AUTOMATED COLLECTION OF HONEY BEE HIVE DATA USING THE RASPBERRY PI

A Thesis
by
MICHAEL CRAWFORD

Submitted to the Graduate School
at Appalachian State University
in partial fulfillment of the requirements for the degree of
MASTER OF SCIENCE

August 2017
Department of Computer Science

AUTOMATED COLLECTION OF HONEY BEE HIVE DATA USING THE RASPBERRY PI

A Thesis
by
MICHAEL CRAWFORD
August 2017

APPROVED BY:

Rahman Tashakkori, Ph.D.
Chairperson, Thesis Committee

R. Mitchell Parry, Ph.D.
Member, Thesis Committee

Alice McRae, Ph.D.
Member, Thesis Committee

Rahman Tashakkori, Ph.D.
Chairperson, Department of Computer Science

Max C. Poole, Ph.D.
Dean, Cratis D. Williams School of Graduate Studies

Abstract

AUTOMATED COLLECTION OF HONEY BEE HIVE
DATA USING THE RASPBERRY PI

Michael Crawford
B.S., Appalachian State University
M.S., Appalachian State University

Chairperson: Rahman Tashakkori

In recent years beekeepers have faced significant losses to their populations of managed honey bees, a phenomenon known as Colony Collapse Disorder (CCD). Many researchers are studying CCD, attempting to determine its cause and how its effects can be mitigated. Some research efforts have focused on the analysis of bee hive audio and video recordings to better understand the behavior of bees and the health of the hive. To provide data for this research, it is important to have a means of capturing audio, video, and other sensor data, using a system that is reliable, inexpensive, and causes minimal disruption to the bees' behavior. This thesis details the design and implementation of a data collection system, known as BeeMon, which is based around the Raspberry Pi. This system automatically captures sensor data and sends it to a remote server for analysis. With the ability to operate continuously in an outdoor apiary environment, it allows for constant, near real-time data collection. The results of several years of real world operation are discussed, as well as some research that has used the data collected.

Acknowledgements

First and foremost, I would like to thank my advisor and friend, Dr. Rahman Tashakkori. Since I first met him as a prospective student, he has been a constant source of guidance, insight, encouragement, and has always embraced new challenges with optimism and enthusiasm. He provided the idea that led to the development of BeeMon, and neither the BeeMon project nor this thesis would have been successful without his help, the resources he provided, or, of course, his bee hives.

I would also like to thank the members of my committee, Dr. Mitch Parry and Dr. Alice McRae, for their input, time and assistance. They, along with the entire Department of Computer Science faculty, have been a fantastic source of knowledge and support during my work on this thesis and throughout my time at Appalachian State.

This research was made possible by funding from the Appalachian State Department of Computer Science and the Lowe's Distinguished Professor Research Fund. The National Science Foundation, through the S-STEM scholarship program, provided funding essential to the completion of my graduate studies. Eastman Chemical Company provided support and flexibility that were invaluable during my work on this thesis as a full-time employee.

Thanks also to the brain trust of Nathan Hernandez and Aleksander Ratzloff, who worked with me on this project. Nate was there from the beginning, providing insight on hardware, design, and all things blockchain. Alek joined the project slightly later, but the BeeMon application as it exists today is largely his baby. Somehow we made it through the

many hours of reading data sheets, testing hardware, and fighting with code, all to the 8 bit soundtrack of a 3D printer.

Finally, to my parents, whose support has been constant and whose belief in me has never wavered: thank you. I would have never made it this far without you. We may have taken the long road to get here, but we made in the end… and that's what counts, isn't it?

Table of Contents

List of Tables

List of Figures

Chapter 1 - Introduction

1.1 Background

Honey bees are an essential part of our environment and the world's economy. More than one-third of global food production is reliant on pollination by honey bees [1], and as of the year 2000 their value to the United States' agricultural industry was estimated to be greater than $14 billion [2]. However, the number of managed bee colonies in the U.S. steadily declined through the second half of the 20th century, and, in the early 2000s, some beekeepers began reporting large numbers of bee die-offs to unknown causes [3, 4], a phenomenon which is now known as Colony Collapse Disorder (CCD). Honey bee populations have also declined in Europe [1].

Since the discovery of CCD, much of the honey bee related research has been conducted seeking to determine the cause of the bee population's decline [3]. Some of these research projects have attempted to bring automation to the study of honey bees [5, 6, 7], but there remains much room for improvement in taking advantage of technological resources that were not available even a decade ago. Many of these projects are described in greater detail in Chapter 2.

1.2 Research Overview

At present, most beekeepers determine the health of their hives by physically visiting the hive, opening it, and visually inspecting the hive and its surroundings for problems. While this approach has worked successfully for millennia [8], the challenges of CCD

warrant a new approach. An automated bee hive monitoring system has the potential to help minimize losses in honey bee populations by enabling beekeepers to remotely assess the status of their hives. Additionally, it can avoid stress on the bees, which is a consequence of opening the hive for inspection.

The first building block necessary to create an automated hive monitoring system is a system capable of capturing data that may then be analyzed to determine the health and status of the bee hive. This system must be capable of withstanding an outdoor environment for long periods of time. Exposure to wind, rain, direct sunlight, etc., are all environmental factors that are detrimental to most electronic equipment. The components of the system located within the hive must be able to withstand humidity, attack by the bees, and being coated with propolis. The system must also be capable of recording high quality audio and video of the bee hive, and must be able to do so for long periods without requiring human interaction to retrieve or move the data to a location suitable for analysis. The ability to integrate with other types of devices, such as temperature or humidity sensors, is another desirable quality.

1.3 Thesis Organization

This thesis presents the design and implementation of such a system, known as BeeMon (Beehive Monitor), as well some of the research that has been done using data collected by the system. Also discussed are many of the alternate approaches that were explored or investigated in detail, though they were not used in the final system.

Chapter 2 includes a discussion of the history of attempts to automate bee hive research, as well as examining more recent works that have analyzed audio and video recordings of bee hives in order to extract useful information. Chapter 3 discusses the

hardware components used in the project, as well as the software tools and applications that are used. Chapter 4 details the design of the system. Section 4.2 enumerates some preliminary work that was not successful, but had a significant influence on the design of the system. Section 4.3 discusses the final design of the BeeMon system, detailing the hardware components as well as the custom application that powers the system. Chapter 5 presents the results of data collected by the system and some of the research that has been enabled by that data. Chapter 6 concludes the thesis, summarizing the system and discussing some options for future work to enhance the design.

1.4 Contributions

The BeeMon project was a collaborative effort including work by Nathan Hernandez and Aleksander Ratzloff. While this thesis describes the overall project, it does not attempt to detail every piece of work that was involved in the project's design and implementation. The author served as the overall manager for the BeeMon project, and the following list summarizes his major project contributions:

- The selection of hardware devices and software tools (Section 3.2 and Section 3.3).
- The microcontroller audio recording system (Section 4.2.2).
- The CAD design and 3D printing of the custom Raspberry Pi case (Section 4.3.2).
- The BeeMon application (Section 4.3.4) was designed by the author, and he assisted Aleksander Ratzloff in its implementation.
- The additional exploration detailed in Section 4.4.

Chapter 2 - Related Work

Many approaches have been used throughout the years to leverage technology to assist in the collection of data that can be used to study the health and behavior of individual honey bees as well as entire bee hives. Earlier efforts in the field often relied on electro-mechanical devices or analog records to obtain their information, although, without the assistance of any technology the study of bees must often fall back on tedious manual methods. For example, Seeley et al. [9] studied the decision making behavior of bees by painting tags on the bees and manually counting them as they appeared at feeders. While such studies have provided much valuable data, their ability to scale to multiple hives as well as their ability to monitor bee hives in the field is severely limited by the amount of labor required by their methods.

In the last 25 years, computer technology has been applied to the study of honey bees [10]. A large body of work has been done using computerized systems to record and/or analyze audio [11, 12, 13, 14, 15] and video [6, 7, 16, 17, 18, 19] data to gain valuable information about bees, such as the behaviors of individual bees [11, 12, 20], tracking the number of bees entering and leaving the hive [5, 10, 18, 19], or monitoring the overall health of the hive [13, 14, 15, 16]. Data collection systems may also capture additional information, such as temperature or humidity [18], that helps researchers better understand the context for the audio or video data that is their primary interest.

One of the earliest bee-counting experiments was done in 1925 by Lundie [21], who constructed a device that tracked bees entering and exiting a hive by means of sensitive pressure plates connected to electrical circuits activated by the weight of a bee walking across the plate. These plates are contained within a channel that is gated on each end, with

the gates connected to a seesaw mechanism that allows a gate to be opened only on one end at a time. When a bee enters the hive through a gate, the incoming gate is closed and the outgoing gate is opened, meaning that the next honey bee allowed to pass through that channel must be leaving the hive. This gating mechanism allowed Lundie to count the number of entering and exiting bees without direct human intervention. While this was an ambitious effort, there were numerous flaws that prevented the system from being viable for long term research. Among other issues, the device interfered with the hive operation because it significantly restricted the flow of bees into and out of the hive, and it was difficult to gain an accurate count of bee traffic due to multiple bees passing through a channel at the same time.

A more advanced bee counting system constructed by Struye et al. [10] used a system of infrared sensors to accurately count the number of bees entering and leaving a hive. The system uses 32 passages large enough for a single bee to pass through, with two infrared beams that detect movement within the passage. The direction of the bee's movement is determined by the order in which the beams are interrupted, indicating whether the bee is entering or leaving the hive. Microprocessors check the status of the infrared beams 2000 times per second, allowing for extremely accurate counts that are regularly recorded to a floppy disk for storage.

Similar efforts to count or track bees near the hive entrance have been done by Campbell et al. [5], who also forced bees to pass through tunnels, but counted their passage by using capacitance bridges. Another common approach to counting or tracking bees is the usage of radio frequency identifier (RFID) systems, as used by Decourtye et al. [22], Streit et al. [20], and Chen et al [19]. However, all of these approaches involve trade-offs that prevent

them from being ideal. Those systems that use specialized equipment at the hive entrance are often expensive and complex to set up, and may risk compromising the normal behavior of the hive. RFID based systems are not practical to use at a large scale, since they require an RFID tag to be purchased and applied to every bee that will be tracked, incurring both a high monetary cost and a prohibitive amount of manual labor. RFID tags weigh enough to be a significant encumbrance to a honey bee, particularly since they must be carried by the bee at all times. Additionally, they require the use of a RFID reader that emits strong electromagnetic (EM) waves. It is not currently known how the additional weight or the exposure to EM waves may affect the behaviors of individual bees or the hive as a whole.

Analysis of the sounds made by honey bees is another area that has seen significant research. The sounds made by bees are an important part of their communications mechanism, and are a component of the "waggle dance," as originally discovered by Von Frisch [23] in 1967. Neih [11] did not attempt to analyze recorded audio, but rather used a probe and speaker to emit sounds emulating the stop signal of the waggle dance described by Von Frisch. More recently, Okada et al. [12] found that the sounds made during the waggle dance are emitted by vibrations of the bee's wings, though their significance is unknown, since the sound is only audible to a small portion of the hive.

Much of the research related to the sounds of honey bees has focused on attempting to predict the swarming behavior of a hive. Work by Mezquida et al. [13] was done to develop an audio monitoring system that stored hourly recordings from a hive, with the goal of eventually predicting swarming events. However, the audio recordings were short (eight seconds), and relatively low quality (8 bit samples, 6.25 kHz sample rate). Ferrari et al. [14] performed similar research, recording audio with 16 bit samples at a 2 kHz sample rate.

However, their system was not intended for long term deployment in the field but was instead deployed specifically to record a series of swarming events at a research farm. One slightly different variation on this topic was done by Bencsik et al. [15]. Rather than recording audible sounds made by the bee hive during the swarming process, they embedded accelerometers into the frame of the hive and recorded the vibrations emitted during swarming. Their data was captured in a manner very similar to audio recordings, using 16 bit samples and an 8 kHz sample rate.

Research using video to track and analyze honey bee behavior is mostly limited to the past few decades, as video recording devices and storage tools became available at more accessible prices commercially. Estivill-Castro et al. [7] demonstrated one of the earlier techniques for tracking honey bees using video. Their experimental setup involved inexpensive color charge-coupled device cameras mounted in waterproof boxes along with a battery power source. These cameras were placed in an orchard, positioned to capture video of bees visiting the flowers of trees that were in bloom. Video was captured at a resolution of 320x240 pixels at 24 frames per second (FPS), with one minute of video captured at ten minute intervals. As they note, their results were limited by the quality of the video, as well as the battery power source, which prevented them from capturing more frequent recordings.

Knauer et al. [16] took advantage of digital video and computer image analysis to perform automated bee tracking. However, their area of interest specifically involved the interior of the bee hive, so their recordings were made using an infrared camera, with illumination provided by near-infrared LEDs that emit wavelengths of light invisible to the honey bees. The camera system was limited to low resolution 480x348 video, and the

researchers fit each bee in the experiment with individually numbered badges for identification. Video was stored in a long duration digital video recorder.

Within the past decade, much research has been done to apply digital image analysis to video of honey bees. To date this technique has primarily focused on tracking the movements of bees. Kimura et al. [17] captured recordings using a JVC GR-HD1 digital video camera, at a resolution of 720x480 at 29.97 FPS. Recordings were made of a bee hive frame, and a ten second video clip was analyzed using vector quantization methods in order to track the movement of bees walking across the frame, as well as identification of bees performing the waggle dance.

Campbell et al. [6] revisited the concept of counting bees as they enter and exit the hive, using digital video to capture the bees' movement and image analysis to identify bees and track their movements. Video was captured at the hive entrance using a small digital camera board mounted to the hive in a protective housing. Unlike previous bee-counting efforts, this approach had the benefit of avoiding disruption of the hive's normal behavior, and the camera mount was largely ignored by the bees after installation. Video was captured at a resolution of 640x480 at 30 FPS. By using adaptive background subtraction to detect the bees and a maximum weighted bipartite graph matching algorithm to track bee motion, they were able to count the incoming and outgoing bees with high accuracy.

The work that most directly precedes this thesis is that done by Ghadiri [18], who used wireless surveillance cameras that were either attached to the top of the bee hive, or placed in front of it. These cameras recorded digital video at a resolution of 640x480 that was transmitted wirelessly to a digital video recorder for storage. However, the recorder used SD cards as its storage medium, and the 16 GB cards available at the time could only

record 72 hours of continuous video before they had to be changed, and the data manually uploaded to a server for storage and analysis. One limitation of this video system was the difficulty of capturing the date and time of the video. To capture this information, a feature of the surveillance camera was used to place a timestamp in the corner of the video, and image processing techniques were used to extract the timestamp from each frame of video. Analysis was done by breaking the video down into individual frames and using an illumination-invariant change detection algorithm to estimate the number of bees present in an image. This project also captured local weather information such as temperature and humidity from the National Weather Service website, and attempted to correlate these parameters with the observed movements of the bees.

As this chapter has shown, previous work has demonstrated that monitoring the activity of bees in and around the bee hive, such as counting the number of bees entering and leaving the hive or monitoring the hive for signs of swarming behavior, provides information of interest to researchers. However, most monitoring systems require significant manual setup, or frequent manual intervention to retrieve data from the system. Very few research platforms have been designed for continuous unattended data collection over a long period of time. In addition, many previous efforts have been hampered by both audio and video data that is of limited quality. This thesis explores the development of a fully automated data collection system capable of capturing both audio and video data at high quality.

Chapter 3 - Methodology

3.1 Introduction

Section 3.2 introduces the major hardware components that were used in the BeeMon system. Section 3.3 discusses software development tools and libraries that were used when implementing the project's custom software, while Section 3.4 covers the third party applications integrated into the project's tooling.

3.2 Hardware

A wide variety of hardware was explored during the course of this project. While the project was always intended to be based around the Raspberry Pi, our preliminary work investigated integration with devices such as the ATtiny2313 [24] and ATmega328P [25] microcontrollers, which were used to interface with temperature and humidity sensors [26, 27], as well as peripherals such as Analog to Digital Convertors (ADCs) [28, 29], external SRAM chips [30], multiplexers [31], etcetera. Later work explored alternatives to the Raspberry Pi such as the UDOO [32]. However, these devices ultimately were not used in our final system.

The following sections detail the Raspberry Pi and the peripherals used to implement the BeeMon system.

3.2.1 Raspberry Pi

The Raspberry Pi [33] (Figure 3.1) is a family of single-board computers, approximately the size of a credit card. Developed by the non-profit Raspberry Pi Foundation with the goal of promoting computer science education in schools and developing countries, the original Raspberry Pi 1 Model B was first released in February

2012. The Raspberry Pi has proven to be extremely popular, seeing use in applications such as robotics and cluster computing.

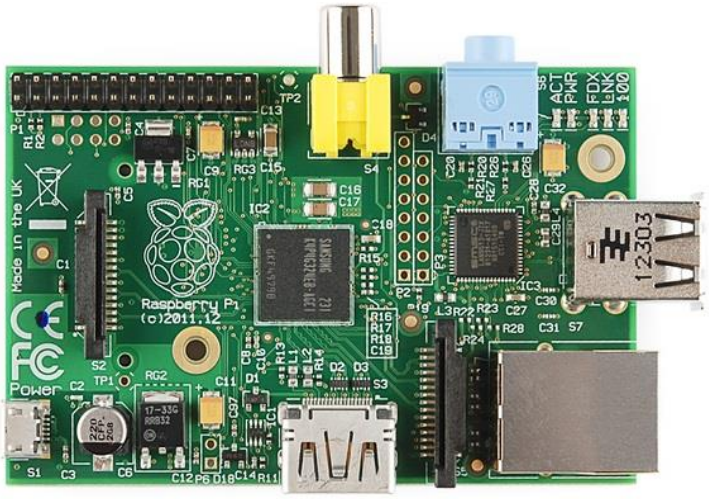

Figure 3.1: Raspberry Pi 1 Model B

Since 2012 several upgraded models have been released, however, the pricing for the "standard" Raspberry Pi model has remained constant at $35, with less powerful models available at lower price points. These models (the Model A/A+ and Zero) were not used in this project due to their lack of connectivity and limited processing power. This project primarily uses the Raspberry Pi 1 Model B, which features a Broadcom BCM2835 System-on-Chip (SoC) that integrates the CPU, RAM, GPU, and other common peripherals into a single chip. With a 700 MHz CPU and 512MB RAM, this gives the system computing power equivalent to an early smartphone, such as the iPhone 3GS. Later variants of the Model B (the Model 1 B+ and Model 2/3) are candidates for future work.

Table 3.1 provides a comparison of the main hardware features of the Raspberry Pi Model B variants currently available.

Table 3.1: Comparison of Raspberry Pi Models [34, 35, 36, 37]

Raspberry Pi	**1 Model B**	**1 Model B+**	**2 Model B**	**3 Model B**
Architecture	ARM v6	ARM v6	ARM v7	ARM v8
SoC	BCM2835	BCM2835	BCM2836	BCM2837
CPU	ARM1176JZF-S	ARM1176JZF-S	Cortex-A7	Cortex-A53
CPU Cores	1	1	4	4
CPU Frequency	700 MHz	700 MHz	900 MHz	1.2 GHz
RAM	512 MB	512 MB	1 GB	1 GB
USB Ports	2	4	4	4
Network	10/100 Mbit/s Ethernet	10/100 Mbit/s Ethernet	10/100 Mbit/s Ethernet	10/100 Mbit/s Ethernet, 802.11n Wireless, Bluetooth 4.1

Connectors for USB, Ethernet, audio output, HDMI video, and RCA video are provided on the Raspberry Pi board. In addition, it includes 17 pins for attaching peripherals, eight of which are dedicated General Purpose Input/Output (GPIO) pins, with the remaining eleven able to serve as GPIO, though they are primarily connected to controllers for the following communications protocols:

- Universal Asynchronous Receiver/Transmitter (UART)
- Serial Peripheral Interface (SPI)
- Inter-IC Sound (I^2S)
- Inter-Integrated Circuit (I^2C)

The board also includes a Camera Serial Interface (CSI) connector for attaching a color or infrared camera module.

Unlike a typical computer, the Raspberry Pi stores its data on a removable SD card rather than using a hard drive. At the time of its release, the Raspberry Pi supported standard SD cards as well as high capacity SDHC cards as large as 64 GB [38]. The board receives

power through a micro USB connector, and can be powered by any 5V power source that provides sufficient amperage.

3.2.2 Raspberry Pi Camera Module

The Raspberry Pi Camera Module is a high definition video and still photograph camera package that connects to the CSI port on the Raspberry Pi board [39]. The module includes a five megapixel, fixed-focus camera that supports still image capture at up to five megapixel (2592x1944 pixel) resolution, or video capture in the following resolutions: 1920x1080 at 30 FPS, 1280x720 at 60 FPS, or 640x480 at either 60 or 90 FPS.

3.2.3 Microphones

Our preliminary work used a preassembled Adafruit Industries microphone package combining a 20-20,000 Hz electret condenser microphone with a Maxim MAX4466 25-125x adjustable gain preamplifier [40]. Later work moved towards the use of low cost, off the shelf complete computer microphone packages with USB connectivity, such as [41].

3.3 Software Tools and Libraries

Whenever possible, existing applications were used to fill needs in the project (see Section 3.4). However, some custom software was necessary to match the requirements of our problem domain.

Development of software was done entirely using C++, primarily due to the need to access hardware at a low level. In particular, usage of the Raspberry Pi's GPIO pins to interface with microcontrollers and other peripherals was felt to necessitate the use of a lower level language. Such communication typically requires significant amounts of bit and byte level data manipulation as well as a measure of control over the timing of the program. However, it was decided early in the project that any C++ development would make use of

the latest version of the language at the time, the 2011 C++ ISO standard (C++11) [42]. C++11 adds many enhancements to the language that help with pitfalls common to traditional C++ development. For example, the `std::shared_ptr<T>` [43] class added to the standard library provides a reference counted managed pointer implementation, which removes the need to manually manage memory allocation and deallocation in many situations.

All C++ development was done on the Raspberry Pi itself. Some attempts were made to set up a cross-compiler which would allow development to be done on a Linux based desktop, and the finished application to be uploaded and run on the Pi. This approach proved sufficiently problematic that it was abandoned in favor of direct development on the Pi. This choice necessitated some limitation in the choice of C++ libraries used in the project. The limited hardware resources of the Raspberry Pi combined with the complexity of compiling and linking C++ applications meant that even relatively simple programs could take several minutes to compile. For example, early in the development process the Boost C++ Libraries [44] were added to an application, but ultimately had to be replaced because they caused build times to increase by a factor of ten.

The primary software library used during development for the Raspberry Pi was the WiringPi library. WiringPi is a set of open source libraries and tools that facilitate programmatic access to the GPIO pins and some other hardware peripherals on the Raspberry Pi [45]. Additionally, the library includes Linux command line applications to access and control GPIO pins, allowing for testing and basic prototyping using the Linux terminal or scripts. The C/C++ portion of the library is structured similarly to the Arduino software libraries, making it very accessible and convenient in a project such as this where

Raspberry Pi's and Arduino systems are used side by side. WiringPi proved to be our primary resource for interfacing with GPIO in our Raspberry Pi applications.

Additional libraries used in our development include GNU Readline, a library used to enhance the functionality of command line interfaces within an application [46], as well as TinyXML, a minimalist XML parser library allowing for easy parsing and manipulation of XML in C++ [47].

3.4 Third Party Applications

A number of third party open source Linux applications are leveraged on the Raspberry Pi to assist with data acquisition and management. These applications include:

- **ecasound**

  A Linux audio processing application [48]. When using microphones that connect via USB, each connected microphone appears as a separate sound device. Ecasound interacts with these devices to record sound to a file, allowing the configuration of many different recording parameters. However, the main limitation of the program is that it only records to uncompressed Waveform Audio File Format (WAV) files. Conversion to a more space efficient format such as MPEG Audio Layer III (MP3) must be done separately using a tool such as LAME.

- **LAME MP3 Encoder**

  LAME is an open source audio encoder that supports converting a variety of audio file formats to MP3 [49]. LAME includes numerous options to control the conversion process, determining how much space may be saved by the compression, but also how much audio quality may potentially be lost.

- **mjpg-streamer**

  Originally intended to work with webcams, mjpg-streamer is a tool that can take a series of JPG images and stream them as a Motion-JPG (M-JPG) [50]. These streams may be displayed in a web browser or video player, or saved to a file. The application uses a plugin based architecture, and the input_raspicam plugin [51] is necessary to allow it to process images from the Raspberry Pi Camera Module.

- **raspistill/raspivid**

  These applications created by the Raspberry Pi Foundation assist with the capture of still images and video using the Raspberry Pi Camera Module [52, 53]. Both programs support a variety of options to control configuration such as the image/video height, width, quality, image/video compression, output format, and etcetera.

Chapter 4 - Design

4.1 Introduction

Our system went through a number of prototypes before converging on the beehive data collection and monitoring system, hereafter referred to as BeeMon, which was deployed in the field. Section 4.2 provides an overview of some preliminary work that was unsuccessful, though it did influence our decisions when implementing the final BeeMon system. Section 4.3 details the structure of the BeeMon system. Sections 4.3.1, 4.3.2, and 4.3.3 discuss the hardware and supporting systems that make up BeeMon, while Section 4.3.4 describes the design of the custom software that integrates with the BeeMon hardware to enable data collection. Finally, Section 4.4 explores alternative systems and how they could be used to enhance BeeMon.

4.2 Preliminary Work

Our earliest efforts were driven by the desire to minimize the total cost of the data collection system. It is typical for even a small scale hobbyist beekeeper to own three to five bee hives. Small scale commercial operations may own several dozen to as many as several hundred hives, while large scale commercial operations often own thousands of hives. Our goal was to develop a system whose cost would make it viable for apiculturists at any level to deploy the system to the majority of their hives.

4.2.1 Audio Recording Formats

Audio recordings made for human consumption are typically recorded at a sample rate of 44.1 kHz, since the highest frequencies audible to the human ear are in the 20 kHz range, and the Nyquist-Shannon sampling theorem dictates that the sample rate must be greater than double the highest frequency that one wishes to capture. Existing literature

analyzing the sounds produced by bees has focused primarily on frequencies lower than 1 kHz, so it was believed likely that our system could record at a sample rate significantly lower than 44.1 kHz. However, since it was uncertain what aspects of the audio data might prove to be the most useful during later analysis, our development goal was to record audio at 44.1 kHz.

Additionally, CD quality audio is encoded using a resolution of 16 bits. That is, each of the 44,100 samples captured during a second of audio is quantized into one of 65,535 values, stored in a 16 bit integer. There exist high-resolution audio formats that use a sample size of 24 bits, however, there is some debate over whether humans can determine a difference in audio quality between these formats.

ADCs that support a 16 bit resolution are available, but models capable of a 44.1 kHz sample rate typically cost at least $5-10 per device. The ATmega328P's internal ADC is capable of a 44.1 kHz sample rate, though it is limited to a resolution of 10 bits (each sample is still stored using 16 bits of memory). Since our audio was not intended for human consumption but rather for algorithmic analysis, this was considered to be an acceptable trade-off.

4.2.2 Microcontroller Sound Recording

Initial work focused around the use of the Atmel ATmega328p microcontroller, via the Arduino Uno development board [54], as the primary interface with our sensors and data capture devices. As a microcontroller, the Arduino is ideally suited to sensor interaction. No code runs on the device except for that specified by the developer, so the processor can perform rapid operations with very precise timing.

One downside to using the ATmega328P for audio recording, however, is the very limited amount of storage available on the device. The math is straightforward: each second of audio captured at 44.1 kHz requires 88,200 bytes of storage, compared to the 2048 bytes of RAM available on the microcontroller. Realistically, the amount of RAM that can actually be dedicated to audio storage is much lower, because the RAM must also store the application's call stack and its data.

In order to overcome this limitation, our design made use of a data transfer bus consisting of two levels of data buffering, with each data level using a double buffering system. The first buffer level lies within the onboard RAM of the ATmega processor, as a pair of 512 byte buffers. The second buffering level is implemented using a pair of 256 kB SRAM chips [30] that are connected to both the ATmega328P and to its controlling Raspberry Pi. A system of multiplexers [31] and control lines provide synchronization and allows the Raspberry Pi to interact with one SRAM chip using SPI, while the microcontroller interacts with the second chip. Figure 4.1 shows the wiring diagram for this bus.

Each first level buffer on the ATmega is capable of holding 32 samples, or approximately 725 μs of audio data. When a first level buffer is filled, the buffers are swapped. The program then proceeds to write audio samples to the second buffer, while simultaneously transmitting data from the first buffer to the external SRAM chips of the second buffering level. With 256 kB of storage available, each SRAM chip can hold up to 131,072 samples, or approximately 2.97 s of audio data. As a SRAM chip in the second buffer level is filled to capacity, the SRAM chips are swapped so that the Raspberry Pi may begin reading data from the chip that is filled, while the ATmega pushes its filled internal buffers to the second, now empty SRAM chip. As the Raspberry Pi reads a chunk of data

from a SRAM buffer, it will append that data to its existing audio data, writing the data as audio files at an interval determined by the desired file size or recording length.

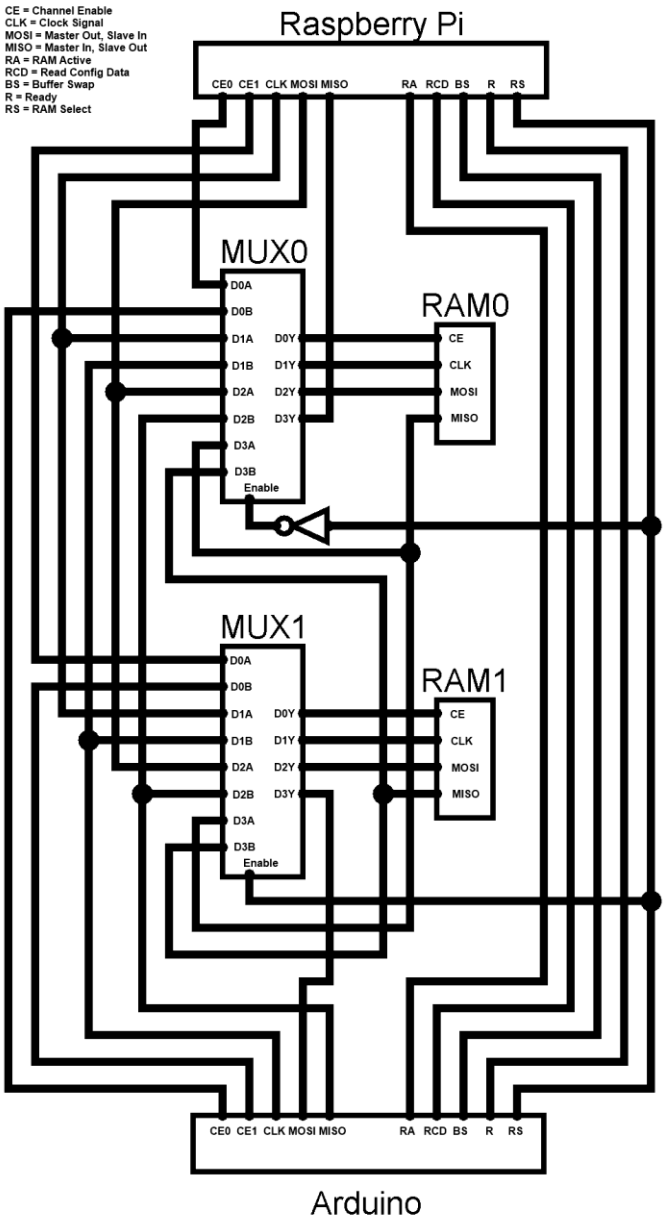

Figure 4.1: Arduino - Raspberry Pi data transfer bus

This system would also be capable of supporting a limited number of additional sensors. Most additional sensors of interest to us, such as temperature and humidity require very infrequent readings in comparison to audio recordings. The DHT22 sensor, for example, can only provide readings at a maximum rate of once every two seconds. This

makes these sensor readings relatively easy to schedule alongside the audio recording, and only a few bytes in the SRAM buffer need to be devoted to holding their most recent readings.

Unfortunately, testing revealed that while this design appeared conceptually sound, it contained some practical flaws. The SPI protocol uses a four wire serial bus to transfer data between the master and slave devices:

- Serial Clock (SCLK) carries a clock signal from the master to synchronize the transmission.
- Master Output, Slave Input (MOSI) transfers data from the master to the slave.
- Master Input, Slave Output (MISO) transfers data from slave to master.
- Slave Select (SS) does not directly carry data, but is used to signal a slave device that it is active on the bus.

In order to achieve the data transmission rates necessary for audio recordings, as detailed above, the SPI clock must be driven at a relatively high clock rate. The minimum speed needed for the bus is roughly 100 kB/s: 88,200 bytes of audio data, plus a ~10% allowance for overhead and additional data. Achieving this transfer rate requires a SPI clock speed of > 700 kHz. However, the nature of the buffering system in our design requires higher transfer rates. Our system does not continuously stream data to and from the SRAM chips, but rather sends the data in bursts as a buffer fills, necessitating that these bursts occur at a higher speed. Our design anticipated the need for a target transfer rate of 256 kB/s (~2 MHz clock speed), capable of reading or writing the entire contents of the SRAM chip in one second.

These clock speeds are theoretically possible. For example, the ATmega328p drives its SPI clock by applying a divider to the main processor's clock source. With a maximum supported processor clock speed of 20 MHz, and a minimum divider of two, SPI clock speeds of up to 10 MHz are obtainable. Similarly, the Raspberry Pi's SPI controller supports clock speeds of up to 32 MHz. However, the primary limitation encountered in our testing lay within the system's wiring. Our prototype system could support SPI clock rates no higher than 1-200 kHz before data transfers to the SRAM chips became badly corrupted. Some testing was attempted using shorter, more carefully placed wires, rather than the long precut wire segments pictured in Figure 4.2, but no significant increase in SPI clock speeds was obtained. Ultimately, we concluded that this system would only be viable if the hardware was mounted to a circuit board utilizing properly designed and routed traces to connect the components. Such a design could not be achieved within the budgetary and time constraints of this project, but it may be worth revisiting in the future.

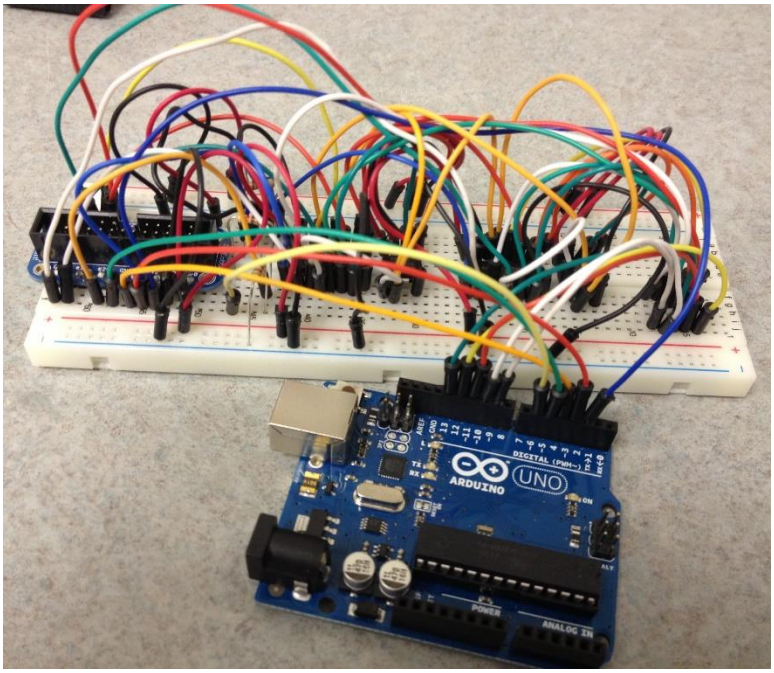

Figure 4.2: Arduino with bus prototype

4.2.3 Linux Timing Limitations

A recurring theme throughout our design is the need to write code that operates with precise timing, in order to interact with sensors and other low level hardware devices. Linux provides the `nanosleep` [55] system call, which theoretically allows applications to sleep for as little as 1 ns. However, the actual minimum sleep time varies depending on the capabilities of the underlying hardware and the settings used to build the Linux kernel.

The primary operating system of the Raspberry Pi is Raspbian [56], a variant of the Debian Linux operating system developed by the Raspberry Pi Foundation that is optimized for the ARM processor on the Raspberry Pi. Testing determined that the minimum possible sleep time in Raspbian was on the order of 150 μs, with 1 ns sleep requests often resulting a latency of 200-250 μs. While adequate for many applications, this level of latency was much too large for some of our sensors. For example, the DHT22 uses a data transmission protocol that sends a binary number by using the length of the signal to distinguish between binary digits [57]. Signals are transmitted at 50 μs intervals, with a signal lasting 26-28 μs indicating "0," while a 70 μs signal indicates "1." Clearly, the Raspberry Pi lacks the ability to read this protocol when `nanosleep` is used.

One possible solution to this issue is busy waiting, which is used by some libraries such as WiringPi. A `nanosleep` system call makes a request to the operating system kernel to put the calling process to sleep, and awaken it after the requested time interval has elapsed. By contrast, busy waiting uses a loop within the process itself, continually checking the system's high resolution clock to determine when the desired amount of time has passed. However, this approach has a significant disadvantage; it requires the process to fully utilize the CPU while waiting. The Raspberry Pi 1 Model B used in our research includes a single

core, single threaded CPU. A preemptive multitasking operating system such as Linux provides the illusion of multiple processes running simultaneously on this type of CPU by rapidly switching between processes, allowing each to use the CPU for a short amount of time. If a single process attempts to fully utilize the CPU, such as while busy waiting, it severely restricts the ability of the operating system to multitask. We knew that our system would need to rely heavily on multitasking, therefore busy waiting was not an acceptable solution.

An alternative approach used by some in the Raspberry Pi community is the installation of a custom Linux kernel that includes patches developed by the Real-Time Linux community [58]. By default, the Linux kernel provides a limited list of scenarios in which one process may preempt another, i.e., return from `nanosleep` to its calling process [59]. This means that even if the system is idle and the sleep time requested by `nanosleep` has elapsed, the kernel may not immediately return to the calling process if a preemption scenario has not occurred. The `PREEMP_RT` kernel patch developed by the Real-Time Linux community expands the set of possible preemption scenarios, as well as other changes that allow Linux to behave more like a real-time operating system.

Our testing found that the real-time patches did improve the minimum sleep time to approximately 100 μs. However, the patched kernel proved to be extremely unstable, and 100 μs was still not sufficient to interact with all of our sensors. As a result, our designs for these sensors focus on using microcontrollers to collect sensor data, which is later transferred to the Raspberry Pi. In this structure, the microcontroller can handle all time sensitive operations, while data transfer to the Raspberry Pi is handled via standard protocols such as UART or SPI. The Raspberry Pi includes dedicated hardware controllers for these

communications protocols (see Section 3.2.1), meaning that these operations are not handled by the main CPU or operating system, and are not subject to the timing issues discussed in this section.

4.3 BeeMon Implementation

The resulting BeeMon system for this thesis can be summarized as a set of hardware devices that capture data of interest from a bee hive and upload that data to a remote storage location for later processing. Figure 4.3 summarizes the flow of data: BeeMon systems are deployed to multiple hives, they collect data from those hives, and that data is uploaded via the internet to a file server. The following sections detail the design and implementation of this system.

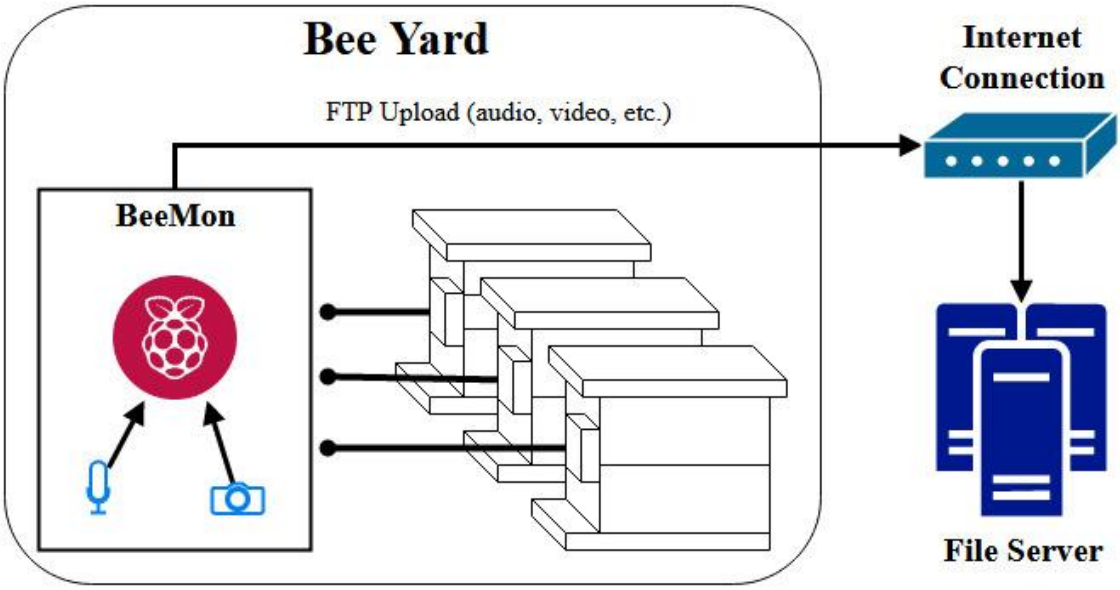

Figure 4.3: BeeMon data collection and transfer

4.3.1 Hardware

As discussed in Section 3.2, the BeeMon system that was deployed to the field consists of a Raspberry Pi 1 Model B, a Raspberry Pi Camera Module, and two USB microphones. The Raspberry Pi runs the BeeMon application that is responsible for both

audio and video data collection (see Section 4.3.4), serves as a temporary data storage device, and uploads the recorded data to a remote server for archiving and analysis.

Our testing found that it is important to use a high quality SD card with the Raspberry Pi when running BeeMon. While most SD cards are very slow [60] compared to mechanical hard drives or solid state drives, the most common Class 2 and Class 4 SD cards are simply too slow to support the needs of the BeeMon system. At a minimum, BeeMon must perform the recording of HD video and two CD quality audio streams, as well the normal, unavoidable, disk operations of a Linux operating system. A Class 10 SD card should always be used to ensure that disk performance does not cause a bottleneck in the data collection systems.

The Raspberry Pi is mounted to the bee hive using a custom 3D printed case, as detailed in Section 4.3.2. Section 4.3.3 discusses the strategies used to mount the microphones and camera to the hive for data collection. In order to avoid unnecessary complications in our system, we chose to deploy only to bee hives that were "on the grid," where the hive was already located near a building that could supply power and internet connectivity via extension cords and Ethernet cables.

4.3.2 Raspberry Pi Case

The standard Raspberry Pi is only available as a bare circuit board (see Figure 3.1). While desirable for its low price, and convenient for rapid prototyping and testing, this configuration is not suitable for deployment in real-world outdoor conditions where the board will be exposed to moisture, weather, and other hazards. A wide variety of cases for the Raspberry Pi are available from the Raspberry Pi Foundation as well as numerous third party vendors. However, at the time we were preparing to deploy the BeeMon system, we were

unable to find a case suitable for outdoor use; most cases were either decorative or meant to mount the Raspberry Pi indoors. Our initial deployments of the system secured the device in the top of the hive, as shown in Figure 4.4, where it was protected from weather and would not be attacked by the bees, but it was clear that a better long term solution was needed.

Figure 4.4: An early BeeMon prototype with audio and temperature sensor

Eventually it was decided that a custom 3D printed case was the best solution, since this allowed for the creation of a custom design that could address three areas of concern: securely mounting the Raspberry Pi to the bee hive, securely mounting the Raspberry Pi Camera Module, and protecting both devices from weather, primarily moisture. Design of the case was done using PTC's Creo Parametric 3D Computer Aided Design (CAD) software

[61], and it was printed on a MakerGear M2 3D printer [62] using Polylactic Acid (PLA) plastic.

Thanks to the rapid prototyping allowed by 3D printing, the case evolved through numerous iterations before converging into the final design comprising six parts that are assembled into four main components:

1. **Case Body**

   The body of the case, shown in Figure 4.5, contains the Raspberry Pi circuit board, fully enclosing it except for the connectors needed for our system (power, SD card, USB, Ethernet). Mounting rails are formed into the sides of the case, allowing the unit to be screwed securely onto the side of a bee hive and providing some ability to adjust its height and angle. Additionally, two cable management hooks are included on the case's left side, allowing the Ethernet cable to be routed along the side of the case to ensure that it does not fall into the Camera Module's field of view.

Figure 4.5: Raspberry Pi case, lid removed

2. **Case Lid**

The case's lid, visible in Figure 4.6, fits snugly over the top of the body, with a lip overhanging the body to ensure that water cannot easily penetrate the lid. Hinges are built into the rear of the lid, which pivot on pins extending from the case body. Likewise, two snaps at the front of the lid snap onto pins integrated in the case body to hold the lid securely closed. The lid also features a small slot allowing the Raspberry Pi Camera Module's ribbon cable to exit. This slot is angled so that, when the case is mounted vertically, water will be unable to enter the opening. The case lid also incorporates a mount for the Camera Module that is printed as a separate piece and permanently attached to the lid using epoxy. Two rails on the camera mount enable the camera case to be attached with a pair of screws, and the camera's distance from the side of the bee hive to be adjusted.

3. **Camera Case**

A camera case and case back plate are printed separately for the Camera Module. These pieces snap together, and are held tightly closed by friction. When the Raspberry Pi is mounted to a bee hive, the camera case is the component farthest from the side of the hive, and mostly likely to be exposed to wind-driven rain or other weather. To minimize the chances of moisture entering the camera case, the back plate is a single piece, and the entrance for the Camera Module's ribbon cable is tightly sealed when the camera case is assembled. The only exposed opening in the case is the hole allowing the camera lens to see out; however, this hole is always facing downward and presents little risk of moisture invasion.

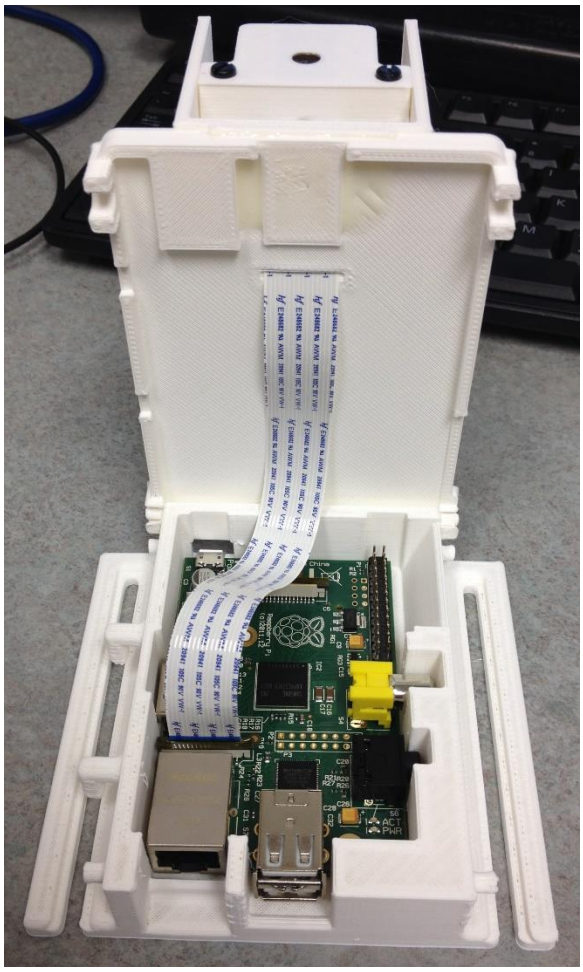

Figure 4.6: Raspberry Pi case, lid open

4. **Rain Shield**

The design of the Raspberry Pi makes it difficult to create a fully sealed or enclosed case, since using the device effectively requires that the power cable, Ethernet cable, SD card, and multiple USB devices be connected, all of which protrude significantly from the unit's main board. To provide additional protection for these openings and the device as a whole, our case incorporates a detachable rain shield, shown in Figure 4.7. Two arms attach the shield to the case body, and a brace ensures that it remains vertically oriented to provide effective protection.

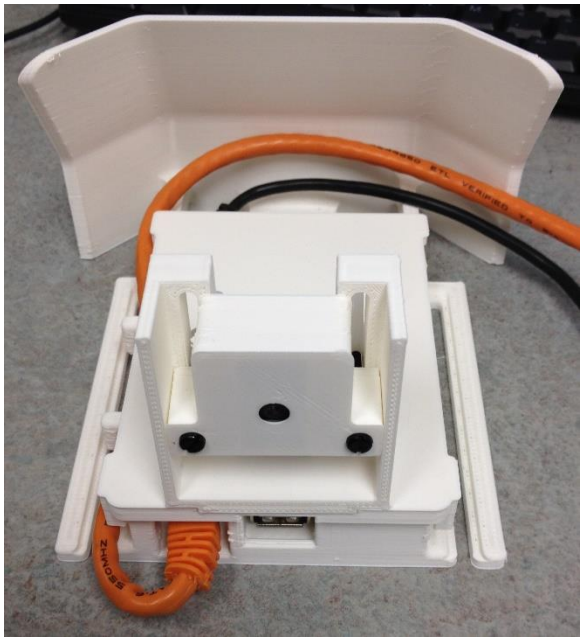

Figure 4.7: Raspberry Pi case, fully assembled

The case was split into six pieces primarily to facilitate printing with a 3D printer, allowing all of the pieces to be printed simultaneously in a single print job requiring approximately five hours to complete. This approach also reduces the amount of material needed to support the parts during printing, material which is discarded as waste when printing is complete. All parts are printed with an 80% infill setting, providing a balance between part strength and unnecessary material usage.

The use of PLA plastic for the case material provides several advantages. PLA is both one of the cheapest 3D printing materials available, and one of the easiest to work with. It is also an environmentally friendly bioplastic [63] made from plant based renewable resources, and it is biodegradable; breaking down after extended exposure to the elements. This led to some concerns about the durability of PLA, since our cases would need to survive outdoors for months at a time. These concerns proved to be unfounded, however, as our cases showed no significant degradation after being deployed for a full season of bee hive

activity. Thanks to the low cost of PLA, the material cost of a complete case is approximately $5.

4.3.3 Sensor Installation

The deployed BeeMon system collects only audio and video data. Audio recording is handled by a pair of microphones mounted on opposite sides of the bee hive, allowing comparison of stereo audio tracks.

To determine the optimal mounting locations for the two microphones, a series of tests were performed using an unoccupied hive in a quiet environment. Three pairs of holes were drilled through the sides of the hive body at regular intervals, and another pair of holes were drilled centered on the ends of the hive body. These holes were then filled using temporary plugs, except for the pair of holes being tested. A recording of a bee hive was played through an omnidirectional speaker mounted in the center of the hive body, and test recordings were made of the pairs of microphones placed in different combinations of the eight holes. Afterwards, the audio recordings were compared to determine if there were any noticeable differences in the audio quality, volume, etc. of the various locations.

No significant difference was found between the mounting locations, so centered mounting holes on the sides of the body were used for simplicity. All bee hives used were Langstroth hives with 10 frame deep bodies. The exterior dimensions of these bodies are 19.875 in. x 9.25 in. x 9.5 in. [64], placing our microphones approximately 9.94 in. from the front and rear faces of the body, and 4.75 in. from the top and bottom. The holes were approximately 0.5 in. in diameter, to allow the microphones to move freely in the hole with minimal tension in order to reduce the risk of damaging wiring as they are inserted and removed.

Bees tend to attack anything that they perceive as a foreign object in the hive, and will also cover gaps or openings with a type of resin known as propolis [65], as shown in Figure 4.8. In order to reduce the bees' ability to damage the microphones and minimize the area that they could fill with propolis, the interior opening of the microphone hole was covered with a fine mesh, the microphone was inserted flush against the mesh, and a plug was inserted into the hole's exterior opening to hold the microphone in place and prevent weather or foreign insects from entering the hive. Figure 4.9 shows a microphone installed using this method. It was found in our deployed systems that the bees would still attack the microphone through its protective mesh and attempt to apply some propolis to it. However, the microphones were sufficiently protected to allow them to survive through most of the bees' active season. Because the USB microphones used in the deployed system were inexpensive, this was considered an acceptable lifetime.

Figure 4.8: Microphone, partially clogged with propolis

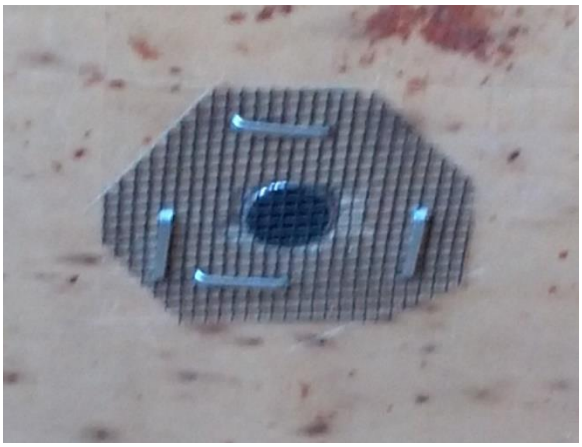

Figure 4.9: Preferred microphone installation

The microphone installation described above is the preferred style, but it requires that the BeeMon system be set up before the hive is populated, since drilling the necessary mounting holes and attaching protective wire mesh would cause significant disruption to an active hive. In situations where microphones needed to be added to an active hive the microphone was suspended by its wiring as shown in Figure 4.10, and secured in place in the approximate position where the normal mounting holes would be drilled. This approach does expose the microphone to more attacks by the bees, since it is an obvious foreign object within the hive.

Figure 4.10: Temporary microphone installation

The Raspberry Pi Camera Module is mounted to the Raspberry Pi's case, as detailed in Section 4.3.2. The case is always mounted above the hive entrance with the camera facing down, though the exact location varies depending on the configuration of the bee hive. The case's mounting rails allow for some adjustment of the camera's angle in the plane of the hive's front face, though significant changes may require the mounting location to be changed. The camera is contained in its own case that mounts to the main Raspberry Pi case on rails, allowing the camera's distance from the front face of the hive to be adjusted. Figure 4.11 and Figure 4.12 show front and rear views of the camera mounting system.

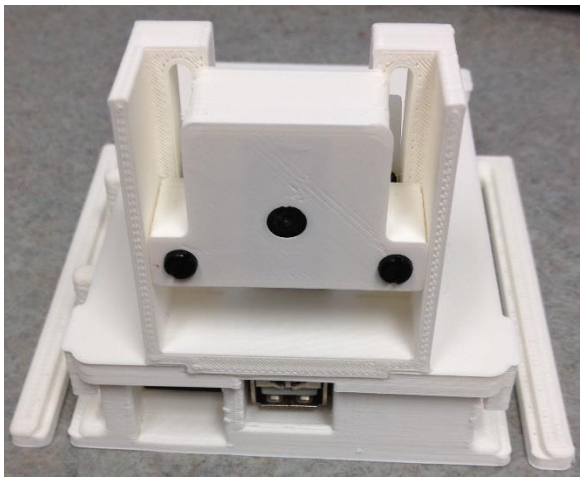

Figure 4.11: Raspberry Pi case, camera mount (front)

One issue encountered with the Raspberry Pi Camera Module is its fixed focus, which is specified as being able to focus at a minimum distance of one meter. Since a typical bee hive has a total height of approximately one meter, and the Raspberry Pi + Camera Module package is mounted to the side of the hive, the camera must be closer to the entrance. After some investigation, we determined that the lens of the camera module was held in place with glue. This glue can be carefully cut using a small knife, such as an X-ACTO type hobby knife. Once freed, the camera lens can be adjusted, using a pencil eraser as a tool, to achieve the desired focus distance, after which it is glued in place again to ensure that its focus does

not change. While this adjustment does limit the camera's ability to focus on more distant objects, this was not a concern for our application.

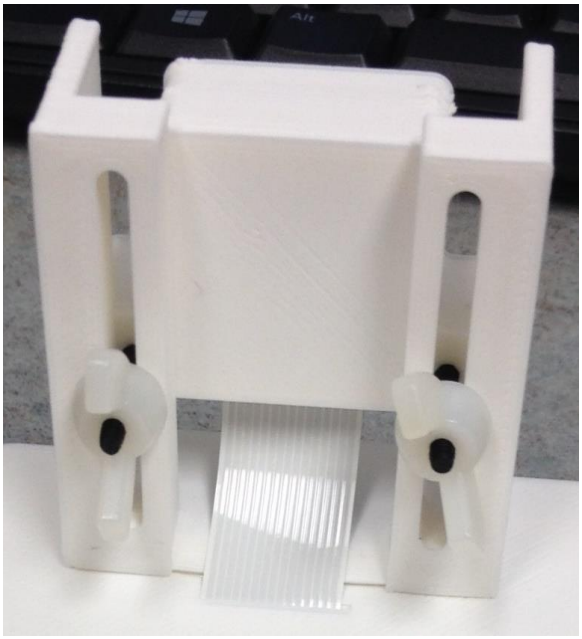

Figure 4.12: Raspberry Pi case, camera mount (rear)

4.3.4 Application

The BeeMon application is a multi-tasking, multi-threaded, command-line C++ application used to control the BeeMon system. At a high level, it initiates the recording of audio and video data, manages the storage of the recorded data on the Raspberry Pi, and uploads data to a remote server for storage and analysis. The application also maintains a detailed log of its activities to facilitate in diagnostics and troubleshooting of problems, as well as status monitoring. To prevent logging from having an impact on time sensitive recording operations, log messages are passed to the log writer using a thread-safe, multi-reader multi-writer queue, and the output file is written to asynchronously using a dedicated logging thread.

Each subsystem within BeeMon runs independently in its own thread to reduce the chances for one subsystem to affect timing of another subsystem. Whenever possible, actions are performed by invoking existing third party programs rather than implementing custom functionality. When such a program is called, it is run as a child process of BeeMon so that its status can be monitored and any output or error messages directed into BeeMon's logs. Unless otherwise noted, the applications referenced in this section are detailed in Section 3.4.

The core of the BeeMon application is its command-line terminal interface, which runs on the application's main thread. After the application startup is complete this interface waits for user input and attempts to match any input to a set of known commands. If matched, the command is executed allowing the terminal to control the behavior of any aspect of the system. The terminal also supports a set of features to enhance its usability, including a history of recently entered commands, a listing of the available commands, and help text for each available command. Since the Raspberry Pis used in the BeeMon system are typically configured in a headless setup with no monitor or keyboard attached, the application is usually started and controlled using SSH. To ensure that the application continues running after the SSH session ends, it is necessary to run BeeMon from within a terminal manager program such as tmux [66] or GNU Screen [67]. These programs allow BeeMon to be started within a terminal session, to be detached from the current SSH session, and to remain active even after the SSH session has been terminated.

In order to reduce the amount of manual configuration information that must be entered when using the application, BeeMon supports the use of an XML configuration file containing predefined settings. Essentially every setting available in the application can be

included in this file. For example, the configuration file may include settings indicating that audio recording should be performed from 5:00 A.M. to 9:00 P.M., and all audio recordings should be uploaded in a daily batch at 1:00 A.M. This allows audio recording to be started with a single short command, rather than requiring all of the recording options to be specified as part of the command. Configuration settings specified in the file may be overridden via the command line arguments provided when BeeMon is started, or by supplying them when invoking the relevant command. Some settings are required and must be provided in either the configuration file or the application command line arguments. Most settings, however, have a built-in default value that is used if the setting is not otherwise specified.

The BeeMon audio recording subsystem allows for recording audio from two microphones, in order to capture sound from both sides of the hive. Two instances of the ecasound application are triggered to perform the recording task. Recordings are always captured as WAV files, using 16 bit samples and a 44.1 kHz sample rate. Additional parameters specify the duration of the recording (typically one minute), the microphone to record (left or right), and the output path for the recording file. In order for the application to address microphones attached to the Raspberry Pi as "left" or "right," an asoundrc configuration file is installed along with BeeMon, configuring the USB microphones to be addressed using those aliases.

The duration of each set of recordings is configurable, along with the output directory used for recordings. If desired, recordings may also be limited to only the left or right channel using configuration options. BeeMon also supports several audio capture modes:

- Continuous recording, where a new set of recordings begins immediately after the current set of recordings are completed.

- Interval recording, where there is a configurable time delay between each set of recordings.
- Scheduled recordings, where the system records daily between a designated start time and end time. Interval settings may also be used with this mode.

Audio recordings may either be uploaded as soon as they are complete, or uploaded in a daily batch at a time designated in the configuration.

The BeeMon video recording subsystem makes use of the Raspberry Pi Camera Module to capture video of the bee hive. While our primary interest was in the more common version of the Camera Module that captures visible light video, this entire subsystem is fully compatible with the infrared version of the Camera Module, if it is installed. Video capture offers options very similar to audio, although video recording is always continuous when enabled. The raspivid application is used to capture the recording, and it is passed options specifying the video's width and height, in pixels, its framerate, duration, and output file name. Video is always encoded using the H264 format. Like audio, video recording supports options to record only during specific time intervals, and to limit uploads to a daily batch. All of these options may be changed in the application's configuration file or when issuing the start video recording command.

The upload subsystem of BeeMon is shared between the other data collection subsystems. This system is based around the use of File Transfer Protocol (FTP) to upload data files. The BeeMon configuration includes settings that are common to all data areas, such as the FTP server host, server credentials, and a base directory that will be prepended to the upload directories for audio, video, and other data. This allows for flexibility in the server's directory layout. For example, each BeeMon system could be assigned its own

directory on the server, with its data uploaded to audio, video, etc. subdirectories. Alternatively, the server could contain top level directories for audio, video, etc., with each BeeMon system assigned a subdirectory in the respective data areas.

4.4 Additional Exploration

The UDOO [32] was explored as a possible alternative to the Raspberry Pi as the primary platform for BeeMon. It was hoped that the powerful features of the UDOO such as quad core CPU, integrated microcontroller, and SATA hard drive support would allow the UDOO to collect data from multiple bee hives simultaneously, reducing the overall cost of the system. Our intention was to use the UDOO's powerful microcontroller to manage many microphones, similar to the system described in Section 4.2.2. However, testing determined that while the CPU and the microcontroller of the UDOO are directly connected, the only communications channels connected on the device's circuit board are those for the UART serial interface which, on those devices, does not provide sufficient bandwidth to support the transfer of audio data. Without the ability to capture audio from many hives simultaneously, the UDOO offered few advantages over the Raspberry Pi, at a much higher cost: approximately four Raspberry Pi's could be purchased for the price of a single UDOO.

An alternative use considered for the UDOO was a role as an on-site server, effectively a data collection middleman. Our test bee hives were located in a residential area, and the internet connection used for BeeMon's data uploads was a residential cable internet service. Internet Service Providers typically limit residential plans to relatively low upload bandwidths, directing customers who need more upload bandwidth to purchase much more expensive commercial plans. The limited upload bandwidth did not directly impact the performance of our system, however, continuous uploading by the BeeMon systems did

consume enough bandwidth to noticeably impact internet performance for the home's residents.

The UDOO can store large amounts of data when configured with an external SATA hard drive, allowing it to act as a low cost server and upload target for the local BeeMon systems. With a sufficiently large hard drive, weeks or even months of data can be recorded, removing pressure from home internet connections or allowing BeeMon systems to be deployed to locations where power is available, but internet service is limited or non-existent. Additionally, the multi-core CPU of the UDOO is powerful enough to encode WAV audio files to MP3 as they are received, significantly increasing the amount of data that can be stored.

The UDOO investigations discussed above were done after the initial BeeMon design had been completed and was deployed in the field. While the "UDOO as a server" option offers some attractive benefits, by the time it could be investigated in detail the batch upload features were added to the BeeMon application, and performing nightly uploads proved sufficient to minimize the effect on the residential internet service.

Chapter 5 - Results

5.1 Data Collection

Initial deployments of the BeeMon system took place in the spring of 2014, as shown in Figure 5.1. The first two bee hives to be monitored recorded only audio, because at that time there was no solution available for securely mounting the Raspberry Pi Camera Module to allow video recording of the hive entrance. During the summer of 2014 the Raspberry Pi case (see Section 4.3.2) was designed, and video recordings began to be captured. By the fall of 2014, the case had been completed and the system as described in Section 4.3 was deployed in the field collecting data. Figure 5.2 shows the completed system.

Figure 5.1: A BeeMon prototype with audio and video recording

Figure 5.2: A completed BeeMon system

Recordings continued throughout the fall of 2014, until the bees' activity declined due to cold weather. Data collection resumed in the spring of 2015, with BeeMon systems deployed to three bee hives. Additional deployments have been done in 2016 and 2017, however, by that time the system had been modified and upgraded beyond the initial BeeMon system described in this thesis.

As of this writing, more than five terabytes of combined audio and video data have been captured by various BeeMon installations deployed to real-world bee hives. Some research has already been completed using data collected by the BeeMon system. Several additional research projects are currently underway at Appalachian State University, using both the audio and video data to gain insight into the health and behavior of honey bees and to work towards the eventual goal of implementing a fully automated bee hive monitoring

system. Figure 5.3 displays an audio sample from a BeeMon audio recording. Figure 5.4 shows a frame taken from a BeeMon video recording.

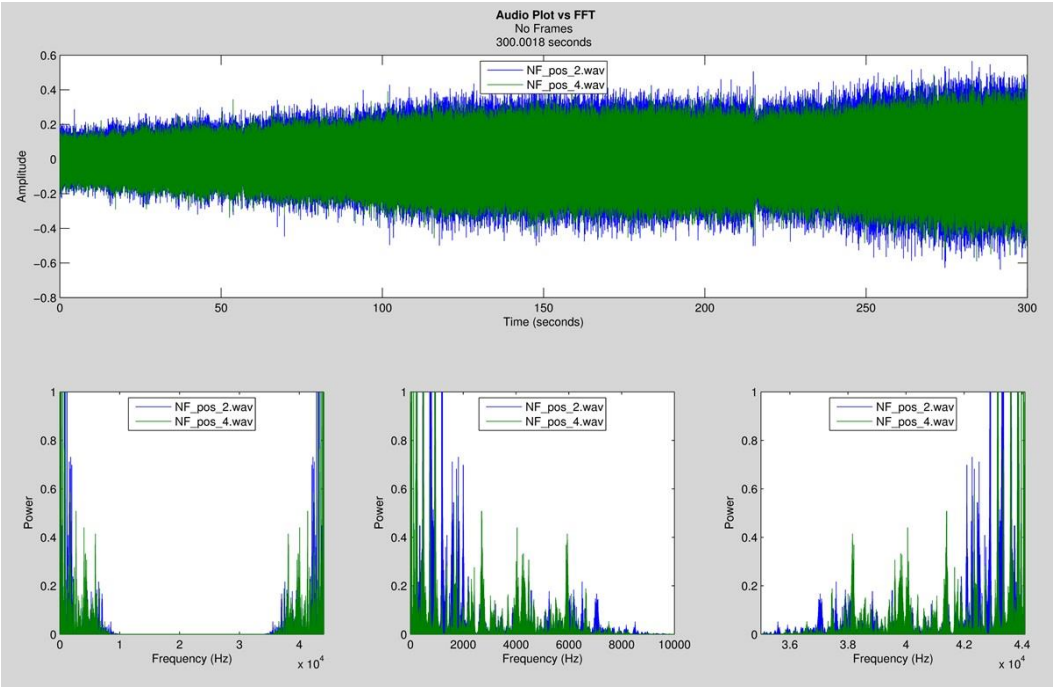

Figure 5.3: Sample BeeMon audio recording

Figure 5.4: Sample frame from a BeeMon video recording

5.2 System Cost

One of the primary goals of this project was the development of a system that was inexpensive, allowing it to be deployed in large quantities. Table 5.1 lists the cost of the components used in the final BeeMon system, as discussed in Section 3.2 and Section 4.3.1. While the cost is low considering the audio and video recording capabilities of the system, it is expensive enough to be prohibitive when considering the deployment of dozens or hundreds of systems.

A significant factor contributing to the price of the BeeMon system is the goal of capturing high quality audio and video. This decision was made due to uncertainty regarding the quality of data needed to be viable for data analysis. After sufficient analysis has been done to determine with some degree of certainty what level of data quality is required, it will be possible to revisit the BeeMon system and update it to use lower cost components that provide data of adequate quality.

Table 5.1: BeeMon Component Cost

Component	**Cost**
Raspberry Pi 1 Model B	$35
Power Supply for Raspberry Pi	$7
16 GB Class 10 SD Card	$9
Raspberry Pi Camera Module	$29
2x USB Microphones	$16
Raspberry Pi Case (PLA material cost)	$5
Total	**$101**

5.3 Computer Vision Analysis

One of the initial research projects to use BeeMon data is a thesis by Kale [68], who applies computer vision object detection and motion tracking to videos recorded by BeeMon. Using these techniques, he is able to count the number of bees visible near the hive entrance and track bees to determine if they are entering or leaving the hive. As one minute long videos are uploaded to the file server by BeeMon, they are downloaded automatically by his application for processing.

For each frame in the video, the frame image is converted to grayscale and background subtraction is applied using a Gaussian mixture model (GMM). Since the video is split into one minute clips, this model is initialized using the first 100 frames of the clip, after which the video is reset to the first frame, and actual processing begins. The adaptive nature of the GMM was shown to be ideal for processing video taken in an outdoor environment where lighting conditions are often imperfect. It is also able to handle periodic changes in the background such as grass or leaves swaying in the wind. However, the performance of background subtraction can become inconsistent when the entrance of the hive is crowded with bees, due to the GMM adapting to consider clusters of bees as part of the background. Application of background subtraction produces a binary foreground mask, as shown in

Next, Kale compares two methods for detecting bees in the image. Blob analysis detects clusters (or blobs) of foreground pixels that correspond to areas of the image occupied by moving objects, which can then be used to determine a border around the object. Alternatively, cascade classification is a machine learning algorithm that can detect objects

regardless of their environment, without requiring knowledge of the image background. Figure 5.6 and Figure 5.7 show examples of the two methods applied to a video frame.

Figure 5.5: Video frame and foreground mask

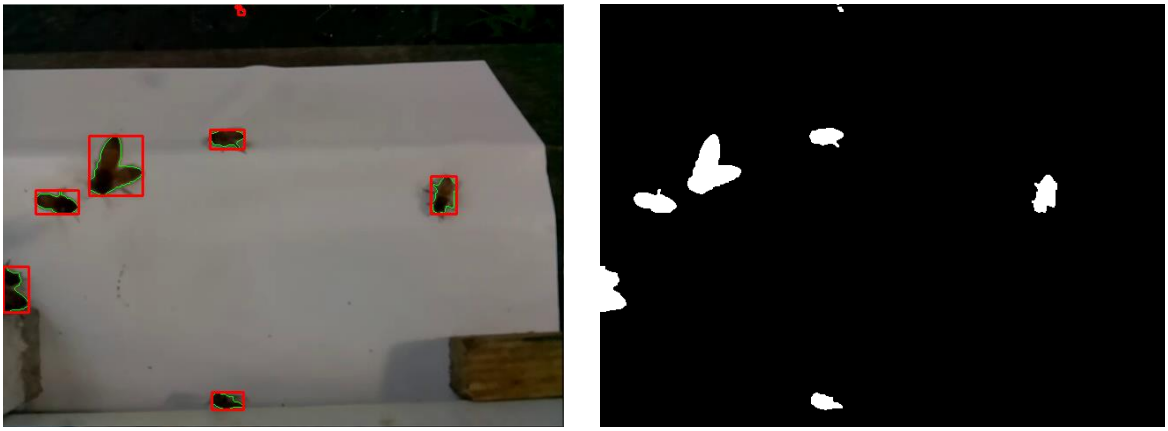

Figure 5.6: Video frame with blob analysis

When applied to his set of test data, Kale found that blob analysis performed significantly better than cascade classification. He hypothesizes that the performance of cascade classification could be improved by expanding the training data for the machine learning algorithm to include a bees in a wider variety of poses, so that it can better recognize them.

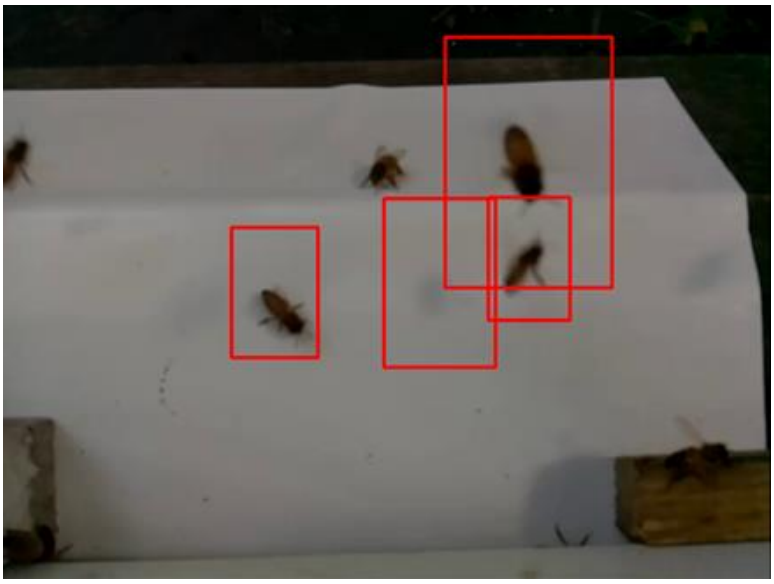

Figure 5.7: Video frame with cascade classification

Finally, Kale compares two approaches for tracking the motion of bees in the video. The optical flow method uses a variant of Lucas-Kanade optical flow estimation to find the motion of a region of pixels (i.e., a bee) between two images. The primary disadvantage of this method is that it is stateless, as it only compares two frames of the video simultaneously. The Kalman filter, however, maintains tracks for moving objects detected in the video, allowing it to make predictions about an object's location and to retain knowledge of objects that are not detected in the current frame. Figure 5.8 and Figure 5.9 demonstrate the motion tracking techniques applied to a frame of video.

The performance of the motion tracking methods was compared by having them count the number of bees entering and leaving the hive during two test video sequences, one with shadows and one without. The optical flow method was very inconsistent in performance, sometimes returning counts that were off by as much as 50%. However, it never over counted either arrivals or departures. The Kalman filtering was much more consistent in performance, displaying a similar amount of error for both arrivals and departures in both tests, though it did over count the number of arrivals in each test.

Figure 5.8: Video frame with optical flow

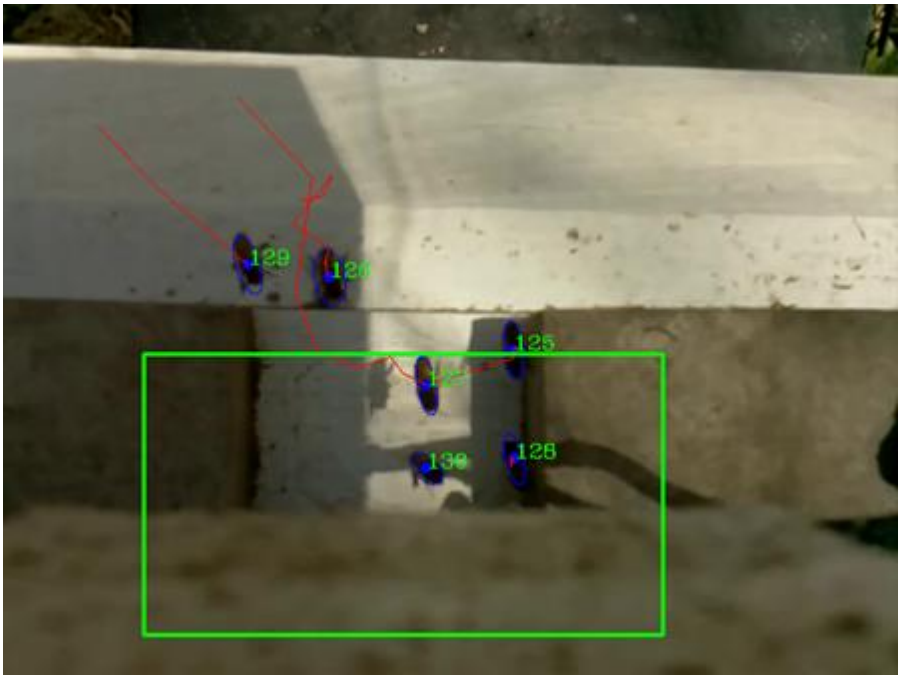

Figure 5.9: Video frame with Kalman filtering

Chapter 6 - Conclusion

6.1 BeeMon System

This thesis presented the design for an automated bee hive data collection system, capable of recording audio and video data. The system is built around the Raspberry Pi Model 1 B computer board. Audio capture is provided by a pair of USB microphones, while high definition video can be recorded using a Raspberry Pi Camera Module. This system is protected from the environment using a 3D printed case combining environmental protection, bee hive mounting system, and camera mount into a single unit.

The BeeMon application manages the system, providing the ability to control all aspects of audio and video recordings. Both types of data may be recorded using several scheduling options and quality settings, allowing the captured data to be configured for the bee hive being monitored and the needs of the research application consuming the audio or video. Once recorded, this data is temporarily stored on the Raspberry Pi until it can be uploaded to a remote server for storage and analysis.

After the BeeMon system has been installed on a bee hive and the application configured and started, the system is capable of completely autonomous operation. Manual action is only required if the data capture or upload settings need to be modified, or if a hardware failure occurs.

6.2 Data Collection and Analysis

The BeeMon design presented here has been deployed on multiple real-world bee hives and has been capturing data for three years. In that time a large volume of data has been recorded, providing a rich data set for use by other researchers that are investigating the sounds and motions of honey bees.

One research project has already been completed using data provided by BeeMon. In that project, computer vision techniques were applied to video recorded at the entrance of a bee hive in order to count the number of bees at the entrance, and track the number of bees entering and leaving the hive. The project was able to determine that a Gaussian mixture model provided good background subtraction results in most circumstances. It also concluded that blob analysis provided better performance for object detection than cascade classification, and that motion tracking results were more consistent when using the Kalman filter algorithm as opposed to optical flow.

6.3 Future Work

Numerous opportunities exist to enhance and expand the BeeMon system. In the time since this project began, several updated versions of the Raspberry Pi have been released (see infrared camera module.), providing increased processing power and capabilities while maintaining the same low cost. Future work could take advantage of this CPU power by having the Raspberry Pi perform onboard data conversions, for example, converting WAV audio files to the more space efficient MP3 format. The potential also exists for audio and/or video analysis to be performed directly on the Raspberry Pi in real time, enabling the Raspberry Pi to host a truly self-contained monitoring system that would need to transmit only the results of its analysis, rather than uploading large volumes of recorded data.

The integration of additional sensors into the BeeMon system is another area that future work could address. Microcontrollers such as the ATmega328P have the ability to interface with a wide variety of sensors and devices that are difficult to control using a higher level device like the Raspberry Pi. At the most basic level, the microcontroller would interface with sensors to record their readings, and it would expose a simple communication

interface by which the Raspberry Pi master unit could query it for the sensor readings. More advanced integration could allow for remote interaction with the hive being monitored. For example, the microcontroller could control a servo in order to perform a knock test; the command would be received by the internet connected Raspberry Pi and passed on to the microcontroller, which would then perform the detailed servo movements required.

In the current design, the server component of the BeeMon system is essentially a dumb file store whose only purpose is to receive recordings uploaded by the Raspberry Pi system. A more organized server component using a database to store file metadata and sensor readings would make it significantly easier to manage and query large amounts of bee hive data, particularly if temperature, humidity, or other sensors are added to the system. Such an enhancement will likely prove necessary before any future BeeMon-based monitoring system can be deployed at scale to a large number of bee hives.

Finally, the current BeeMon application is written C++ because it was believed that it would be necessary for it to directly access sensors at a low level. Section 4.2.3 details limitations of the Linux operating system that prevent this from being viable, meaning there is little reason for the application to be written in a low level language. Rewriting BeeMon in a higher level language such as Python would simplify the application, making it easier to perform maintenance and add new features. As discussed above, additional sensors added to the system would need to be managed using microcontrollers. Communication with these microcontrollers can be implemented using hardware built into the Raspberry Pi's processor, allowing the communications code to be written in a high level language.

Bibliography

[1] S. G. Potts et al., "Declines of managed honey bees and beekeepers in Europe," *Journal of Apicultural Research,* vol. 49, no. 1, pp. 15-22, 2010.

[2] R. A. Morse and N. W. Calderone, "The value of honey bees as pollinators of US crops in 2000," *Bee Culture,* vol. 128, no. 3, pp. 1-15, 2000.

[3] J. D. Ellis et al., "Colony losses, managed colony population decline, and Colony Collapse Disorder in the United States," *Journal of Apicultural Research,* vol. 49, no. 1, pp. 134-136, 2010.

[4] P. Meumann and N. L. Carreck, "Honey bee colony losses," *Journal of Apicultural Research,* vol. 49, no. 1, pp. 1-6, 2010.

[5] J. M. Campbell et al., "Capacitance-based sensor for monitoring bees passing through a tunnel," *Measurement Science and Technology,* vol. 16, no. 12, p. 2503, 2005.

[6] J. Campbell et al., "Video monitoring of honey bee colonies at the hive entrance," *Visual Observation & Analysis of Animal & Insect Behavior, ICPR,* vol. 8, pp. 1-4, 2008.

[7] V. Estivill-Castro et al., "Tracking bees-a 3d, outdoor small object environment," in *Proc.: 2003 Int. Conf. Image Processing (ICIP 2003)*, Barcelona, Spain, 2003.

[8] M. D. Meixner, "A historical review of managed honey bee populations in Europe and the United States and the factors that may affect them," *Journal of Invertebrate Pathology,* vol. 103, pp. S80-S95, 2010.

[9] T. D. Seeley et al., "Collective decision-making in honey bees: how colonies choose among nectar sources," *Behavioral Ecology and Sociobiology,* vol. 28, no. 4, pp. 277-290, 1991.

[10] M. H. Struye et al., "Microprocessor-controlled monitoring of honeybee flight activity at the hive entrance," *Apidologie,* vol. 25, pp. 384-395, 1994.

[11] J. C. Nieh, "The stop signal of honey bees: reconsidering its message," *Behavioral Ecology and Sociobiology,* vol. 33, no. 1, pp. 51-56, 1993.

[12] R. Okada et al., "The dance of the honeybee: How do honeybees dance to transfer food information effectively?," *Acta Biologica Hungarica,* vol. 59, no. 2, pp. 157-162, 2008.

[13] D. A. Mezquida and J. L. Martinez, "Platform for bee-hives monitoring based on sound analysis. A perpetual warehouse for swarm's daily activity," *Spanish Journal of Agricultural Research,* vol. 7, no. 4, pp. 824-828, 2009.

[14] S. Ferrari et al., "Monitoring of swarming sounds in bee hives for early detection of the swarming period," *Computers and Electronics in Agriculture,* vol. 64, no. 1, pp. 72-77, 2008.

[15] M. Bencsik et al., "Identification of the honey bee swarming process by analysing the time course of hive vibrations," *Computers and Electronics in Agriculture,* vol. 76, no. 1, pp. 44-50, 2011.

[16] U. Knauer et al., "Application of an adaptive background model for monitoring honeybees," *Proc.: Conf. Visualization, Imaging, and Image Processing (VIIP 2005),* Benidorm, Spain, 2005.

[17] T. Kimura et al., "A new approach for the simultaneous tracking of multiple honeybees for analysis of hive behavior," *Apidologie,* vol. 42, no. 5, p. 607, 2011.

[18] A. Ghadiri, "Implementation of an Automated Image Processing System for Observing the Activities of Honey Bees," M.S. thesis, Comput. Sci. Dept., Appalachian State Univ., Boone, NC, 2013.

[19] C. Chen et al., "An imaging system for monitoring the in-and-out activity of honey bees," *Computers and Electronics in Agriculture,* vol. 89, pp. 100-109, 2012.

[20] S. Streit et al., "Automatic life-long monitoring of individual insect behaviour now possible," *Zoology,* vol. 106, no. 3, pp. 169-171, 2003.

[21] A. E. Lundie, "Flight Activities of the Honeybee," United States Department of Agriculture, 1925.

[22] A. Decourtye et al., "Honeybee tracking with microchips: a new methodology to measure the effects of pesticides," *Ecotoxicology,* vol. 20, no. 2, pp. 429-437, 2011.

[23] K. Von Frisch, *The Dance Language and Orientation of Bees,* Cambridge, MA: Harvard Univ. Press, 1967.

[24] Microchip, "ATtiny2313 Microcontrollers and Processors," Microchip Technology Inc., 2017. [Online]. Available: http://www.microchip.com/wwwproducts/en/ATTINY2313. [Accessed 7 January 2017].

[25] Atmel Corporation, "ATmega328P," [Online]. Available: http://www.atmel.com/devices/atmega328p.aspx. [Accessed 11 December 2016].

[26] Analog Devices, Inc., "TMP36 Datasheet and Product Info," [Online]. Available: http://www.analog.com/en/products/analog-to-digital-converters/integrated-special-purpose-converters/digital-temperature-sensors/tmp36.html. [Accessed 12 December 2016].

[27] Adafruit Industries, "DHT22 temperature-humidity sensor + extras," [Online]. Available: https://www.adafruit.com/product/385. [Accessed 12 December 2016].

[28] Microchip, "MCP3008 - Mixed Signal - Successive Approximation Register (SAR) A/D Converters," Microchip Technology Inc., 2017. [Online]. Available: http://www.microchip.com/wwwproducts/en/MCP3008. [Accessed 7 January 2017].

[29] Texas Instruments Incorporated, "ADS1015 12-Bit ADC with Integrated MUX, PGA, Comparator, Oscillator, and Reference," Texas Instruments Inc., 2016. [Online]. Available: http://www.ti.com/product/ADS1015. [Accessed 7 January 2017].

[30] Microchip Technology Inc., "23K640 - Memory," Microchip Technology Inc., 2017. [Online]. Available: http://www.microchip.com/wwwproducts/en/23K640. [Accessed 7 January 2017].

[31] Texas Instruments Incorporated, "SN74LS157 Quadruple 2-Line To 1-Line Data Selectors/Multiplexers," Texas Instruments Incorporated, 2016. [Online]. Available: http://www.ti.com/product/SN74LS157/description. [Accessed 7 January 2016].

[32] SECO USA INC, "All-You-Need ARM Cortex A9 SBC Development Board," [Online]. Available: http://www.udoo.org/udoo-dual-and-quad/. [Accessed 12 December 2016].

[33] Raspberry Pi Foundation, "Raspberry Pi Products," [Online]. Available: https://www.raspberrypi.org/products/. [Accessed 11 December 2016].

[34] Raspberry Pi Foundation, "Raspberry Pi 1 Model B," [Online]. Available: https://www.raspberrypi.org/products/model-b/. [Accessed 12 December 2016].

[35] Raspberry Pi Foundation, "Raspberry Pi 1 Model B+," [Online]. Available: https://www.raspberrypi.org/products/model-b-plus/. [Accessed 12 December 2016].

[36] Raspberry Pi Foundation, "Raspberry Pi 2 Model B," [Online]. Available: https://www.raspberrypi.org/products/raspberry-pi-2-model-b/. [Accessed 12 December 2016].

[37] Raspberry Pi Foundation, "Raspberry Pi 3 Model B," [Online]. Available: https://www.raspberrypi.org/products/raspberry-pi-3-model-b/. [Accessed 12 December 2016].

[38] Toby, "SD & SDHC Cards that are compatible with the Raspberry Pi," ORBITS, 7 June 2012. [Online]. Available: http://www.raspberry-pi.co.uk/2012/06/07/compatible-sd-cards/. [Accessed 19 July 2017].

[39] Raspberry Pi Foundation, "Camera Module - Raspberry Pi," [Online]. Available: https://www.raspberrypi.org/products/camera-module/. [Accessed 12 December 2016].

[40] Adafruit Industries, "Electret Microphone Amplifier - MAX4466 with Adjustable Gain," [Online]. Available: https://www.adafruit.com/products/1063. [Accessed 3 January 2017].

[41] Amazon.com, "VAlinks® Plug and Play USB Desktop Audio Microphone Elegant Arc Design Adjustable USB Mic for PC, Laptop and Mac," [Online]. Available: https://www.amazon.com/VAlinks-Microphone-Adjustable-Applicable-Broadcasting/dp/B011K7EFKW. [Accessed 3 January 2017].

[42] "C++11 Overview, C++ FAQ," Standard C++ Foundation, 2017. [Online]. Available: https://isocpp.org/wiki/faq/cpp11. [Accessed 10 July 2017].

[43] "std::shared_ptr - cppreference.com," cppreference.com, 12 April 2017. [Online]. Available: http://en.cppreference.com/w/cpp/memory/shared_ptr. [Accessed 10 July 2017].

[44] B. Dawes, D. Abrahams and R. Rivera, "Boost C++ Libraries," [Online]. Available: http://www.boost.org/. [Accessed 10 July 2017].

[45] G. Henderson, "WiringPi," [Online]. Available: http://wiringpi.com/. [Accessed 14 January 2017].

[46] C. Ramey, "The GNU Readline Library," 13 September 2015. [Online]. Available: https://cnswww.cns.cwru.edu/php/chet/readline/rltop.html. [Accessed 14 January 2017].

[47] L. Thomason, "TinyXML Main Page," [Online]. Available: http://www.grinninglizard.com/tinyxml/. [Accessed 14 January 2017].

[48] K. Vehmanen, "Ecasound - multitrack audio processing tool," 2014. [Online]. Available: https://ecasound.seul.org/ecasound/. [Accessed 14 January 2017].

[49] R. Amorim, S. Mares and S. Fisher, "LAME MP3 Encoder," October 2011. [Online]. Available: http://lame.sourceforge.net/index.php. [Accessed 14 January 2017].

[50] A. Redmer and T. Stoeveken, "MJPG-streamer download," 15 May 2014. [Online]. Available: https://sourceforge.net/projects/mjpg-streamer/. [Accessed 14 January 2017].

[51] "jacksonliam/mjpg-streamer: Fork of http://sourceforge.net/projects/mjpg-streamer/," [Online]. Available: https://github.com/jacksonliam/mjpg-streamer. [Accessed 10 July 2017].

[52] Raspberry Pi Foundation, "raspistill," [Online]. Available: https://www.raspberrypi.org/documentation/usage/camera/raspicam/raspistill.md. [Accessed 14 January 2017].

[53] Raspberry Pi Foundation, "raspivid," [Online]. Available: https://www.raspberrypi.org/documentation/usage/camera/raspicam/raspivid.md. [Accessed 14 January 2017].

[54] Arduino, "Arduino - ArduinoBoardUno," [Online]. Available: http://www.atmel.com/devices/atmega328p.aspx. [Accessed 12 December 2016].

[55] "nanosleep(2): high-resolution sleep - Linux man page," [Online]. Available: https://linux.die.net/man/2/nanosleep. [Accessed 10 July 2017].

[56] "FrontPage - Raspbian," [Online]. Available: https://www.raspbian.org/FrontPage. [Accessed 10 July 2017].

[57] L. Aosong Electronics Co., "DHT22.pdf," [Online]. Available: https://www.sparkfun.com/datasheets/Sensors/Temperature/DHT22.pdf. [Accessed 19 July 2017].

[58] "RTwiki," 22 August 2016. [Online]. Available: https://rt.wiki.kernel.org/index.php/Main_Page. [Accessed 10 July 2017].

[59] P. McKenney, "A realtime preemption overview," 10 August 2005. [Online]. Available: https://lwn.net/Articles/146861/. [Accessed 19 April 2017].

[60] SD Association, "Speed Class - SD Association," SD Association, [Online]. Available: https://www.sdcard.org/developers/overview/speed_class/. [Accessed 9 July 2017].

[61] "Creo | PTC," PTC, 2017. [Online]. Available: https://www.ptc.com/en/cad/creo. [Accessed 10 July 2017].

[62] "MakerGear M2 - Top-Rated Desktop 3D Printer - MakerGear™," MakerGear, [Online]. Available: https://www.makergear.com/products/m2. [Accessed 10 July 2017].

[63] K. M. Nampoothiri et al., "An overview of the recent developments in polylactide (PLA) research," *Bioresource Technology,* vol. 101, no. 22, pp. 8493-8501, 2010.

[64] S. E. Tilmann, "In the Beekeepers Workshop," Michigan Beekeepers' Association, 23 March 2011. [Online]. Available: http://www.michiganbees.org/wp-content/uploads/2012/01/Hive-Bodies_20110323.pdf. [Accessed 3 July 2017].

[65] M. Simone-Finstrom and M. Spivak, "Propolis and bee health: the natural history and significance of resin use by honey bees," *Apidologie,* vol. 41, no. 3, pp. 295-311, 2010.

[66] N. Marriott, "Home · tmux/tmux Wiki," 8 June 2017. [Online]. Available: https://tmux.github.io. [Accessed 10 July 2017].

[67] "Screen - GNU Project - Free Software Foundation," Free SOftware Foundation, 27 April 2016. [Online]. Available: https://www.gnu.org/software/screen. [Accessed 10 July 2017].

[68] D. J. Kale, "Automated Beehive Surveillance Using Computer Vision," M.S. thesis, Comput. Sci. Dept., Appalachian State Univ., Boone, NC, 2015.

Vita

Mr. Michael Crawford was born in 1984 in Taylorsville, North Carolina to James and Joann Crawford. After finishing high school he briefly attended North Carolina State University, but was unsure what field he wished to pursue and left without finishing a degree. In the years that followed, he was employed in a diverse mixture of jobs such as loading delivery trucks, IT support, and banking customer support. In 2009 he graduated from Catawba Valley Community College with an Associate of Science degree. He eventually transferred to Appalachian State University, completing a Bachelor of Science in Computer Science before proceeding into graduate studies. As a student, he worked at Appalachian State University's Technology Support Center, as an undergraduate research assistant, and as a graduate teaching assistant. After completing his graduate coursework in 2015, he accepted a position with Eastman Chemical Company as a software engineer. In August 2017, he received the Master of Science in Computer Science degree.