BiVo Final Report

Date: April 27, 2021

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. For example, the Clark's Nutcracker disperses the seeds of the Whitebark Pine in western North America. For human efforts to do the same, it is estimated to take $12.6 billion annually for the entire range of these pine trees. In the Southeastern US salt marshes, oystercatchers, curlews, and plovers feed on the Salt Marsh Periwinkle snails, keeping their populations in check. Without these birds, the snails would completely destroy the marshes, leaving mudflats where they used to stand.

To better understand this relationship with birds, we need more in-depth research. There are over 180 organizations globally that are dedicated to the research of birds, and among many things, how humans are affecting them. The National Audubon Society is a society of zoologists who study birds around the world. They have over 450 chapters local to just the United States. The USGS Bird Banding Laboratory is dedicated to tracking and banding birds in North America, and has banded over 77 million birds since its founding 101 years ago.

However, despite the 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 developed BiVo, an open-source foundation for remote monitoring of bird vocalizations. BiVo is open source and developed on widely-available hardware to let the user easily create and modify their own sensors. Audio is analyzed to keep only segments that contain bird vocalizations, so that the user doesn't have to listen to them to determine if they are useful.

2 Process Overview

For our development process, we decided on an Agile-hybrid approach. Every standard, weekly meeting every member of the team would 'stand-up' and explain what they accomplished since the last meeting, what challenges they were facing, and what they were going to work on for the next week. We also met with Dr. Flikkema on a weekly basis to communicate with him about our development status and get any feedback he had. To help us assign and keep track of tasks for each of the team members, we used an issues organization plugin on GitHub.

We used many tools to aid us with our development. For version control, we used Git in cooperation with Github. For designing and drawing diagrams and flowcharts we used Draw.io. We also used Google Drive and MS Teams for document creation and cooperative editing, and we used Dropbox and email for sharing documents, photos, and code with Dr. Flikkema and Roo, our mentor.

Each team member also played a crucial role in the development of BiVo. Daniel was the Release Manager, making sure that our GitHub was always working correctly and fixing issues when they appeared. Yasmin was our team Recorder and took meeting minutes for all of our important meetings with ourselves, Dr. Flikkema, and Roo. Anqi was our architect and made sure we followed our development plans. Kevin was the team lead, making sure we stayed on task and on track to completing the project, and the customer communicator, acting as the link between Dr. Flikkema and the team.

3 Requirements

Our requirements acquisition process began early in the Fall 2020 semester. As a team, we communicated with Dr. Flikkema about what requirements BiVo should fulfill and what goals we should strive for on the side in order to create a well-rounded and robust solution.

By exchanging ideas and solidifying what BiVo has to accomplish, we had finalized a number of functional and nonfunctional requirements. At its core, the BiVo system will be using the Thunderboard EFM32GG12 by Silicon Labs. By using this board, the use of the Simplicity Studio IDE is also required to start development on the board. The Thunderboard will use its microphones to record audio in configured length segments from its environment. The audio segments are to then be analyzed using frequency filtering to determine if the audio segment in question contains bird vocalizations. If the segments are flagged as containing bird vocalizations, the sensor will forward the segments to the desktop application through a serial connection. The desktop application brings the audio segments into MATLAB where the user can play the audio and inspect its corresponding spectrogram.

4 Architecture and Implementation

The BiVo system consists of two major components: a sensor and a desktop application that manages it. Figure 1 displays five back-end modules that are contained within the BiVo sensor component and three front-end modules that are contained within the desktop application component.

Consider the use case of an end-user wanting to record audio within an environment. First, in order to initiate recording, the GUI module would call on the desktop application's custom communication module. This module would in turn set up the connection between the computer and the Thunderboard via a handshake with the help of the desktop application's USB communication module. In order to complete the handshake, messages from the desktop application communication modules will need to contact the

sensor's communication modules, whose USB communication module will send confirmation messages. Once complete, the desktop application's USB communication module sends a message to the sensor's communication end to start recording. The record message will signal to the sensor to utilize the Microphone Driver to record audio using its on-board microphones. The sensor would send the recorded audio to the Audio Analysis module, where its responsibility is to determine if the recorded audio contains bird vocalizations. If the recorded audio segment is marked as having bird vocalizations, the sensor will send the audio segments back up to the communication modules on the sensor side, where it will then be sent to the communication modules on the desktop application side. Finally, the data management module would form the audio segments obtained into playable audio files available inside of the MATLAB GUI, where it can be visualized as a spectrogram and played.

Each module discussed contains details that help it perform its functionality. The MATLAB GUI module commands the Python back end to import the produced audio files in order to view and listen to them. The GUI allows the user to interact with the BiVo system. The data visualization primarily makes use of MATLAB's Signal Processing Toolbox in order to compute the spectrogram. Data management uses Python's wav library to create playable .wav files. Both communication modules on both ends of BiVo use basic python to create a general communication schema which the USB communication modules implement. Specifically, the desktop application's USB communication modules utilizes the PySerial library, and the sensor's USB communication module uses serial VCOM over USB. The sensor core is simply the sensor's core program. The audio analysis module uses the digital signal processing library provided for the processor: the CMSIS DSP library. Additionally, this module a Fast Fourier Transformation and filtering to identify bird vocalizations. As for the microphones module, the PDM Peripheral and on-board microphones are used to carry out the job of recording.

Our final implementation of BiVo differs from what we intended to build initially. For example, we intended for BiVo to contain a few more modules than shown in Figure 1. Time constraints and failed expectations caused the minimization of BiVo's functionality. The storage manager module, which would have utilized the FLASH storage on the Thunderboard, would have been responsible for the saving and retrieval of audio data and sensor data to device storage. This module was not possible because we anticipated that the board would be able to hold more audio than what is actually possible. As a result of being unable to store board settings and audio segments, we had to discard other modules and functionality that would have depended on these features. The data management module did not implement mode changes to the sensor (Time-Interval Mode, Triggered Mode, Power-Down Mode, Idle Mode), settings changes, and deleting and downloading files from the sensor. The import/export module on the desktop side was not implemented since we couldn't handle sensor configurations. The timers module was going to be responsible for the wav metadata inside of the recorded audio data. Although, since we can't save audio segments, this idea was discarded. Furthermore, we anticipated that the microphones module would take a few weeks. Though, this was not the case, and further drove us as a team to decide what modules just weren't feasible in the time we had left.

There was one main function we had to implement differently than we previously thought. The way the desktop application would be commanding the sensor to record had to be reworked. In the beginning we thought threading would be the answer in running the application and running the program that initiates the call to record. Although, we mistakenly thought MATLAB workers functioned the same as threads, which was not the case. Because of this, we decided to have the user press a record button on the GUI which ran the record program a number of times based on the user's selection seconds to record.

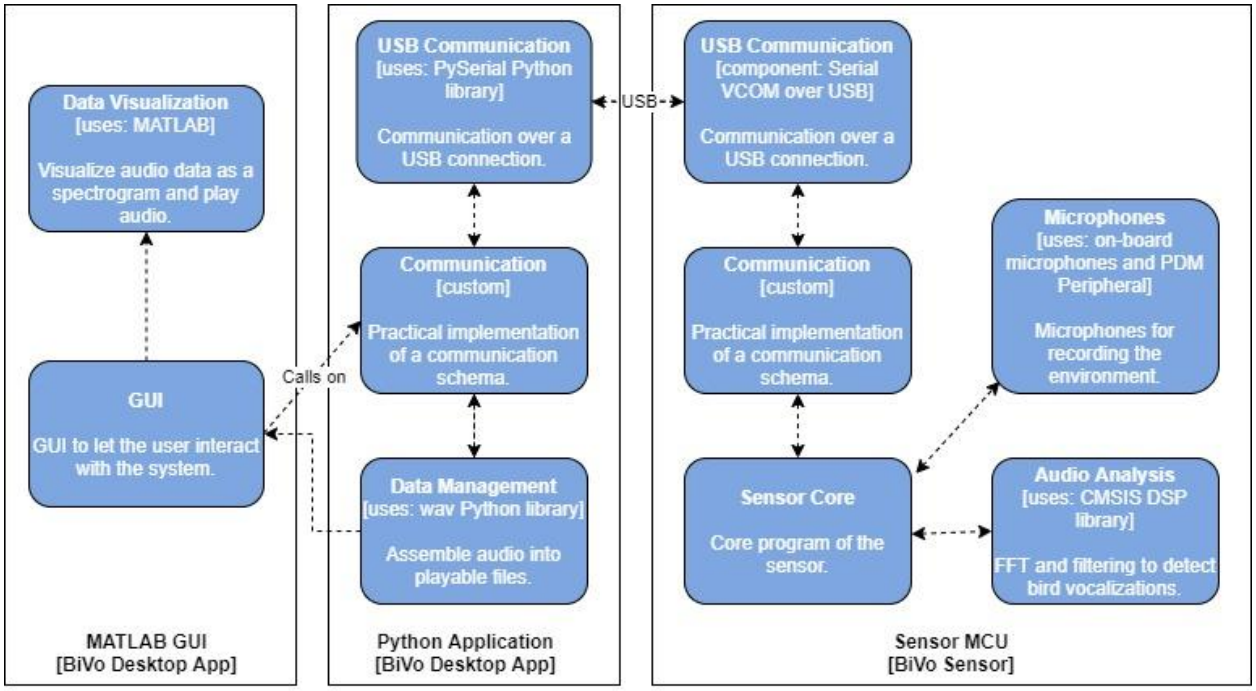

Figure 1. High level overview of BiVo's architectural implementation.

5 Testing

Our testing plan for this project was to perform unit testing and integration testing on each module itself to ensure that the code worked well without bugs. After these two types of testing, we performed usability testing on the product as a whole to verify that the product works as intended with users. We decided to do these three types of testing to ensure that our product works as expected and that users can use the product effectively without any problems arising.

To ensure that our product is well tested, we first had to do unit testing that would ensure that our lower level modules worked as expected. This involves testing each module individually, which required us to come up with ways to test code on the Thunderboard and front end separately. For the Thunderboard, we decided to write custom C scripts that helped us test and verify values gathered and sent from the Thunderboard. For the front end, we decided to use the standard "unittest" module to test the Python modules and the MATLAB application testing framework for the MATLAB GUI. These unit

tests were to ensure that our source code at the module level worked as expected.

Next, we tested how all of these modules worked together with integration testing. For this type of testing, we wanted to be sure that the data flowing between modules was working as expected. This was done by the Python "unittest" module as well as custom C scripts. We also had to do some manual tests to ensure that it was well tested to accommodate for the lack of an embedded testing framework. Most of the manual testing we did was to check if the data from the Thunderboard was accurate and being sent correctly. This was done with the help of some Python scripts that were tested previously.

Our last piece of testing was Usability Testing, which helped us find deficiencies in our product and allowed us to fix these deficiencies to create a more usable product entirely. We gathered nine participants for our usability who all partook in non-assisted testing. Each participant was assigned three use cases to complete: record an audio segment, generate a spectrogram, and play an audio file you generated from the first task. We tracked the amount of time, clicks, and any possible questions or problems that arose throughout the testing process. Usability testing helped us figure out how users would use our product and brought to our attention a couple of problems with the previous implementation. We used this data to combine the functionality of two tabs into one and change the way users browse for files.

6 Project Timeline

Our project has evolved over the course of this semester. Mainly, by having changes in development times on major modules when we learned about how long it takes to research and develop certain modules. We split up our team with Daniel and Yasmin working on the front end, and Anqi and Kevin working on the back end. We had two major milestones: our Alpha Prototype and our completion of usability testing. In these milestones, we completed and made a usable product to be tested and demonstrated.

For our Alpha Prototype, we made sure that we had a usable product in order to demonstrate the product's full functionality. We accomplished this by completing Communication on both the sensor and front end, USB Communication on the sensor, and data visualization on the front end. At this time, the microphone module was usable, but it was not how we would have liked it to be implemented. Although, this worked well for our Alpha Prototype since it fulfilled our main requirements.

For our completion of usability testing, we decided to improve some less features that could be improved upon based on some occurrences during the usability testing phase. Some of these changes were things such as how the user interface is organized, how the input was taken, and how browsing for a file worked. This allowed us to receive input in better manners that fixed a lot of the problems that were brought to our attention during the usability testing.

7 Future work

BiVo has immense potential for extending functionality. Possibly the most useful addition for making BiVo more applicable to different environments is settings to control the recording and analysis behavior. For recording, settings include changing values such as the sample rate, sample depth, and the gain, as well as the option to record with one or both microphones. For analysis, settings such as changing the range of frequencies to analyze between and changing the value of the power threshold for marking a segment as 'significant' could be very useful as well.

The audio analysis performed could also be improved on greatly. Currently, our basic audio analysis tests the power of frequencies against a threshold. This analysis takes no consideration to the time domain, which could make analysis much more accurate at marking segments, or even only marking segments that match a specific profile, such as only marking segments with robin calls. A spectrogram or wavelet analysis would be very useful here, and so would machine learning.

Many other changes we can suggest with moving forward all rely on the ability for BiVo to not be wired directly to a computer to function. Attaching BiVo to a battery or solar-powered source would allow for many more opportunities. To communicate, BiVo should incorporate some sort of wireless technology such as low-energy bluetooth. This would allow the sensor, or many sensors, to record and forward their segments to a computer without the need for constant, wired connections. Adding on to this, multiple sensors may be connected by bluetooth networking to span a large area, and connected to a central server to store the audio collected. This server could additionally forward all of its segments using cellular data, making it possible to access the data and control the sensors entirely remotely.

To increase the reliability of sensors, additional storage may be added to the sensors to allow segments to be stored locally in the event that segments cannot be forwarded. The addition of an SD card would allow the scientist user to manually retrieve audio segments when networking or when remote use is not possible.

8 Conclusion

Birds play an immensely important role for our environment and ourselves. Dr. Paul Flikkema, our client, is creating a tool to classify birds based on audio recordings but needs an extensible option for gathering such recordings. Many of the products available lack documentation or are expensive. BiVo aims to address these deficiencies by being open source and modular, and is developed on a relatively cheap and widely available development board.

While BiVo in its current form is simple, it will act as the foundation for further developments into a full-fledged and powerful sensor system to help scientists remotely monitor and record bird vocalizations. Through this version of BiVo, we have shown the feasibility to use the EFM32GG12 Thunderboard development board as a sensor to record and analyze bird vocalizations. We are confident our work will help Dr. Flikkema with his bird classification tool, and will help scientists perform better research on birds.

We have learned a lot about software development over the past two semesters. From toolchain troubles, to team member conflicts, to lack of sufficient research, we have immensely more experience now than we had before we started Capstone. We have confidence that Capstone has better prepared us for working in our future careers.

Appendix A:

Hardware:
As a team, we developed on both Mac and Windows operating systems. The specs for each teammate's computing system is listed below.

- Anqi
    - Intel i5 2 GHz 4-core
    - 16 GB
- Kevin
    - Intel i9 2.4 GHz 8-core
    - 32 GB
- Daniel
    - Intel Core i7 2.30 GHz 4-core
    - 16 GB
- Yasmin
    - Intel i7 2.80 GHz 4-core
    - 8 GB

We believe that development for this project can be done on any machine as long as it can run MATLAB app designer studio for effective development of the user interface. However, the process may take longer on a machine with less than an i5 processor and 8GBs of ram.

Toolchain:
The software tools we used for development are listed below.

- Visual Studio Code - This tool was used to develop the Python front end. It was helpful because it is a lightweight text editor that allows the use of extensions for ease of development.
- Simplicity Studio - This tool was used to develop the back-end embedded system code. It was helpful because it is the main development tool to program and flash the Thunderboard EFM32GG12.
- MATLAB - This tool was used to develop the front end User Interface. It was helpful because it was the way to help get data into MATLAB efficiently.

- Serial package - This tool was a Python pip package to help develop the Python front end to allow it to easily communicate over a USB serial connection. It helped us easily set up communication between the front end and back end.

Setup:

1. Download Simplicity Studio
    a. Plug the board into the computer so Simplicity Studio can install the SDK for the Thunderboard EFM32GG12.
    b. Download the BiVo Backend repository and import the project into Simplicity Studio.
    c.
2. Download MATLAB
    a. Install the Signal Processing Toolbox add-on within the MATLAB add-on management tool.
3. Download Version 3.8 of Python
    a. Ensure that this is your main version of Python for development with the front end. One way to do this is to create a virtual environment within the front-end directory.
4. Pip Install the Python Serial package
    a. If using a virtual environment ensure that this package is installed within that virtual environment.

Production Cycle:

To understand the production cycle, we will provide two examples of changing code, one for the front end and one for the back end. Since the back end and front end are separated you can do changes on each part separately.

1. Front-end change in code
    a. If you changed the Python code, then you will have to ensure that the updated code is in the MATLAB directory before you run the User Interface.
    b. Then, you can double click on the user interface to ensure that it works correctly.

    c. After you ensure that the program works correctly you can then make a commit and push it up to the appropriate branch.
2. Back-end change in code
    a. To ensure the changes run error free, run a build on the preferred directory within Simplicity Studio.
    b. Once that is done, plug in the Thunderboard to the computer to start flashing the program to the device.
    c. Within the directory for the project you are working on, select the binaries folder.
    d. Right click on a binary that you would like to flash onto the board and select the “Flash to device” option.
    e. Choose a program. Then, the program should be on the board to test and use.
    f. Once you have ensured that the program works as intended, make a commit and push it up to the appropriate branch.

Turkey Vultures, Flagstaff, AZ. Imlay, K. 2017.