Final Report: Boo-Bot

Devin Pohl
Department of Electrical and Computer Engineering
Colorado State University
Email: Devin.Pohl@colostate.edu

Daniel Garcia
Department of Computer Science
Colorado State University
Email: daniel95@rams.colostate.edu

***Abstract*—This project presents a small, remote controlled robot. This robot, nicknamed the "Boo-Bot" by the authors, is capable of sending real-time video/audio and taking real-time controls over the internet, allowing it to be controlled from anywhere with a network connection. This is showcased by a specific development of the project: interfacing over a Twitch live-stream. Throughout development this project has been made as from-scratch as possible. This has included designing of circuitry, PCBs, and 3D printed parts, as well as significant programming for drivers, communication, and interaction.**

I. Introduction

Built as a semester project for CS 370, Colorado State University's Operating Systems Course, *Boo-Bot* is a robot: small, self-contained, remote-controlled, high-quality, and designed from the ground up. It is made to answer the lack of a cheap, commercially available, semi-autonomous, long-distance, highly-portable IOT device for animate telecommunication. Despite the numerous challenges encountered during development, the final product is considered a successful implementation, providing significant learning experiences for the authors. In design and construction, the device provided key insights into areas of robotics, circuitry, and of course, operating systems. This paper provides an overview on why this project exists, the process of its design, and how that design was implemented. And while this paper presents a completed prototype, it is just that: a prototype. Additional development will be done in the future.

There are two main areas of development in this project: hardware and software. The first area, hardware, took the most time. Between custom designing a PCB to drive several peripherals, and designing and 3D printing the chassis, a diverse set of skills were required. The second area, software, was not without its developments. With components including GPIO interfacing, IRC communication, video streaming, and operating system interfacing, a healthy mix of written-from-scratch software, libraries, and complete pre-existing software was employed. Adding that all the software written requires non-trivial cross-compilation, several learning opportunities presented themselves.

II. Problem Characterization

The need for this project was born from a simple desire to experience a remote place. While photos and videos provide a good approximation, they lack integration. Virtual reality, while a great solution, is prohibitively expensive and lacks any sort of real-time factor. The solution, of course, is a remote controlled robot. However, long-distance remote controlled robots are few and far between in commercial spaces and often unable to provide a reasonable price tag. Designing and implementing this robot in-house is the clear solution.

With the end-goal in mind, specific objectives and constraints were then drafted. After careful consideration, we decided on the following set of items:

- Low cost: The target was $150 for a prototype unit.
- Small size: To ensure portability, low power consumption, and a more satisfying end product.
- Cleanliness: We wanted this to be done right. This required a custom PCB to improve cable management.
- Many peripherals: With a slew of sensors, motors, and other devices planned, this required a competent development board.
- High quality software: This decision lead us to the Rust programming language.
- Potential for future work: Required significant focus on easily installing and upgrading parts down the line when more development occurs.
- Low work duplication: Not wanting to completely reinvent the wheel, we wanted to use a popular ecosystem with plenty of community development.

With the problem described and a list of goals outlined, specific design decisions could be made. These will be discussed in further sections of this paper.

III. Proposed Solution and Implementation Strategy

This is a project with an abundance of moving parts. Most importantly is that both software and hardware have a large amount of notable developments. As such, this section will include a description of work done on hardware: first explaining the various pre-built tools utilized in this project, then the original work on software, and finally the original work on hardware.

A. Methodology

Due to the scope of this project, a large amount of existing solutions were used. This includes development tools for both hardware and software. On the hardware side is the development board and peripherals. The board we chose for the task is a Raspberry Pi 3 A+, decided on due to its low cost, abundance of ports, and vast community support. While we initially planned on using stepper motors, the output torque

of the motors we chose was too low. This resulted in us using standard DC motors for a first prototype that were on hand; we will return to stepper motors in further iterations of this project in the future. We also included a speaker amplifier, microphone, and several miscellaneous sensors. For a complete list of hardware, including sensors, electrical components, and material, see Appendix A. All hardware in this list has been successfully integrated into the current prototype.

The software side of this project uses a large amount of pre-built solutions, the largest of which being the operating system. Running on the Raspberry Pi 3 A+ is Raspbian Lite with a few modifications. These modifications are in the form of SPI-interfacing systemd modules, used to drive the microphone [1] and speaker amplifier [2]. Several shell-based programs are used in the current ecosystem as well:

- raspivid [3]: Used to read video data from the standard Raspberry Pi camera.
- ffmpeg [4]: Used to process video and stream to Twitch.
- Flite [5]: Used for text-to-speech. Allows the bot to read Twitch chat out-loud.

All other software is written from scratch, using the Rust programming language. Source code, explained further in III-B, is hosted on GitHub [6].

Hardware development can be seperated into two main categories: CAD and PCB. The CAD development culminated in the 3D printed chassis; parts were developed in AutoCAD and printed on Prusa i3 MK3S+ 3D printers. PCB development was done in EasyEDA, culminating in a custom hat for the Raspberry Pi to interface and power all peripherals. All aspects of hardware development are explained below.

B. Software

For this prototype, a git repository was set up, hosted on GitHub [6]. This repository holds all code written for this project, as well as all documentation. The main implementation language is Rust, which occasionally uses OS calls to spawn the shell-based programs described above. The only notable crate (Rust's term for a library) used is twitchchat [7]. It is essentially an IRC client implementation with Twitch-specific features – Twitch chat is a *very* non-standard IRC. Additional minor crates can be found at the GitHub repository [6]. All other code simply leverages the Rust standard library; this includes code for GPIO manipulation, process spawning, shell interaction, configuration reading, and connection to Twitch. Essentially the main program reads from a configuration file, connects to Twitch, and dispatches commands such as moving motors via GPIO, calling Flite for text-to-speech, and relaying system info back to chat. Additionally, admin controls were implemented to allow for fine-grain control over what users are allowed to run what commands in chat at what times.

The culmination of these efforts is a single program which can stream video to Twitch, interact with Twitch chat, and execute commands. A snapshot of this behavior can be seen in [Fig. 1]. Here, the bot can be seen connecting to Twitch chat and accepting commands. An audience member can request information, make the bot move, or issue a phrase from the speech synthesis engine. At this point, all commands shown in [Fig. 1] do as they say: `!info` requests information, `!forward` and `!b` move the bot forward and backward respectively, `!say` evokes text to speech, and `!quit` (an admin-only command) shuts down the Twitch chat monitoring.

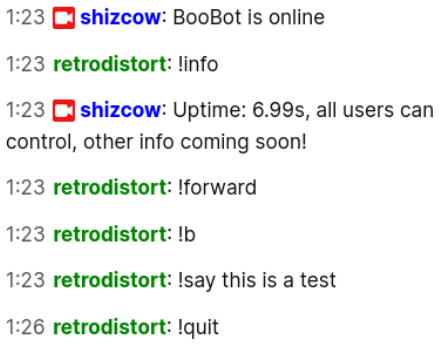

Fig. 1. Interactions with the bot in Twitch Chat

C. Hardware

This prototype uses a vast amount of sensors and actuators. Due to such complexities in hardware, a custom PCB was required to simplify the assembly process. The PCB was designed to fit as a hat on the Raspberry PI for quick removal in development. The PCB design was done using EasyEDA and manufactured through JLCPCB. A major focus for the PCB design was compatibility. Design considerations were taken from component datasheets and standardized GPIO pins. Additionally, significant emphasis was placed on redundancy during design; this includes power supply bypasses and a semi-modular motor driver interface to facilitate unexpected swapping of motors (as we ended up utilizing). The current iteration of the PCB is rendered in [Fig. 2].

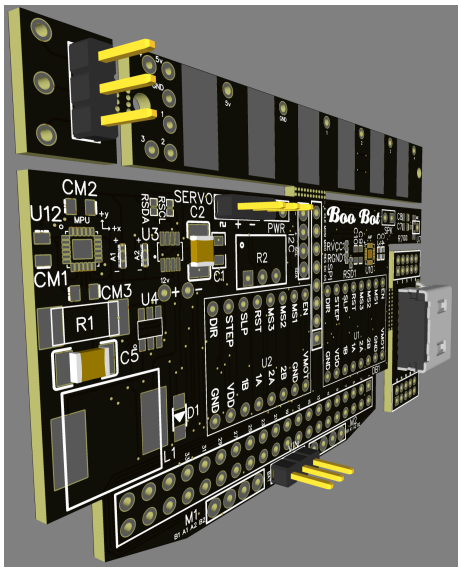

Fig. 2. 3D rendering of the PCB design

This prototype is a very visual device. Therefore, careful consideration was employed in designing a visually appealing chassis. Excluding the translucent dome, all 3D parts were designed using AutoCAD. Reference dimensions when designing were taken from component data sheets. For components lacking datasheets, measurements were done manually. Finally, once all parts were designed, 3D models were printed with a Prusa i3 MK3S+. After the printing process, the bot could be assembled. The prototype, assembled as it is currently, can be seen in [Fig. 3].

Fig. 3. The finished prototype, assembled

IV. Conclusions

A. Challenges

Throughout development of the project, we ran into several problems. These problems required us to spend additional time researching, testing, and overall gaining valuable experiences in the debug process.

One such challenge was creating a reproducible build chain. As the main code runs on an ARM-based processor, and as the development board has nowhere near the power to compile large programs on-board, cross compilation is required. Unfortunately, cross compiling Rust for the Raspberry Pi is a poorly documented task. While in the beginning a myriad of attempts were required to get even a simple binary built, over the course of development a robust pipeline emerged. Lessons learned from this encounter are to not be afraid of the official documentation. While easily accessible tutorials on websites may be dead-simple to follow, they may be out of date or incorrect. The only true up-to-date instructions are those derived from the source documentation.

Another major challenge came in the form of making the speaker amplifier work correctly. As previously mentioned in this report, Aidafruit officially makes tutorials for the parts they sell available online. However, the tutorial for the speaker amplifier we decided to use did not produce the claimed results. While we were easily able to create the required circuits, the systemd modules seemed destined to silently fail. The solution involved manual editing of udev rules, systemd trigger files, and build configuration for the amplifier drivers. If not for prior experience on the parts of the authors, this objective is unlikely to have been met.

Several additional challenges presented themselves while assembling and testing the circuit. The first of these was the severe lack of torque in the stepper motor. Because of this issue, we decided to instead go for a dc motor. Although significantly higher in both speed and torque, the DC motor lacked the precision of a DC motor. Fortunately, while designing the PCB, we anticipated issues regarding torque. Replacing controller boards were meant to be easy and without any trace modifications. Because of this design, the motors were replaced within 30 minutes. Another issue was the current and voltage sensors. While looking for the wiring of the IC, there were a few confusions for the pinout. Because of this, the sensor was wired incorrectly and removed afterward. Regarding the OLED display, this component was removed from the final design. Because of the lack of time and the less than essential purpose, it was removed completely. This feature may be added in a future iteration.

B. Final Thoughts

We consider this project a success. The current iteration is a feature-complete, highly portable IoT device which meets all the baseline goals we set during the planning phase as well as several stretch goals. As a class requirement, this project absolutely fulfilled its purpose of introducing concepts of operating systems in a tightly integrated environment. The learning experience in robotics, electronics, and operating systems was highly remarkable, inspiring further iterations in the future.

References

[1] L. Ada, "Raspberry Pi Setup: Adafruit I2S MEMS Microphone Breakout," 2017. [Online]. Available: https://learn.adafruit.com/adafruit-i2s-mems-microphone-breakout

[2] ——, "Raspberry Pi Setup: Adafruit MAX98357 I2S Class-D Mono Amp," 2016. [Online]. Available: https://learn.adafruit.com/adafruit-max98357-i2s-class-d-mono-amp

[3] B. Nuttall, J. Hughes, R. M. Churcher, D. T. Baker, and A. Molinaro, "Raspbian documentation: raspivid," 2020. [Online]. Available: https://github.com/raspberrypi/documentation/blob/master/usage/camera/raspicam/raspivid.md

[4] FFmpeg Developer Organization, "Ffmpeg," 2021. [Online]. Available: https://ffmpeg.org/

[5] A. W. Black, "Flite," 2018. [Online]. Available: https://github.com/festvox/flite

[6] D. Pohl and D. Garcia, *Shizcow/BooBot: 0.1.0*. Zenodo, Apr 2021. [Online]. Available: https://zenodo.org/record/4726009

[7] twitchchat Developer Team, "twitchchat: interface to the irc portion of Twitch's chat," 2021. [Online]. Available: https://github.com/museun/twitchchat

APPENDIX A
BILL OF MATERIALS

Below, [Tab. I] presents a bill of materials for this project, including all hardware used, parts ordered, and material required. All items present have been successfully integrated into the current iteration of the prototype.

TABLE I
BILL OF MATERIALS

Amount	Component	Price Ea ($)	Description	Cost ($)
10	Capacitors	0.466	16v 1000UF Electrolitic SMD	4.66
2	Drivers	7.82	39-1500RPM DC 6V Electric Motor with Gear Box	15.64
2	Motors	3.98	MINEBEA NMB 2-phase 4-Wire 18° Stepper Motor	7.96
1	9-axis MPU	4.60	MPU9250 (Gyro, Accelerometer, Compass)	4.60
1	ADC	1.69	INA219 DC current and voltage sensor	1.69
1	Amp	4.99	MAX98357A I2S Class D amplifier	4.99
1	Lipo	15.05	Lipo battery pack	15.05
1	Microphone	7.51	I2S MEMS Microphone SPH0645LM4H	7.51
1	PCB	8.00	5 Custom PCBs from EASY EDA	8.00
1	PLA Filament	5.00	100g Black PLA filament 1.75 mm	5.00
1	Raspberry pi	29.99	Raspberry Pi 3 Model A+ 2018 model	29.99
1	SD Card	5.00	32 GB Class 10 Micro SD Card	5.00
1	Servo	1.79	SG90 9G Micro Servo Motor	1.79
1	Speaker	0.99	8 ohm speaker	0.99
1	Voltage Regulator	0.79	B628 3-24V to 12V 2A Adjustable Boost Step-Up Converter	0.79
			Total:	113.66