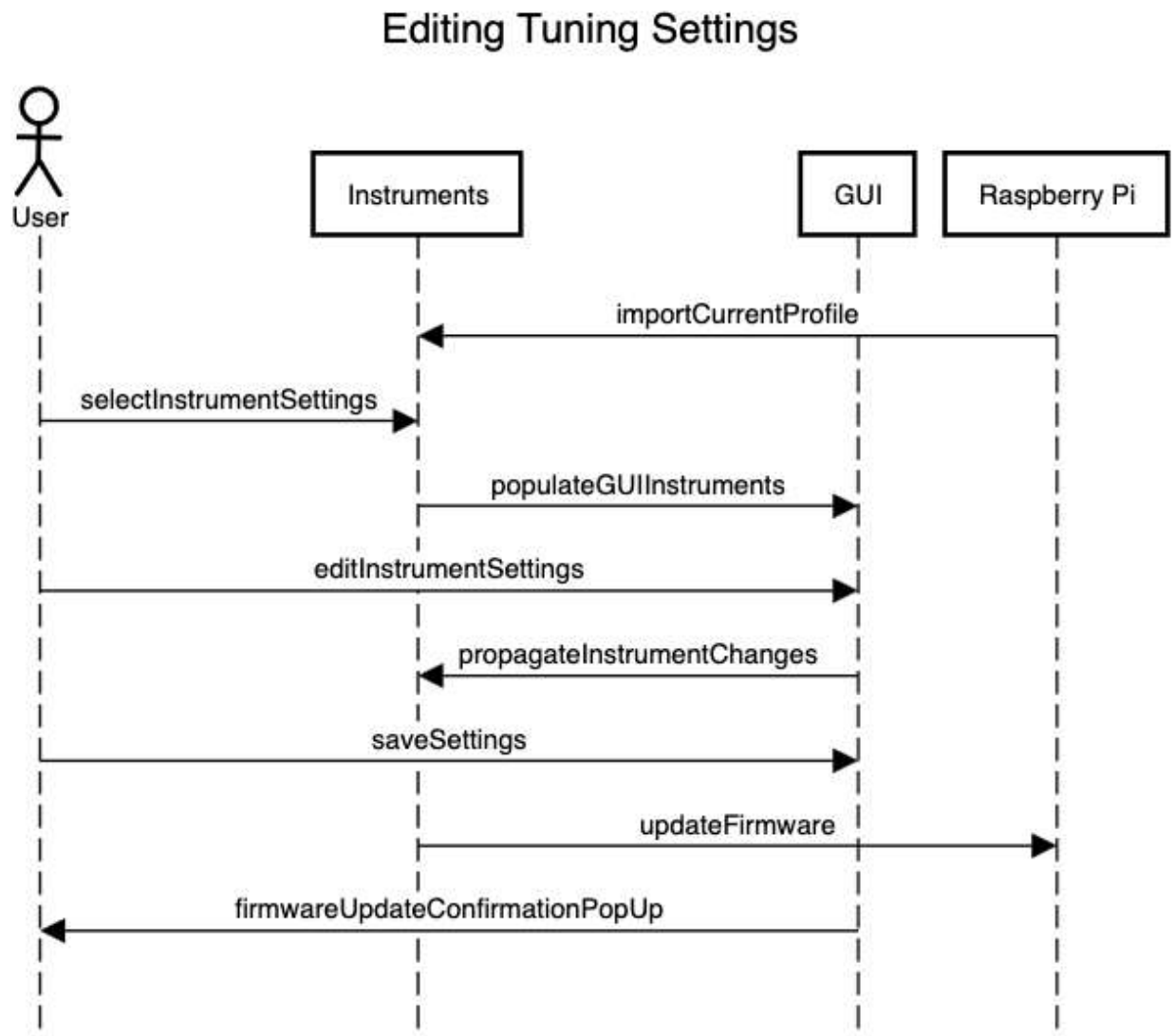


Hardware Functions:

To set up the tuner, the microphone will first have to be attached to the bell of the instrument in order to correctly detect the pitch frequency. Then the stepper motor will be attached to the instrument at three connection points. The design of these connections will ensure that they are correctly placed on the instrument because they will fit snugly around the bars of the instrument and the assembly will be rigid and precise. The controller and battery will then be attached to the instrument and the stepper motor and microphone will be connected to the controller. These electronic components will be contained in a housing and instructions will clearly show how the motor and microphone should be attached. A usb cable will then be connected from the controller to the desktop computer to allow the systems to connect. Next instrument configuration settings will be entered into the desktop app for the specific instrument being used. At this point the device can be started through the desktop app. Once the device is started it will wait until it picks up a frequency. When a note is recognized it will use the tuner to detect the note and the pitch variation and direct the motor to either push or pull accordingly. Then the device will wait to hear another note and continue tuning until the device is turned off.

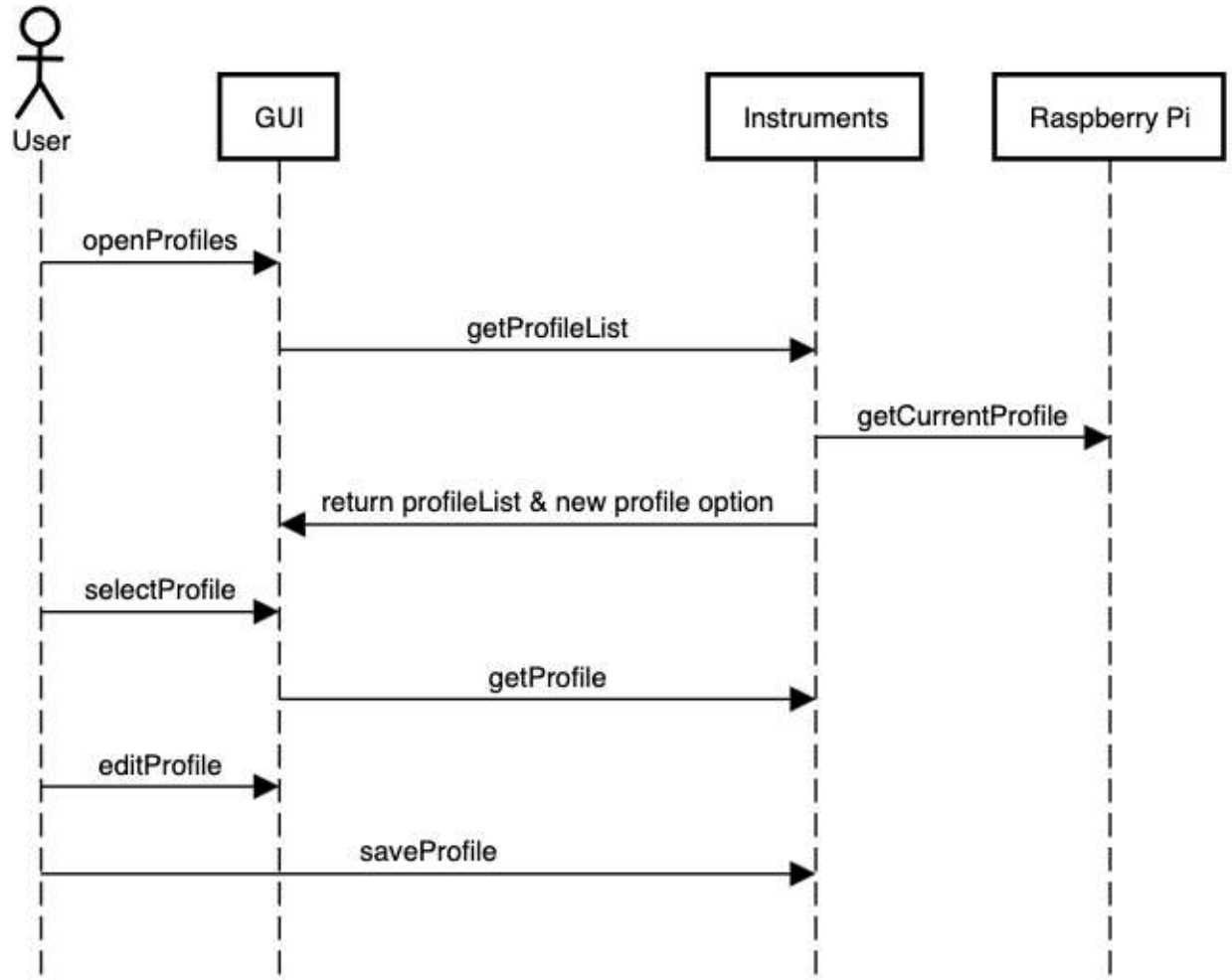


Sequence Diagrams

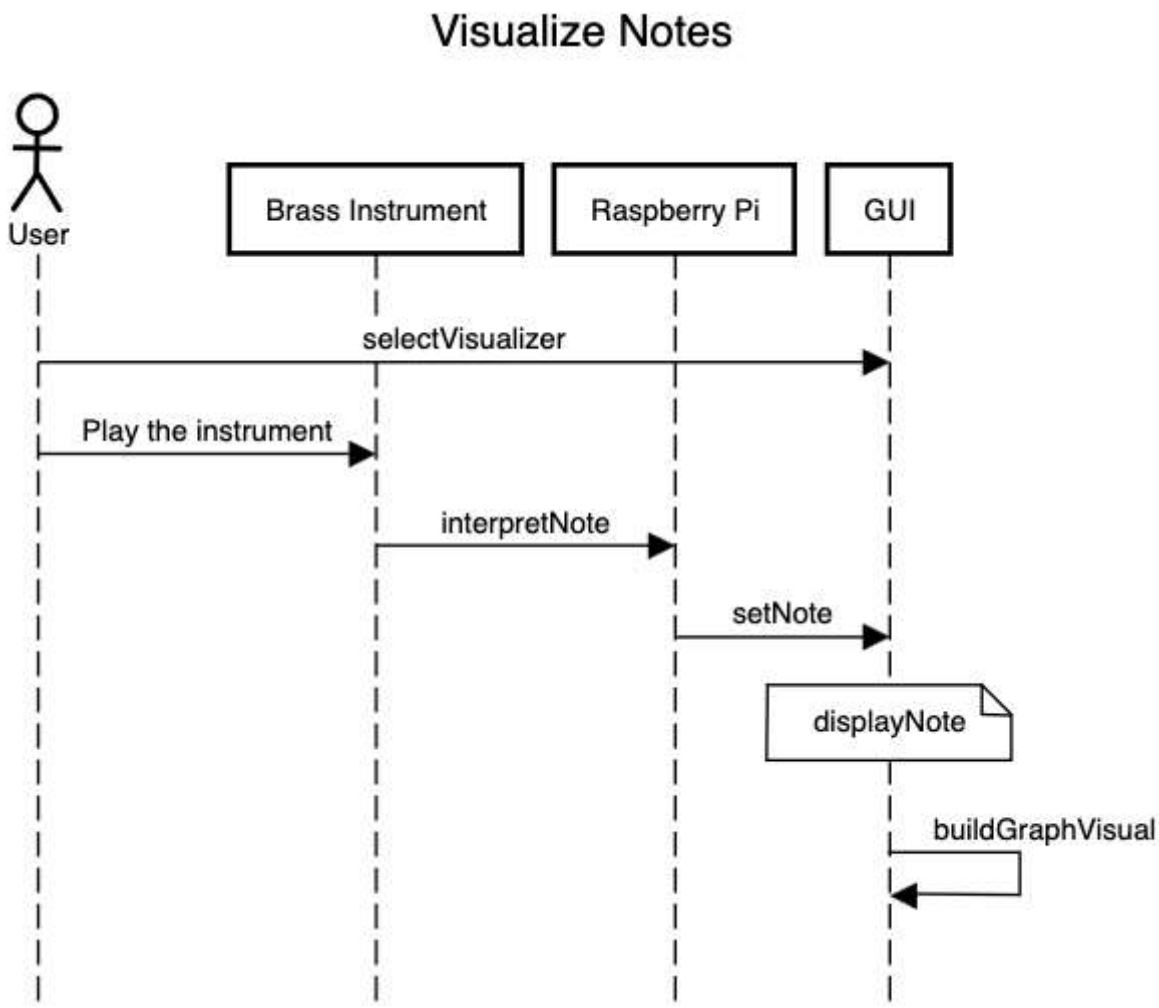


When the user wants to edit instrument configuration settings, they will first select which instrument profile they would like to edit. This can come from their already made profiles, the settings currently in the Raspberry Pi, or by creating an entirely new profile. The GUI will then autofill this profile's settings in text boxes that the user can edit to change the profile's settings. Once the settings are satisfactory to the user, they can update the Raspberry Pi's current settings and they will receive a confirmation box letting them know it was successful.

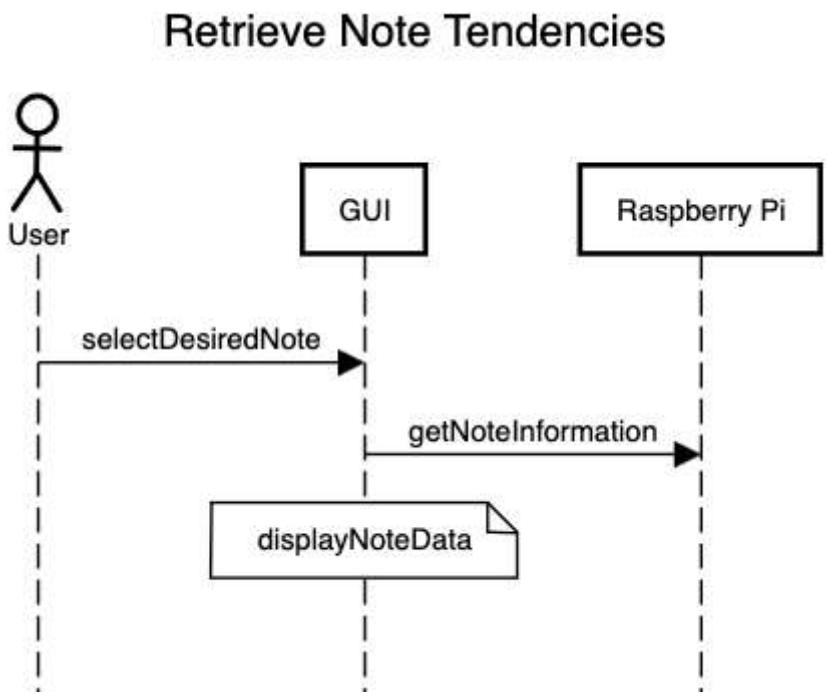
Edit Profiles



When the user wants to edit a profile, the GUI will call for all available profiles, including the current Raspberry Pi profile. The GUI will also have an option to create a new profile. The User will select whichever profile they desire, which will then auto populate the GUI's text boxes. Similar to editing the tuning settings, the user will edit the text boxes as they see fit. Finally, they will be able to save the profile, which will also allow them to assign a name to this profile.



When the user wants live feedback on their performance skill, our desktop app can show them how far away they are from their closest note. As the user plays their instrument, the raspberry pi will interpret the note as shown by the diagram on the top of this document, and send a recorded note to the GUI. The GUI will display this note, as well as the distance the performer's note was compared to the most accurate representation of the note. Eventually, after a brief period of time has passed and enough data has been recorded, the GUI will also display a graph. This graph indicates the notes the user was playing on the Y axis and time on the X axis. This gives the user a visual representation of the performance, and if they have sheet music to compare to, the ability to see where they are messing up.



Sometimes a user will have issues with a specific note that is particularly tough for them. If this is the case, they can see their history of when the tuner recorded that note, and whether the tuner recorded them playing flat or sharp *on average*. It can also say by how much they were off by, in terms of cents. This gives the user personalized feedback on their own weak points.