Ancilla

A personal assistant built from scratch

Designer and developer:

Marc Soda

CSE371 Principles of Mobile Computing

Professor Mooi Choo Chuah

Final Project Report

3 December 2022

Introduction and Motivation

Ancilla is a personal assistant, similar to Amazon's Alexa or Google's Google Home that performs tasks given to it via audible cues. It has the ability to control devices, manage alarms, and perform simple internet search queries. I was motivated to do this project for three reasons. 1: because I thought it would be super cool. 2: because I very much wanted to enjoy the benefits of a personal assistant, but I am unwilling to be constantly tracked by scummy companies like Amazon and Google. 3: because I wanted my personal assistant to have the features that I want and to perform functions in the manner that I decide. I did not want to be limited by the limited capabilities of existing solutions. I did not plan on producing this device, meaning that I could design its features to be convenient for me and me alone. I did not have to consider any other users. The product was an extremely efficient and convenient personal assistant that is very private and modular. The system has no limits other than a short list of features, but it was designed in such a way that more features can easily be modularly built and added independently of any other feature. I am very happy with how it turned out.

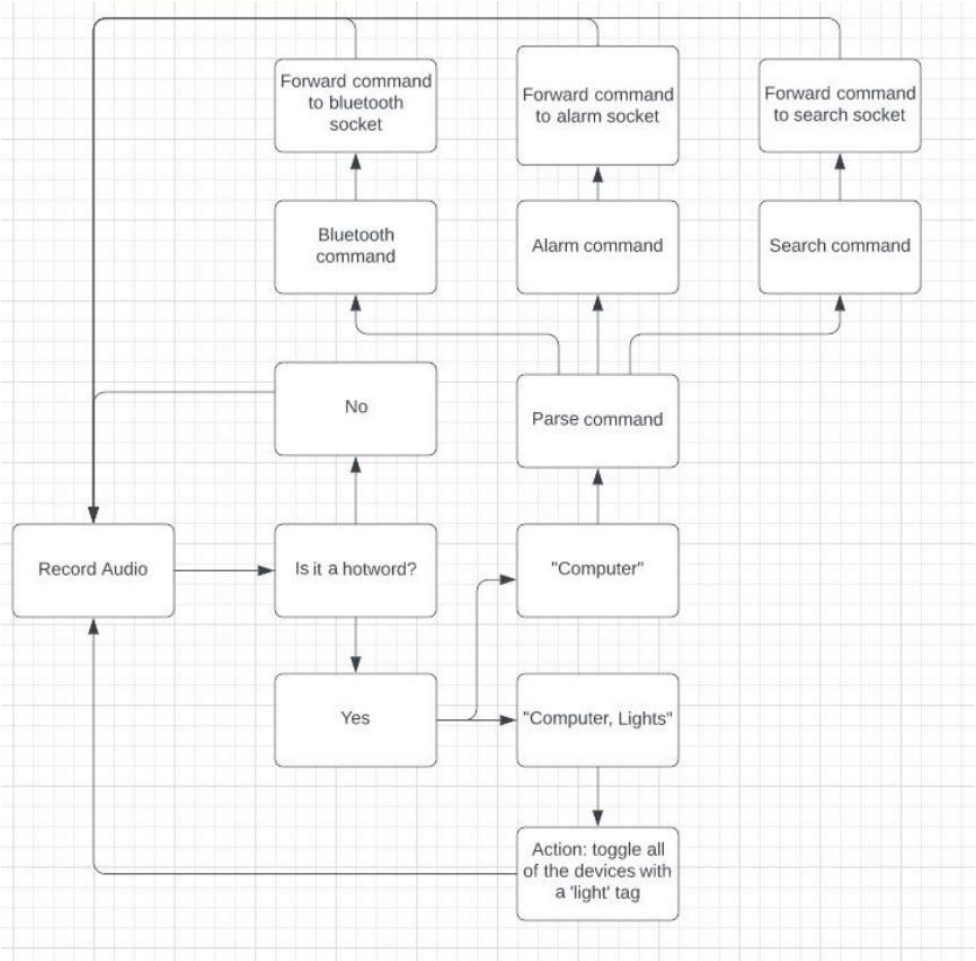
Hardware, software, and dataflow overview

The brain of the system is a Raspberry Pi 3B+ running a headless derivative of Debian known as Raspbian OS. The only other hardware on the main system is an arbitrary microphone and speaker combination.

Everything is a socket. Each of the three features are set up using client Unix sockets to the main Ancilla server Unix socket. They communicate back and forth between each other via socket communication. The Ancilla socket does many things: It performs hotword detection on a separate thread (so as to not block the UI thread) using the (deprecated) Snowboy hotword detection library. A hotword (also known as a wake word) is the word(s) which rouse the system. For example, the hotword on an Amazon Alexa is “Alexa”. The hotword for Ancilla is “Computer”. Ancilla also supports a second hotword, “Computer Lights” which toggles all of the lights very quickly. It is able to do this hotword detection using machine learning. Each of the hotwords is represented as a trained model from a dataset of audio files in which a person says the hotword. After a hotword is detected, it begins listening for speech and is able to detect exactly when there is a break in the user’s speech. It sends the recorded data to Google’s (more on this choice later) speech to text API. It analyzes this text and dispatches the command to

the appropriate client socket who then performs some action depending on the nature of the command. It listens (on a separate thread) for an optional response from each socket. If the response contains verbal data, it is fed into the PYTTSX3 text to speech library which causes Ancilla to give a verbal response. The responses may also contain error messages or messages that ask Ancilla to forward information to another socket. The client sockets do not communicate with each other without using the Ancilla server socket as an intermediary. It also manages the integrity of the connections between the client sockets. It is able to have direct control over client processes and, in the event that a client socket is broken or nonexistent, it can send systemd commands to interact with them. Each feature is set up as a systemd service for convenience. Each of the other client sockets receive input from the Ancilla socket and perform some action on that input. Details are discussed in the following sections.

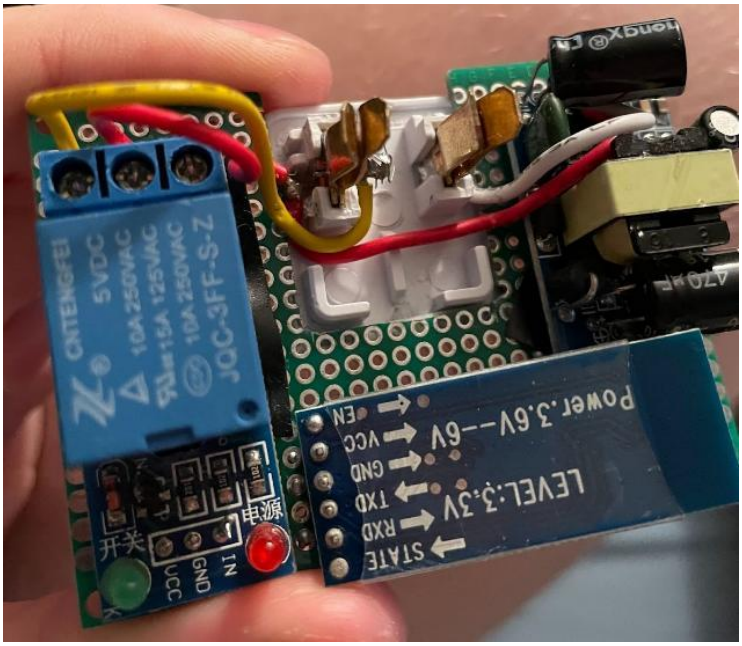
Some of the logic and dataflow on the main machine can be realized through the following rough state diagram:

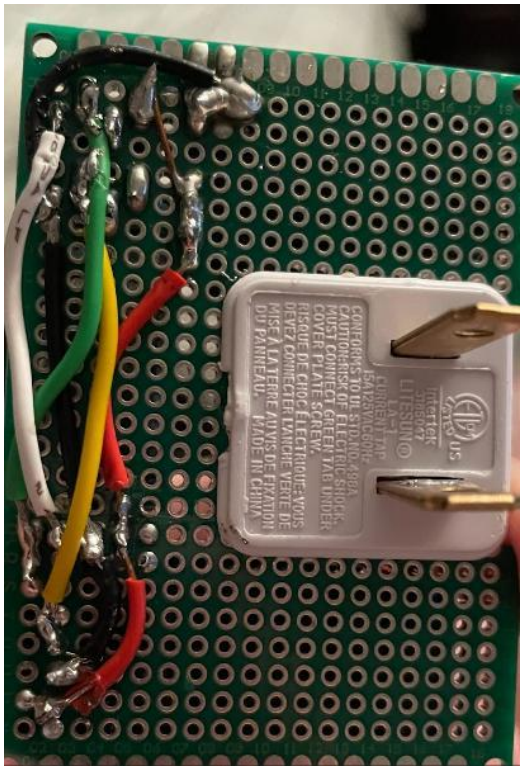


The figure is extremely simplified and does not show what happens to the data when it reaches the client sockets, responses from the clients, or what happens on each distinct thread.

Device control

The device control client uses bluetooth sockets to communicate with Arduino bluetooth devices that I have designed. The following two figures show a front and back view of one of these devices.





The device is controlled by an ATTiny85 Arduino (not visible, under the bluetooth module). The devices plug directly into a 120V wall outlet. There is a 120V AC to 5V DC converter module that receives AC power from the wall and provides DC power to each module (Arduino, bluetooth, relay). The left terminal on the outlet is bisected which allows the power flow to be controlled by the relay. The Arduino listens for messages to the bluetooth module. If a ‘0’ is received, the Arduino instructs the relay to sever the connection. If a ‘1’ is received, the connection is reestablished. If a ‘2’ is received, the power state is toggled.

The device management process maintains the integrity of the bluetooth connections to the various devices on a separate thread. The user need not be concerned with connection issues. The manager thread handles all issues. It has never failed during my testing.

The device management socket also blocks while waiting for instructions from the main Ancilla server socket. Each device has a tag associated with it that acts as a description of the device. Hypothetically, if I had 20 lights connected to Ancilla, the tags would be useful in grouping each device into different categories so they can be controlled more conveniently. Using inheritance, Ancilla supports different types of devices. For example, all devices have functions such as `connect()`, `sendData()`, and `ping()` and subclasses such as lights or heaters would inherit those methods while being able to define their own. Currently the only built device type is lights, but there is no reason why I couldn't build subclasses for a heater or a toaster if the need arose.

Alarm management

Ancilla will also manage the user's alarms. The alarm process will block until it receives a command from the main Ancill socket. Note that the alarm management happens on a separate thread so the UI thread is never

blocked. A user can set alarms, name alarms, list alarms, delete alarms, and clear all alarms. Alarm entries persist on disk so they are maintained if the power is lost. Alarms can recur or happen only once. When an alarm goes off, all of the lights turn on and an annoying noise plays until the user dismisses it by voice.

Simple search

Ancilla supports internet search queries. The simple search process will block until it receives a command from the Ancilla socket. After it receives a query, it will forward that query to the WolframAlpha search API. It will then respond by sending the response from WolframAlpha to Ancilla who then instructs PYTTSX3 to verbalize said response. The WolframAlpha search API is not designed to be spoken so some (few) responses contain symbols and images. Responses that contain no text data cause Ancilla to say “Search result contained no text data”. The search API is therefore a little awkward at times, but it certainly adds significant value to the system. I have some ideas for improvement which I will talk about in a future section.

Future work and major design considerations

I, of course, intend to further build out the functionality of the device. I would like to support different kinds of devices such as heaters which would

require me to build a heater subclass on the device super class. More importantly, I will need to modify the design of the bluetooth device by adding a thermometer. I would also like to have support for reminders, calendar management, todo lists, shopping lists, etc.

I intend to have multiple Ancilla devices throughout my home, so they will need to be connected via some sort of network. My plan is to buy a beefy Digital Ocean Droplet which will serve as the central server for each of my Ancillas. Each Ancilla will push and pull data from this machine so they all have consistent copies of the state of the overall system.

You may have found it curious that I bashed Google when I was talking about privacy and yet I am using Google's speech to text API. This is a temporary but major design flaw. I did a lot of testing with various free speech to text APIs and even some offline speech to text libraries and Google outperforms them in terms of speed and accuracy by a huge margin. My options were either to have Ancilla be slow and inaccurate or to slightly reduce privacy. I opted to use Google temporarily so I could get my submission ready in time, but I have a better option. I plan to use the central Digital Ocean Droplet to handle speech to text. The issue was that the limited processing power of the Raspberry Pi did not allow for me to use offline

speech recognition because good offline speech recognition requires much more power than the Raspberry Pi can provide. The Digital Ocean Droplet solution solves the privacy and speed and accuracy issue perfectly. It will just take some work to get it setup properly.

The responses from WolframAlpha tend to be slightly awkward when they are read by Ancilla. I intend to build out templated queries which will increase readability. For example, when you ask Ancilla what the weather is like, Ancilla will recognize the weather query as a templated query. It will feed the response into the weather template which will extract the data from the API response and reconstruct it into a more naturally spoken format.

It is important to note that after the speech to text is done, there is no machine learning or other complicated mechanism to determine intent of the command. This is by design. All commands must be prepended by the name of the socket to forward the command to. For example, if you say “How old is Obama", the device will do nothing because it doesn’t know what you’re talking about. However, if you say “Search How old is Obama”, it will forward your command to the search socket. I prefer communicating rigidly with the device. It, however, might be an interesting project to create a machine learning model for intent detection. After speech recognition is

performed, the text will be fed into the intent detection model which will tell Ancilla where to forward the command. This will eliminate the need to specify the name of the socket thus making communication more human.

Conclusion

Ancilla is a personal assistant from the ground up. It allows the user to efficiently control the power flowing to any device, manage alarms, and do simple internet searches. The system was built in such a way that allows new features to be modularly added without affecting any other feature due to the fact that all feature processes are completely isolated from each other. New types of devices can be modularly added through the magic of inheritance, so Ancilla is not limited to only being able to control light devices. There are a lot of features and devices that are planned to be added and supported. Ancilla is private, fast, and efficient.