Technological Feasibility Report

Date: October 23, 2020

Team Name: SongBird

Project Sponsor: Paul Flikkema

Team's Faculty Mentor: Andrew Abraham

Team Members: Kevin Imlay, Daniel Mercado, Yasmin Vega-Nuno, Anqi Wang

Table of Contents

1 Introduction

Birds play an essential role in the health and development of many ecosystems around the world. Pest insect populations are regulated by bird predation, dead animals are disposed of by bird scavenging, and plant seeds rely on birds for distribution and priming for sprouting. For example, in India, vultures are responsible for an estimated $34 billion worth of clean-up of cow carcasses, which in turn reduces the spread of rabies in feral dog populations [1]. In Sweden, it is estimated that it will cost $9,400 per hectare for human seed dispersal services if the Earasian Jay were to disappear from their oak forests [2].

Humans rely on bird populations to help with agriculture as well. A study by Arizona State University and Humboldt State University showed a correlation between bird predation on pest-insects and crop productivity of $310 per hectare per year [3]. Despite this extreme importance of birds, the scientific community still has numerous questions surrounding the behavior of birds and the ways that they communicate. Questions such as where birds eat and nest, how they move from area to area, and how they communicate with each other remain unanswered. Possibly more importantly, it is not known how these behaviors are changing in response to sound pollution, contact with humans, and climate change.

Currently, scientists are using monitoring systems to collect data for their research into these questions. Monitoring systems such as kits of microphones and recorders for in-person monitoring, or remote recording tools such as the AudioMoth, are either costly and difficult to use or are poorly documented and don't allow for customizations to suit their uses. Our client, Dr. Paul Flikkema, a professor at Northern Arizona University, is developing a cheap and easy-to-use sensor system for recording bird vocalizations. Dr. Flikkema has decided on the hardware for the system and needs us to develop the accompanying software to create a vital tool for bird monitoring.

We are developing BiVo, a front-end-back-end system for recording bird vocalizations. BiVo will be open-source and modular for easy modification and addition of functionality. The sensors will be capable of streaming audio and

saving recordings for later retrieval and the desktop application will be able to operate alone or interface with Matlab for easy data analysis.

BiVo will allow scientists to deploy low-cost and open-source sensors to gather data on bird vocalizations. With this data, scientists will be able to conduct meaningful research towards our understanding of birds and their behaviors.

In this report, we will discuss some of the technological challenges we see as being major hurdles and some possible alternatives for solving these challenges.

2 Technological Challenges

BiVo has to take use of the Thunderboard EFM32GG12 board, a development board that incorporates two microphones, and be compatible with Matlab. Below are the major technological challenges we foresee the SongBird team will need to overcome in the creation of our product. Each challenge is accompanied by a brief description.

2.1 Audio Storage

We will need to store audio recordings on the sensor for later retrieval during stand-alone operation. This storage should be able to hold enough audio to allow for at least a day of stand-alone operation.

2.2 Audio Analysis

We will need to analyze the captured audio to determine if the audio contains bird vocalization. This analysis needs to happen in real time as recording is taking place so that the device does not need to stop recording during analysis.

2.3 Desktop Application

We will need a desktop application that is capable of receiving streamed data, downloading data, and manipulating the sensor and its settings. This application will need to be compatible with Matlab, but can stand alone.

2.4 Matlab Compatibility

We will need to integrate the system with Matlab, Dr.Flikkema's choice of statistical software, for easy transfer of data for analysis.

2.5 Wireless Communication

We will need to analyze what wireless technology is suitable for our wireless communication between the sensor and desktop application.

3 Technology Analysis

In this section, we will discuss each technological challenge in more detail, present the possible methods for approaching the challenge, and rank the options by feasibility of implementation. We also include our next steps in determining the feasibility of these solutions.

3.1 Audio Storage

To allow the sensor to run on its own the sensor needs to be capable of saving audio that contains bird vocalizations. To do this, the sensor will need to save audio to some sort of Flash memory.

3.1.1 Desired Characteristics

Ease of Implementation - The difficulty of implementing the memory module into the project.

Storage Capacity - How much data can be stored on the alternative in question.

Driver Support - Do the alternatives have drivers to support their implementation.

Cost - The cost of supplemental memory may influence our decision on which alternative we may pick.

Power Consumption - The amount of power needed to supply energy to the device.

3.1.2 Possible Solutions

On Board Flash - The on board Flash memory capacity on the board is 1024 kB. Flash was developed by Fujio Masuoka, a Toshiba electrical engineer, who patented the idea in 1981 and released the first Flash chips in 1987. Flash memory is used for data storage and does not need a power source to retain that information.

SPI Flash - An SPI Flash module is a memory module that is interfaced over SPI (Serial Peripheral Interface), a bus protocol for accessing Flash memory

which was developed by Motorola in the mid 1980s. It is used for different purposes such as storing data files and code.

SD card - An SD card, or a secure digital card, is a small and removable storage device that was first introduced by the SD Card Association. This type of memory module is commonly used to store large amounts of data from mobile devices such as cameras and smartphones.

3.1.3 Analysis

The alternatives above will be analyzed by considering the ease of implementation, the amount of data that can be stored, whether or not there is driver support for the alternative, how expensive the device is, and how energy efficient the devices are.

On Board Flash - The on-board Flash is an enticing option because the Flash is integrated onto the Thunderboard EFM32GG12; in other words, that means there is no extra cost on our end. Because the Flash is on the board, it is easier to implement and there is driver support. Power consumption is the lowest out of the alternatives, the specs are as follows: 80μA/MHz EM0 Active current and 1.9μA EM2 Deep Sleep current. Although, the drawback is that the storage capacity of the Flash is quite small at approximately 25.6 seconds of audio.

SPI Flash - SPI Flash is generally a low cost device ranging from a couple of cents to a few dollars. A benefit of SPI Flash is that it is able to store more audio than the on board Flash by being capable of holding a few minutes of audio, which is better than a couple of seconds. SPI Flash usually has a power consumption in the range of μA or lower mA, which is a bit more than the on board Flash, but it is still more efficient than the SD card. The ease of implementation is quite low; our client advised us that extra memory modules are a complex matter. Although, Simplicity Studio has an SPI Driver to handle the situation.

SD Card - The SD card is able to store a significant amount of data (GB range), more than both SPI Flash and the on board memory. Like the SPI Flash, our client advised us about the complexity of implementing supplementary

memory, but Simplicity Studio has a SDIO driver to handle the situation. A drawback of implementing an SD card is that it is the most expensive option out of the alternatives ranging from a few dollars up to $30.00 and up. In addition, the SD card has the highest power consumption of the two alternatives ranging in the mA and even up to 100 mA.

3.1.4 Chosen Approach

Having these details in mind, consider Table 1, where the ideas are visualized in a table and the alternatives are ranked for each factor from Low to High. The exception is driver support, which is a Yes or No value.

	On Board Flash	SPI Flash	SD Card
Ease of Implementation	5	3	3
Storage Capacity	1	2	5
Driver Support	5	5	5
Cost	5	4	3
Power Consumption	5	3	2
Total	21	17	18

Table 1. Comparison table of the ease of implementation, storage capacity, driver support, cost, and power consumption for the onboard memory, SPI Flash, and SD card.

From our analysis, we've decided that not adding an additional QSPI Flash or SD card module is our best option. While this is not ideal, power consumption is a major consideration and would take additional considerations from a hardware standpoint. Additional factors with high importance are ease of implementation and cost. As can be noted from Table 1, the onboard flash ranks the highest in the power consumption, ease of implementation, and cost categories. Therefore, while an SPI flash and SD card module rank higher in memory, the above three important factors take priority leaving us with on-board memory triumphing over the other alternatives. Nonetheless, we

will keep the option of adding an SD card module as a secondary option and revisit it in the future if time allows.

3.1.5 Proving Feasibility

Moving forward, we will continue to assess if the on-board Flash is suitable for our needs at the moment. We made assumptions that the on-board Flash would be used solely for audio storage, but it is likely a portion of this will be reserved for other purposes.

We will create demonstrations showing our capabilities to read and write to the on-board Flash such as directly recording and saving audio, and playing it back on a computer.

3.2 Audio Analysis

Our sensor will need to analyze audio as it is being captured to determine if the audio contains bird vocalizations. This analysis will tell the sensor if it needs to store the audio or delete it. One of the major challenges of this analysis is that it needs to take place in the limited space and processing power of the Micro Controller Unit in our sensor. Another major challenge is that analysis must happen fast enough that it keeps up with the incoming audio samples.

3.2.1 Desired Characteristics

Lightweight - Needs to be small in code size and not be processor intensive on the microcontroller.

Ease of Implementation - Should be easy to implement/extract the needed code and use within the rest of the system.

Documentation - Should have complete and detailed documentation.

3.2.2 Possible Solutions

Aubio - Aubio is a C/Python library for listening and detecting events in audio. It provides analysis tools that are common and useful for basic audio analysis. Aubio was created by a team in 2003 and was maintained by that team until 2018. Since then, it appears the library has been maintained by 19

contributors in Github with the most recent contribution July 2, 2020. We found this option while searching Github for audio analysis tools in C.

LibXtract - LibXtract is a C/C++ library for audio feature extraction. It was created by Jamie Bullock in 2012 with the purpose of creating a lightweight and simple tool that does not perform calculations any more than necessary. It appears that main development ended in 2014 and has since been maintained by 6 contributors, with the most recent contribution July 16, 2019. We found this option when googling open source audio analysis tools in C.

Custom Code - We can create custom code from scratch made specifically for this project. We discussed this idea when finding a lack of C-based and open source audio analysis tools.

3.2.3 Analysis

Our abilities to perform preliminary testing for feasibility for this challenge is impractical without having the hardware the code will be running on. Due to this, our analysis is based solely on research and reasoning. As we get our hands on the hardware we will be using, we may find we have ranked these alternatives incorrectly.

Aubio - Since we can only use C code in our environment, it is possible the useful tools can't be used if they are in Python. There is also no clear design choice for making it lightweight and takes a total of 1.8 MB on disk, which can also be a major issue. Aubio does provide documentation but it is fairly limited.

LibXtract - It is designed to be lightweight and portable, and is designed to remove redundant calculations. However, LibXtract is 1.9 MB on disk, which can be a major issue. Also, half the library is in C++, meaning there is a good chance important functionality would not be able to be used. Documentation is also very limited for this library.

Custom Code - Creating custom code has the downside of adding more time for implementation, but allows for designing code that is optimized for the sensor's needs. We would also be able to create very extensive documentation for this code fairly easily.

3.2.4 Chosen Approach

With all of this in mind, we have evaluated each of the alternatives against the desired attributes. Each evaluation is ranked on a scale of 1 to 5, with 5 being the best. Our table describing this is below in Table 2.

	Aubio	LibXtract	Custom Code
Lightweight	1	3	5
Ease of Implementation	2	2	2
Documentation	3	1	5
Total	7	6	12

Table 2. Comparison table of the lightweightedness, ease of implementation, and documentation of Aubio, LibXtract, and custom code.

From our analysis, we've decided creating custom code from scratch is the best option. We had ranked custom code the same as the other alternatives because the documentation for them is lacking and would require us to read the code and possibly update it to match our needs. Custom code creation will allow us to make it tailored to the system's restrictions and to create in-depth documentation.

3.2.5 Proving Feasibility

Moving forward, this will be our greatest challenge and will require significant work and testing to overcome. We will continue to test the feasibility of writing our own code by creating a demonstration where the hardware board will perform real-time simple audio analysis and output the result of that status to a computer for viewing. These analyses may include fourier transforms, spectrogram generations, and frequency analysis.

3.3 Desktop Application

We will need a command line interface (CLI) capable of interacting with the sensor. This CLI will ultimately need to be able to connect to the board and view its data, delete its data, download the data, review its status, and

manipulate that status. To create such a CLI, we will need to look into which language best suits our needs.

3.3.1 Desired Characteristics

Ease of implementation - This will determine how difficult it is in implementing the command line interface with a language.

Portability - How easy it is to install the application on multiple types of computer systems.

Libraries - Language libraries that support the functionality of the application.

Documentation - How well documented the language is and how readable that documentation is.

3.3.2 Possible Solutions

Python - Python is a general purpose programming language known for its ease of use and its readability. We considered using Python as a language option while searching forms discussing languages used for command line interfaces. Python was created by Guido van Rossum in 1991 with the aim of creating a highly-readable object-oriented language. Since then, Python has been adopted by many industries including those in science and engineering.

Node JS - Node JS is a cross-platform environment used to run JavaScript outside a web browser. We considered using Node JS while searching forms discussing languages used for command line interfaces. Node JS was created by Ryan Dahl in 2009 and is typically used for web development, command line utilities, and real time applications due to its event-driven, non-blocking I/O model.

C++ - C++ is a general purpose programming language with many low-level capabilities. We considered using C++ while searching forms discussing languages used for command line interfaces. C++ was created by Bjarne Stroustrup in 1985, and is considered an extension of the C programming language. This language can be used for operating systems, compilers, and web browsers.

3.3.3 Analysis

Each of the alternative options discussed above were analyzed for ease of use, portability, support libraries, and documentation. Ease of implementation was determined by our knowledge and prior experiences with these languages. Portability was determined by researching how runtime environments work and our prior knowledge of these languages. Libraries was determined by researching the language references, language websites, and Github. Documentation was determined by researching the language references.

Python - Python is very easy to use both syntactically and logically. Because Python runs on top of a runtime environment, it is highly portable. There are numerous libraries provided both by Python itself and developers on Github. The documentation for Python and its libraries is excellent. Two of our members have a steady foundation of Python, one who has experience from five years using Python for classes and personal projects, and the other having used it for about two years in both classes and projects. One member has learned the language on their own with less experience in comparison with the two above, and one other member has not taken any courses with Python and as such, has no experience in the language.

Node JS - JavaScript is easy to use but does have some oddities, particularly with counter intuitive logical comparisons and confusing scope rules. Because Node JS acts as a runtime environment that the JavaScript runs on top of, Node JS code is highly portable. Node JS and LavaScript are a bit lacking in libraries for command line interfaces, and okay documentation is limited or unclear at times. Two members have a steady foundation in Javascript with one member having had a five month internship using Javascript, and the other having learned Javascript for a year and a half through courses. Two other members have less experience with the language than the two above, one member has used the language in one class and in a current project, and the other member has used the language in one class and sparingly in a second class.

C++ - C++ allows for detailed control over the program, but because of this, makes implementation a bit more difficult. C++ is very portable but requires compilation to the specific system it's being run on. There are limited libraries for C++, and the available documentation is difficult to read and sometimes

nonexistent for some libraries (or at least very difficult to find). Two members of the group have a steady foundation in C++, one having used it since two years ago in classes and projects, and the other having one year of experience through classes. Two members do not have experience with C++, but have experience with the language's predecessor, C. One member has learned the language through courses for a year and a half, and the other has learned the language for almost a year and a half, but does not have the same steady foundation.

3.3.4 Chosen Approach

With all of this in mind, we have evaluated each of the alternatives against the desired attributes. Each evaluation is ranked on a scale of 1 to 5, with 5 being the best. Our table describing this is below in Table 3.

	Python	Node JS	C++
Ease of Implementation	4	4	2
Portability	5	5	4
Libraries	5	4	3
Documentation	5	3	3
Total	19	16	12

Table 3. Comparison table of the ease of implementation, portability, available libraries, and documentation of Python, Node JS, and C++.

From our analysis, we've decided that Python is the best choice of programming language to build our desktop application in. Python provides many excellent libraries for the functionalities that we need, is easy to learn, and is very portable because it is interpreted in its run time environment. Node JS, like Python, is highly portable because it is interpreted in its runtime environment. Node JS is a bit lacking with documentation and doesn't provide nearly as many useful libraries as Python does. C++ is a bit more difficult to use than Python and Node JS, giving it a lower score in ease of implementation. While there are libraries that are common with C++, many are poorly

documented or very out of date. C++ also has the disadvantage of having to compile on every machine it runs on, making it less portable than Python and Node JS.

3.3.5 Proving Feasibility

Moving forward, our plans for testing the use of Python is to create a demonstration for sending and receiving data from the board. We will use Python's serial communication library (pySerial) to send the command line input to the board over a USB connection, and to receive the echo of the command line input from the board. This will ensure that we are able to set up a connection between the two.

3.4 Matlab Compatibility

Our desktop application will need to either be compatible or integrated with Matlab. This will allow users to perform advanced analysis of the sound data collected from the sensors.

3.4.1 Desired Characteristics

Ease of Implementation - It should be easy to perform the functionality needed from the platform/language.

Flexibility - There should be flexibility in capabilities and ability to add features.

3.4.2 Possible Solutions

Matlab Based Solution - Matlab provides functionality to call libraries in other languages within the Matlab environment. This interface is compatible with calling C, C++, MEX, Java, Python, .NET, and COM. Many structures within these languages can also be used directly within Matlab. This solution was found while looking through Matlab's Help Center website.

Library Based Solution - Matlab also provides libraries to call Matlab within another language's environment. These libraries are compatible with C, C++, Java, Python, Fortran, and COM. This can help add Matlab functionality to the desktop application. This solution was found while looking through Matlab's Help Center website.

Indirect Compatibility - To make our desktop application indirectly compatible with Matlab, we would have to make sure any files we want to transfer between them are compatible formats. This would not require any real integration with Matlab but would make it possible to use with Matlab and other languages such as R. This solution was thought up during a discussion about design considerations.

3.4.3 Analysis

Each of the alternatives were evaluated for ease of use by implementing a simple test program to make sure the environments can, in fact, call each other. For the flexibility characteristic, design ideas were drafted and reasoned to assess the advantages and disadvantages.

Matlab Based Solution - Calling libraries from other languages has some major advantages. This would allow us to write a stand alone application and build on top of it within Matlab. Matlab also provides a GUI builder, giving us the option to create a GUI within Matlab that can control the desktop application. This solution also has the benefit of being able to easily add more Matlab functionality because it is within Matlab. A disadvantage to this approach is that we would need to create two components to the desktop application: one for the actual control and communication with the sensor and a second one for functioning within Matlab.

Library Based Solution - Calling matlab functionalities from within another language environment has the advantage of creating one application to perform both Matlab and control and communication functionalities. This does have a disadvantage for flexibility and ease of implementation, where adding Matlab functionality requires changing the application code instead of working directly in Matlab for functionality. We also would have to find additional support for adding a GUI if desired.

Indirect Compatibility - Creating our application to export files compatible with Matlab has the advantage of leaving all Matlab functionality within Matlab. This makes for a simple solution for compatibility, but has the disadvantage of adding complexity to usage. We would also still need to find additional support for adding a GUI if desired.

3.4.4 Chosen Approach

With all of this in mind, we have evaluated each of the alternatives against the desired attributes. Each evaluation is ranked on a scale of 1 to 5, with 5 being the best. Our table describing this is below in Table 4.

	Matlab Based Solution	Library Based Solution	Indirect Compatibility
Ease of Implementation	4	4	5
Flexibility	5	3	2
Total	9	7	7

Table 4. Comparison table of the ease of implementation and flexibility of the Matlab based solution, library based solution, and indirect compatibility solution.

From our analysis, we've decided that the Matlab based solution is the best option. While the indirect compatibility approach was the easiest to implement, it provides little flexibility with the addition of features. The library based solution had the same ease of implementation as the Matlab based solution, but falls short in flexibility for the same reasons the indirect compatibility approach does. The Matlab based solution provides excellent flexibility with addition of Matlab features and is only marginally more difficult to implement than not using Matlab at all.

3.4.5 Proving Feasibility

Moving forward, we will continue to test the feasibility of the Matlab based solution. Testing will consist of writing test libraries and running them through Matlab's external language interface to import dummy data and manipulate dummy settings on a simulated sensor. A simplistic GUI will also be included to test the feasibility of the GUI creation system within Matlab.

3.5 Wireless Communication

In order to make our wireless transmission more stable, more convenient and faster, so we will discuss the wireless technology used by this voice collection device.

3.5.1 Desired Characteristics

Ease of Implementation - The difficulty of implementing the wireless module into the project.

Large range of connections - It should be suitable for a wide range of connections.

High Security - We need a secure wireless communication method that can better protect project data privacy.

Low Power Consumption - It should be power saving and low energy consumption

Real-time transmission - The collected audio data should be transmitted in real time

Fast data transmission - It should transfer data quickly.

3.5.2 Possible Solutions

Wi-Fi - It is a wireless local area network technology created based on the IEEE802.11 standard. The coverage of Wi-Fi technology is generally within 100 meters, the technology is more complex, the transmission rate can reach 54Mbps, the working frequency band is 2.4GHz, and the transmission power is less than 100mW. Compared with Bluetooth wireless communication, the data security performance is relatively poor. [4]

Bluetooth - Bluetooth is a short-range wireless communication technology. It uses 2.4~2.485GHZUHF radio wave ISM in the frequency band. Bluetooth wireless technology has high complexity, fast device networking, only 10 seconds; high integration and reliability; transmission rate is generally 1Mbps; low cost, relatively simple installation. [4]

Zigbee - ZigBee wireless communication technology is a low-power local area network protocol based on the IEEE802.15.4 standard.

ZigBee technology was developed to meet the needs of industrial automation, and has the characteristics of simple layout, anti-interference, reliable transmission, convenient use, and low cost. The communication distance is extended to 10 meters. From the opening distance to a few hundred meters, it can reach about 50 meters in indoor scenes. [4]

3.5.3 Chosen Approach

With all of this in mind, we have evaluated each of the alternatives against the desired attributes. Each evaluation is ranked on a scale of 1 to 5, with 5 being the best. Our table describing this is below in Table 5.

	Wi-Fi	Bluetooth	Zigbee
Ease of Implementation	3	5	1
Large range of connections	5	2	3
High Security	1	4	5
Low Power Consumption	1	3	5
Real-time transmission	4	4	2
Fast data transmission	5	4	2
Total	19	22	18

Table 5. Comparison table of the ease of implementation, strong stability, high safety, low power consumption, real-time transmission and fast data transmission based Wi-Fi, Bluetooth and Zigbee.

From our analysis, we've decided that Bluetooth is the best option. While the indirect compatibility approach was the easiest to implement, it provides a small range of connection. Although Zigbee does very well in the part of Low Power Consumption and High Safety, it is hard to implement and it also has some disadvantages in real-time and fast data transmission. The most serious problems of Wi-Fi are the low security of data and high power consumption. The Bluetooth provides excellent performance in high security, real-time and fast data transmission.

3.5.4 Proving Feasibility

Moving forward, our plan to test Bluetooth usage is to demonstrate how to transfer the received data to a desktop application. We will use Bluetooth to

transmit the collected data to the desktop application in real time via a wireless connection. This will ensure that we can establish a connection between the two.

3.6 Trivial Solutions

We need to discuss the technology that is required in the project, but has no other alternatives based on our environmental requirements. This section will feature multiple technologies explaining what the technology is and why we do not have multiple alternatives for them.

3.6.1 Development Board

The type of board we are using for this project is the keystone into why the Trivial Solutions section exists, we have a predetermined board from Dr.Flikkema named the Thunderboard EFM32GG12. This board contains two PDM microphones, EFM32 Giant Gecko 12 Microcontroller , and lastly flash memory that is about 1024kb of storage. This was chosen by Dr.Flikkema based on the microcontroller being powerful for a microcontroller as well as having microphones on board as well as his own research into microcontrollers that fit the requirements of this project.

3.6.2 Microphones

The microphones are also a trivial solution because of them being installed on the board itself. This however is not a limitation considering the connection is directly to the microcontroller making it easy to analyze and stream whenever necessary. This also allows us to be able to grab and save data easier without having to get any drivers that are not supported by the board allowing for easier integration with the rest of the project.

3.6.3 Wired Communication

The Thunderboard EFM32GG12 has a debugging capability that connects itself to a computer using a micro USB port on the board. This can be used to maintain a serial communication that has a format of 115200 bps, 8 bits, no parity, and 1 stop bit. Knowing this exact format helps us integrate a communication from the user interface to the board with ease or from the board to the user interface to send commands to the board. Since this is

installed and supported by the Thunderboard we do not need to worry about any alternatives for the serial communication on the boards side. On the user interface side there is a Python library that supports serial communication called pySerial, this is supported by the Python Software Foundation allowing us to ensure that this will work correctly with the user interface itself considering it is written in Python.

4 Technology Integration

All of the technology choices discussed in this document will have to work together to create the whole system. In this section, we will look at how each of these technologies will be integrated with each other. Below we have provided a flowchart to help visualize how each technology will fit into the system (Figure 1).

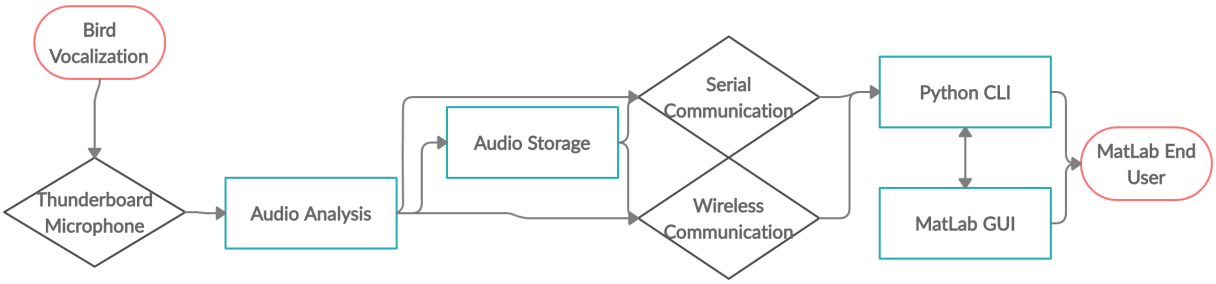

Figure 1. Flow chart showing how the technological choices fit together to create the system.

Following is an ordered list of operations describing the flowchart in Figure 5.

1. Audio Analysis gathers the recorded audio using the Thunderboard microphone and analyzes the sound to make sure that it contains bird vocalizations. This will be done using our own custom algorithm that is optimized for working within the constraints of the sensor's hardware.
2. The analyzed audio will then be passed to either the Python command line interface for audio streaming, or stored in the Flash storage supplied by the board using the serial wired connection or wireless connection.
3. The Python CLI will communicate with the sensor to gather the data saved in the audio storage, accept streamed data from the sensor, and manipulate the settings and state of the sensor. The CLI can also be called by Matlab to enhance the end-user's experience, or it can be used stand-alone.
4. The Matlab GUI will be built on top of the Python CLI and can download the incoming data within the Python CLI. This gives the user a GUI for downloading and streaming data from the sensor, and manipulating the sensor's state and settings.

5 Conclusion

Birds play a very important role in ecosystems around the world. Without birds, many plants and animals would not survive and ecosystems would fall apart. Consequently, it is imperative to learn more about the behaviors of birds and how human interactions are influencing them.

Currently, scientists use sound recording devices to track what birds are where. The sensors can be expensive and cumbersome, or they are cheap and inflexible. We are designing BiVo to solve this issue. BiVo will be cheap and flexible to allow the user much more control over what the sensor does.

There are many technological challenges that we will need to overcome in the creation of BiVo. The most challenging of these are storage for saving recordings, analyzing the audio so we only save data with bird vocalizations, a desktop application that can manage the BiVo sensors, and Matlab compatibility to give users the ability for advanced analyses. These challenges may have many solutions, so we have analyzed each one and picked what we believe will suit our project best. Each of our decisions are listed below.

- Audio analysis will be done using our own custom code because it allows for the most flexibility and compatibility with the sensor's board.
- Audio storage will be done primarily with the on-board Flash memory because adding additional storage can complicate the software and hardware design. We will revisit this in the future.
- The desktop application will be developed in Python because Python provides great resources to help with development and is highly portable.
- A GUI will be built in Matlab connecting the Python application's functionality to Matlab's environment.

Our analysis shows clear advantages for the technologies we have chosen to use. These analyses are mostly preliminary, and it is possible we will find our choices will not work. Moving forward, our next step is to create several demonstrations to test that our chosen technologies work well together and are able to fulfill the requirements in completion.

6 Works Cited

1. Peisley, R. K., Saunders, M. E., Robinson, W. A., & Luck, G. W. (2017). The role of avian scavengers in the breakdown of carcasses in pastoral landscapes. *Emu-Austral Ornithology*, *117*(1), 68-77.

2. Hougner, C., Colding, J., & Söderqvist, T. (2006). Economic valuation of a seed dispersal service in the Stockholm National Urban Park, Sweden. *Ecological economics*, *59*(3), 364-374.

3. Johnson, M. D., Kellermann, J. L., & Stercho, A. M. (2010). Pest reduction services by birds in shade and sun coffee in Jamaica. *Animal conservation*, *13*(2), 140-147.

4. Jotrin, https://www.jotrin.jp/technology/details/types-of-wireless-communication-modules.

Female Eastern Bluebird in Flagstaff, AZ. Imlay, K. 2017.