ChatterPi Documentation

Mike McGurrin

Version 0.9, October 9, 2020

Table of Contents

Introduction

ChatterPi is a software package that turns a Raspberry Pi into an audio servo controller. In other words, the Pi outputs commands to control a servo based on the volume of the audio input. The input can be either stored audio files (in either mono or stereo .wav format) or from an external source, such as a microphone or line level input. One of the uses is to drive animatronic props, such as a skull or a talking bird.

The documentation may seem a bit overwhelming at first glance, but that is primarily because of the large number of user options that are provided. Once you have the hardware setup and the prerequisite software loaded, see the QuickStart section to get started with default settings for most of the parameters.

ChatterPi was developed on a Pi 3 A+ and tested on a Pi 3 A+ and a Pi Zero W. Given that it will run on a Pi Zero, it should run on any Pi.

Background: A Brief History of Talking Skull Control

A common prop that still makes a good impact is a talking object, whether a skull or animal. Some lower cost commercial props use a motor and spring. Another approach is to pre-program a complete sequence to match the vocals, but this is very time consuming and if you want to change the vocals, or even just edit them slightly, you need to reprogram the entire sequence. For that reason, the use of an audio servo controller to drive a servomotor controlling the jaw is a very popular approach. There are several variations. One of the earliest use hardware to detect when the audio exceeded a threshold, and then began moving the jaw towards a fully open position, and when the audio went below the threshold, it would begin closing the jaw. “Scary Terry” Simmons may have been the first to develop an electronic hardware board to do this, and Cowlacious Designs has continued to improve and sell commercial versions, with many added features such as a built in audio player, various triggering options, and the ability to control LEDs as eyes.

Later, someone named Mike (no relation) combined an Arduino with a hardware volume level board to produce the Jawduino. This went from having just 2 levels to 4. The original project just took audio in and controlled the servo, but others added extensions to play stored mp3 files and/or randomly move additional servos (for example, http://batbuddy.org/resources/Halloweenstuff/TalkingSkull.php).

A few years ago, Steve Bjork from Haunt Hackers combined dedicated hardware with a propeller microcontroller to increase the number of levels to almost 256 and also to filter out low and high frequencies that don’t tend to result in jaw movement for spoken sound. The result is the Wee Little Talker. This commercial board also has an onboard mp3 player, can be triggered externally, control LED ‘eyes,” and adds a wide array of features including a voice feedback menu system.

It occurred to me that with current single board computer capabilities and powerful software libraries, it should be possible to incorporate most of the best features of all of these into a single, software-based system running on a Raspberry Pi. The result is ChatterPi. ChatterPi was developed from scratch using the Python language, but ideas for capabilities and features were freely borrowed from previous audio servo controller projects.

Features

ChatterPI is designed to be extremely powerful and flexible without requiring the user to modify any of the code (although advanced users can certainly do that as well). Table 1 compares the current capabilities of ChatterPi with other audio servo controllers. The full range of capabilities and options are described in the Operations subsection.

Table 1. Feature comparison of ChatterPi and other audio servo controllers

	Cowlacious Scary Terry Board	Original Jawduino	Wee Little Talker	ChatterPi
Audio signal controls servo	✓	✓	✓	✓
External Trigger	✓	X	✓	✓
Timer Trigger	X	X	X	✓
File or ext. audio input	✓	**external only**	✓	✓
Control Levels	**2**	**4**	**~256**	**4**
Bandwidth Filter	X	X	✓	✓
Plays "ambient" sound files	X	X	**Multiple**	**Multiple**
"Gain Control"	X	X	**Automatic**	**via Control Panel**
Trigger Out	✓	X	X	✓
Control LED "eyes"	✓	X	✓	✓
Settings	**Few (control knobs)**	**Requires software edit**	**Comprehensive voice menu**	**GUI or edit config file**

Using ChatterPi

This section describes how to set up and install both the hardware and software for using ChatterPi, and for using it.

Hardware

ChatterPi was developed on a Pi 3 A+ and tested on a Pi 3 A+ and a Pi Zero W. Given that it will run on a Pi Zero, it should run on any Pi. The original version had a slow list processing loop in the audio processing section which kept it from running on a Pi Zero. The latest versions switched from using structures, lists, and loops to numpy arrays, and that fixed the problem.

In addition to the Raspberry Pi, you'll need a USB sound card. This is needed for several reasons. First, if you plan to use an external sound source, you need a way to get audio into your Pi. Second, besides not producing very good sound, the audio out connector may share timing with the Pulse Width Modulation (PWM) code that is needed to drive the servo, creating conflicts. Use of an inexpensive USB sound card solves both issues. I've used one from Adafruit that sells for less than $5 and works well (see https://www.adafruit.com/product/1475). You will need TRS (standard stereo) plugs or an adapter to go into the headphone and microphone jacks on the sound card. The card does not work with a TRRS (combination microphone / stereo headphone plug. You only need a microphone or other external sound source if you wish to use one. Otherwise you can use audio .wav files that you save on the Raspberry Pi. You still need the USB sound card for audio output, however.

That's all you need for the audio servo controller. Of course, you'll need a power supply and a servo that you want to control, such as a servo-equipped talking skull, and a passive infrared sensor (PIR) if you want to trigger your prop using one. I used this one (https://www.parallax.com/product/555-28027) from Parallax for development, as I had a spare one already. ChatterPi can also be set to trigger off of a repeating timer or to just turn on and run if you don't want to use an external sensor.

Figure 1 shows a test bench setup for testing operation. The Red LED is attached to the "TRIGGER_OUT" pin for testing purposes. It can be moved or another LED and resistor attached to the "EYES_PIN" to test that feature. The TRIGGER_OUT pin goes high for 0.5 seconds when the controller is triggered. This can be used to trigger another prop or controller. The EYES_PIN stays high for as long as the audio plays.

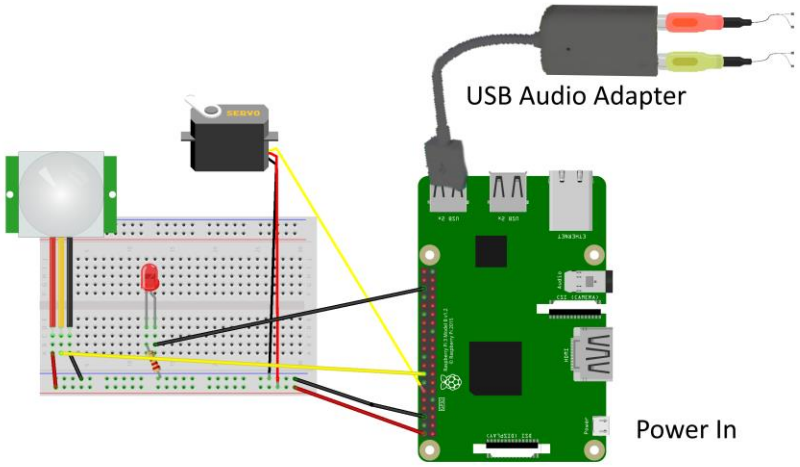

Figure 1. A layout for fully testing the ChatterPi

The default PIN selection (which can be changed in the config.ini file) are:

- Jaw servo: 18
- PIR input trigger: 23
- Trigger out: 16
- Eyes: 25

Figure 2 is a photograph of my test setup. The placement of the wiring on the breadboard is slightly different because I was using it to test a variety of items and also a 3-wire servo controller wire, but the schematic connections are identical.

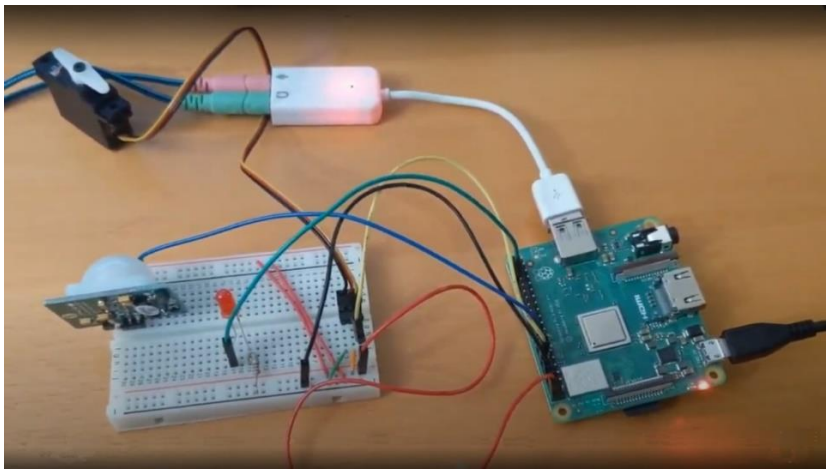

Figure 2. A picture of the test seup used for development

Software Installation and Setup

This section assumes you have a Raspberry Pi with Raspbian installed and either running with a keyboard and display or in headless mode remoted to another system. There are good instructions on the Internet already on how to get a Raspberry Pi up and running. You also need to have the Pi connected to the Internet to download software.

ChatterPi is written in Python 3, and Python 3 comes already installed with Raspbian. Hower, in addition to the ChatterPi modules, you'll need to install and initialize several other libraries that are not included out of the box in Raspbian. These are:

- GPIO Zero : This is already installed if you have loaded the full Raspbian operating system, which is recommended. This is the library ChatterPi uses to control the General-Purpose Input/Output (GPIO) pins on the Raspberry Pi. It is very user friendly for programming.
- PiGPIO: this is the underlying pin control library that is used "underneath the covers" of Gpio Zero.
- PortAudio: A library for audio playback and recording.
- PyAudio: The wrapper that lets Python programs access PortAudio.
- Scipy: A Python library for scientific and technical computing. ChatterPi uses its bandwidth filter function in some modes.

Here's how to go about setting this up. On the Raspberry Pi, open a command line. It's always a good idea to be sure all the packages on your system are up to date. Type the following:

```
$ sudo apt-get update
$ sudo apt-get upgrade
```

If you are running the full Raspbian installation, you can skip these steps. If you are running Raspbian Lite (untested), type:

```
$ sudo apt update
```

Then install the package for Python 3:

```
$ sudo apt install python3-gpiozero
```

To install PiGPIO, type:

```
$ sudo apt install pigpio
$ sudo systemctl enable pigpiod

$ sudo apt-get install python3-pigpio
```

The first command downloads and install PiGPIO, while the second command ensures that it starts running in the background whenever the Pi starts. The third command installs the required python wrapper module.

To install PortAudio:

```
$ sudo apt-get install portaudio19-dev
```

To install PyAudio

```
$ sudo apt-get install python3-pyaudio
```

To install Scipy, use the command:

```
$ sudo apt-get install python3-scipy
```

Audio Device Setup and Volume Control

The easiest way to select the USB sound card for audio input and output is through the volume control on the PI's desktop. Right click on the volume (speaker) icon on the upper right of the screen and select USB Audio Device for input and output. You can also use this control to adjust the volume.

To adjust the volume from the command line, type:

```
$ alsamixer
```

Do not set the microphone input or speaker output levels too high, as you may generate too much noise or feedback.

If your USB card does not show up, then that card does not have hardware support. Google "raspberry pi software volume control" to find out how to control the volume via the command line.

Last step: reboot your Pi so that all changes will take effect.

Congratulations, the prerequisites are now all installed, configured and ready for use!

Operation

QuickStart: Over time, you should explore and experiment with the many parameters can be changed in the config.ini file. However, the number of parameters can seem intimidating, so this section will get you up and running with a minimum of work. Initial default values have been set for all the parameters, the source is set to use audio files, and a sample audio file is included, so you do not need your own audio file or a microphone to get started. The trigger is set to TIMER mode, so an external sensor such as a PIR is not needed.

First, install the ChatterPi files in `/home/pi/chatterpi`. If you know the upper and lower limits for your servo, open the `config.ini` file for editing and set the SERVO_MIN and SERVO_MAX values (in

microseconds) the values for your servo. Then save the file. Now go to the terminal window and go to the `/home/pi/chatterpi directory. Then run ChatterPi:`

```
$ cd /home/pi/chatterpi
$ python3 main.py
```

You'll see a lot of warnings and errors concerning "ALSA" but don't worry about them. Believe it or not, this is normal. Every 10 seconds, the audio will play and the servo will move in sync. *Congratulations!*

Control-C will stop the program.

Detailed Instructions: The ChatterPi files should be installed in `/home/pi/chatterpi`. Within the chatterpi directory there is a subdirectory called `vocals`. The sound files you wish to play should be placed in that folder. The files must be 16 bit .wav files (the most common type) and can be either mono or stereo files. You must use the naming convention `v##.wav`, where xx ranges from 01 to 10. The purpose is so that you can have the prop play different audio tracks, in a set order, one each time the prop is triggered. The first time the prop is triggered it will play the lowest numbered file, and cycle through the list, skipping any missing numbers, and starting over again with the first track after all have been played. If you only want to play one track, that's fine, just name it `v01.wav`. If you need to edit your files, or you have mp3 or other format files that you need to convert to .wav to work with ChatterPi, the free and excellent Audacity software can do this and much more.

Ambient tracks, which you can optionally choose to play between event triggering, work similarly. They must be 16 bit.wav files, placed in a subdirectory called `ambient` in the chatterpi directory. Again, you can have up to ten different ambient tracks to cycle through, named `a01.wav` through `a10.wav`.

Use of Separate Tracks in Stereo Files: If you just have a mono or stereo sound file that you want to play and control the servo, you can skip this section. However there are some tricks that can be used with stereo .wav files to improve the fidelity of your jaw motion in some scenarios. The jaw motion is actually controlled exclusively by the right channel of stereo .wav files. The audio output sent to the speakers can be set using the OUTPUT_CHANNELS parameter to be either LEFT (only) or BOTH (both left and right channels. The reason for providing this option is that some haunters want to do something different with the right channel. One option is to record the desired complete audio output (the voice along, with music or other sounds) on the left channel (channel 0) and record a separate track of just sounds or signal tones, to drive the servo but not be played out through the speakers, on the right channel (channel 1). That way you can more precisely control the servo action and have it, for example, not open wide when a loud "ssssss" sound occurs. To do this, set the OUTPUT_CHANNELS parameter to LEFT, since you do not want to play the right channel through the speakers. Another option is to provide the voice on the right channel, and any accompanying audio or other sounds on the left. In this case, set the OUTPUT CHANNELS to BOTH. The right channel, as always, will control the servo, but both the voice and accompanying music will be sent out to the speakers.

Configuration: The options and parameters for ChatterPi are set in the `config.ini` file, which should be edited and saved with your desired settings prior to running the program. ChatterPi comes with a default config.ini file. It is recommended that you save a backup copy under another name. That way, you can always revert back to a working configuration file. `Config.ini` is divided into 5 sections, which we will now walk through. A copy of the contents of the default `config.ini` file is provided in

Appendix B. There are a lot of parameters you can change, in order to provide flexibility, however you do not have to set or change them all. Simply focus on the parameters of interest to you. As described in the **QuickStart** section, you can run ChatterPi with the default settings right out of the box.

To edit the settings, you can either use your favorite editor to directly edit the `config.ini` file or you can run the program `controlPanel.py`. This program provides a simple GUI interface for editing the `config.ini` file.

[SERVO]

This section has four parameters. SERVO_MIN and SERVO_MAX are the minimum and maximum values to set your servo to, to avoid going beyond it or the prop's limit. This is provided in microseconds (the time of the pulse signals).

MIN_ANGLE and MAX_ANGLE are the open and closed position for your servo, in degrees. If your jaw is *closed* when the servo is commanded to a high value and *open* when the servo is commanded to a minimum value, you should set the MAX_ANGLE to the value for closed and the MIN_ANGLE to the value for open.

[CONTROLLER]

This section selects the "style" to be used for controlling the servo and the threshold values for each style. There are 3 styles, numbered 0, 1, and 2. Style 0 operates like a Scary Terry style audio servo controller. It begins to open the jaw fully when the audio input volume exceeds the threshold, and begins completely closing the jaw when the volume is below that threshold.

Style 1 operates like the jawduino. The servo is commanded to open 1/3rd of the way if the first threshold is exceeded, 2/3rds if the 2nd threshold is exceeded, and completely if the third threshold is exceeded.

Style 2 functions like style 1, but first applies a bandpass filter to filter out low and high frequencies, since in speech, these do not tend to cause jaw movements.

The audio volume of each sample of the audio input is an integer that may range from 0 to 32767. The thresholds should similarly be set to integer values in this range.

[AUDIO]

This section specifies the source for the audio input (either FILES or MICROPHONE). Files must be 16 bit .wav files, either monophone or stereo. External input can be either from a microphone or line in from another source.

The OUTPUT_CHANNELS specifies, for stereo files, whether both channels should be sent to the speaker (BOTH) or just the left channel (LEFT). See the subsection above titled **Use of Separate Tracks in Stereo Files** for more information on why and how you may want to use this feature. If you have not done any special editing on your files, leave it set to BOTH.

When you are using .wav files the software can determine when it has reached the end of the file. If you are using external input by setting SOURCE to MICROPHONE, the software has no way to determine when an audio stream has ended. The MIC_TIME setting, in seconds, specifies how long to keep the

audio stream running form the external input when ChatterPi is triggered. I've found that the input levels on my setup are lower when using a microphone so I have to lower my trigger levels.

The AMBIENT parameter selects whether you want ambient audio tracks, placed in the `ambient` subdirectory, to be played between triggering events. If set to `ON`, ambient tracks will be played, beginning upon start-up. The playback will be interrupted if a triggering event (timer or sensor) occurs. Then, when the vocal track has completed, the ambient track will begin playing again after an additional delay set by the `Delay` parameter.

There is one other parameter in this section, the BUFFER_SIZE. This parameter is not currently used.

[PROP]

The parameters in the PROP section control how the prop is triggered and what else may be triggered or turned on at the same time.

PROP_TRIGGER takes one of three values, PIR, TIMER, or START. PIR will have ChatterPi start the audio and controlling the servo whenever the PIR_PIN is sent a 3V signal from another device. Often, this may be a Passive Infrared (PIR) motion detector sensor, but it can be anything you'd like. When set to TIMER, ChatterPi starts the audio and controls the servo every time the timer expires. The Delay time, in seconds, between the end of one activation and the start of the next is set by the DELAY parameter. START is only a valid setting when the SOURCE is MICROPHONE. It turns on the audio input and servo control immediately upon start up and leaves it open and running until the program is ended.

In PIR mode, the Delay time specifies the dead time, in seconds, between completion of a vocal track playing back an when the sensor trigger is again active. If Ambient is set to ON, it is also the delay between the endo of a vocal track playing and the ambient track starting to play.

EYES may be set to ON or OFF, to control whether or not the EYES_PIN is set high when the prop is triggered and stay high until the audio input ends.

TRIGGER_OUT may be set to ON or OFF to control whether or not the TRIGGER_OUT_PIN is set to high for 0.5 seconds when the prop is triggered. This can be used to trigger another device or prop.

The effects of the SOURCE and PROP_TRIGGER selections are summarized in Table 2.

Table 2. Behavior is Dependent Upon SOURCE and PROP_TRIGGER Settings

	PIR	TIMER	START
FILES	Audio files and servo control starts when trigger goes to high, stays active until the audio file ends. Eyes, if ON, light up while active. TRIGGER_OUT, if ON, goes high for 0.5 seconds at start of each activation.	Audio external input and servo control starts time expires, stays active until the audio file ends. Then the timer is restarted. Eyes, if ON, light up while active. TRIGGER_OUT, if ON, goes high for 0.5 seconds at start of each activation.	N/A. Will cause an error

MICROPHONE	Audio external input and servo control starts when trigger goes to high, stays on for length of time set by the MIC_TIME parameter (in seconds) Eyes, if ON, light up while active. TRIGGER_OUT, if ON, goes high for 0.5 seconds at start of each activation.	Audio external input and servo control starts when trigger goes to high, stays on for length of time set by the MIC_TIME parameter (in seconds). Then the timer is restarted. Eyes, if ON, light up while active. TRIGGER_OUT, if ON, goes high for 0.5 seconds at start of each activation.	Audio external input and servo control starts as soon as ChatterPi starts running and stays active until the program is terminated. Eyes, if ON, light up the entire time. TRIGGER_OUT, if ON, goes high for 0.5 seconds at start of the program.

Unless you are using the pins for some other purpose, it is fine to leave EYES and TRIGGER_OUT set to ON, even if you are not using them.

[PINS]

This section specifies which GPIO pins are used for the servo control channel (JAW_PIN), the input trigger (PIR_PIN), the LED "eyes" output (EYES_PIN), and the trigger output (TRIGGER_OUT_PIN).

Running the Program: To run the program, you can start main.py from the command line by going to the chatterpi directory and typing:

```
$ python3 main.py
```

There will be a large number of ALSA warnings and errors displayed on the screen. Believe it or not, this is normal, and ChatterPi will run properly. Control-C will stop the program.

The Control Panel

Running `controlPanel.py` allows you to edit the `config.ini` file using a GUI. A screenshot of the GUI is shown in Figure 3.

To edit values, simply type the new values into the appropriate fields and select `SAVE`. Each time you open the control panel, the current values are stored. If you wish to return all settings to what they were at the beginning of the editing session, select `RESET`.

The parameters are case sensitive. alphabetic values should always be entered in all caps.

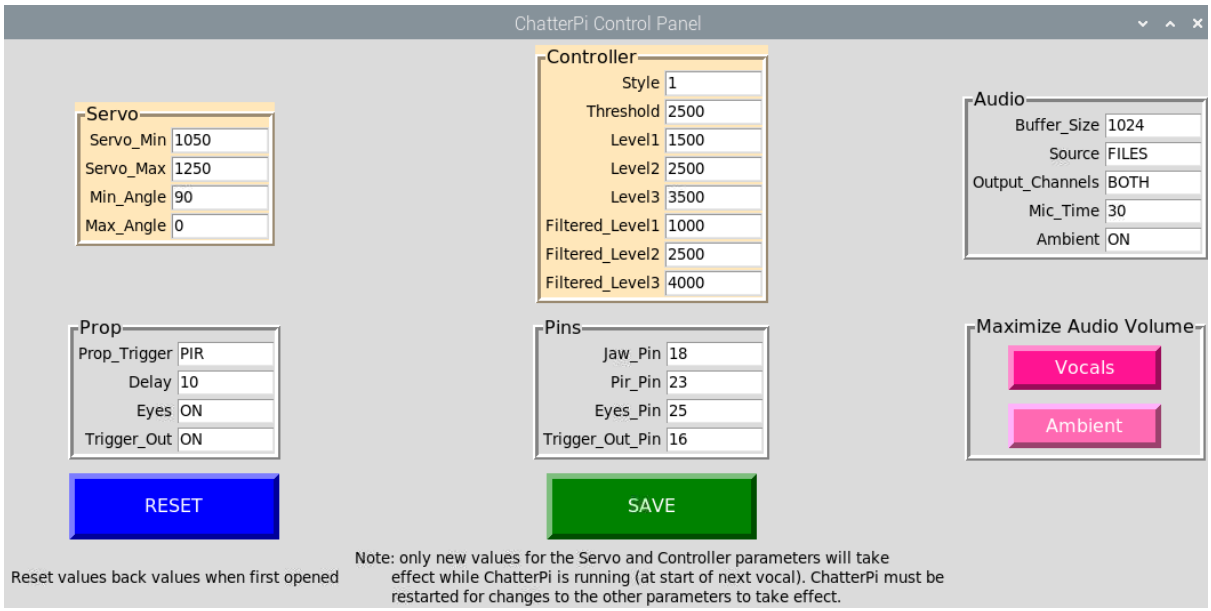

Figure 3. Screenshot of the Configuration Control Panel

Maximizing the Volume of Audio Files: If you are playing back multiple files, you may have issues with differing volume levels. The Control Panel provides an easy interface to a utility program that will open each vocal or ambient sound file (depending on your selection), analyze it, and save it back with the volume levels raised to the maximum possible without introducing clipping or distortion. Select either `Vocals` or `Ambient` to maximize the volumes of each of the files in the selected subdirectory.

Changing Settings During Execution: Starting with this version, you can edit some of the settings in the config.ini file (either directly or using the Control Panel) while ChatterPi is running, and they will take effect the next time a vocal file is triggered, rather than requiring you to stop and restart the program. This should make it quicker and easier to tune the servo settings to your vocal files. The parameters that can be changed and take effect during runtime are the Servo and Controller parameters, which are shown in yellow on the Control Panel.

Run Automatically On Boot

Your ChatterPi is hopefully now up and running and you have adjusted the parameters. If you want to start ChatterPi from the Pi's command line or GUI, you're good to go. But for use in a prop, you may want to configure your Pi to automatically begin running ChatterPi on boot, with no need for user interaction. To do this, move the `chatterpi.service` file in the `chatterpi` directory to `/lib/system/system`. Then type:

```
$ sudo systemctl enable chatterpi.service
```

This tells the operating system to run `chatterpi` at the end of the boot sequence. See https://www.raspberrypi.org/documentation/linux/usage/systemd.md for more information and commands you can use form the command line to restart or stop ChatterPi when it is set to run automatically on boot.

Project Roadmap

This version, 0.9, includes all the features currently planned for ChatterPi. That said, there are two additional features that might be added at a later time (or if anyone cares to add them to this open source project:

- The ability to use .mp3 files. Simply playing MP3 files on a Raspberry Pi is easy, but they must be processed in real-time as a stream to drive the servo controller.
- Add drop down pick lists for many of the options in the control panel, and allow lower case values to be entered, auto-correcting to upper case.
- Add the ability to start and stop the execution of ChatterPi from the control panel.

Bug Reporting

To report a bug, make a suggestion, or ask a question, please go to the GitHub repository for the project (https://github.com/ViennaMike/ChatterPi) and open an Issue. To do so, first click on the issues tab, and then use the green "New Issue" button. It's a good idea to first browse through or search other reported issues to see if someone has already reported the same issue or asked the same question. You can then add comments or suggestions to existing issues, rather than opening a new, duplicative issue.

Appendix A: Software Overview

The code is open source and published on GitHub (https://github.com/ViennaMike/ChatterPi). Knowledge or understanding of the software code is not required to operate or use the ChatterPi. Those who are not interested can skip this section of the documentation.

The ChatterPi package consists of eight Python 3 modules and one configuration file, as shown in Figure 4.

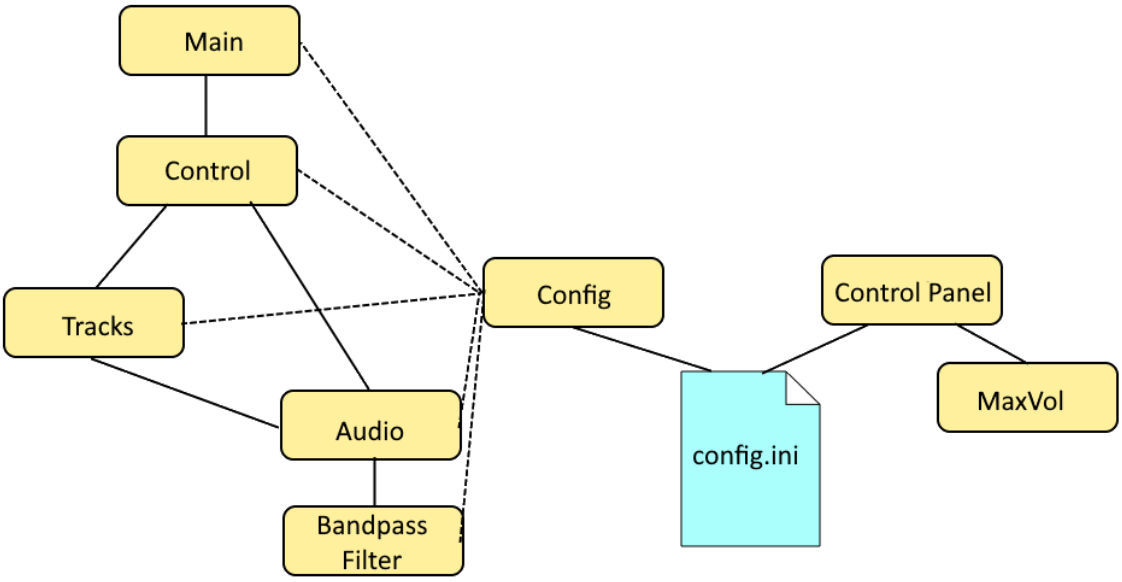

Figure 4. The eight python modules and one configuration file in ChatterPi

The configuration file, `config.ini`, holds all of the user selectable parameters, including which pins are used for which functions, whether the audio source is the microphone input or stored .wav files, which servo control mode should be used, and the servo threshold levels. The `config.py` program simply reads these values and makes them available in memory during runtime.

The `main.py` program essentially simply loads the configuration parameters on start-up and calls `control.py`. The functions in `control.py` are not folded into `main.py` in order to avoid a submodule having to import the main program, which can be problematics.

Most of the processing occurs in the `control.py` and `audio.py` modules. The `control.py` program handles most of the triggering (either a timer, an external trigger such as a PIR, or immediately upon startup, with the method specified in the `config.ini` file. It uses the GPIO Zero and PiGPIO libraries to monitor the triggering sensor and send output to the output trigger and led pins. PiGPIO is used as the GPIO layer underneath GPIO Zero because it uses DMA control for the Pulse Width Modulation (PWM) control used to the control the servo. Some other libraries, including the default one used by GPIO Zero, use software PWM, which is adequate for tasks such as controlling the brightness of LEDs, but not precise enough for servo control.

Unless the triggering mode is START, the file enters an infinite loop waiting for either a timer to expire (TIMER mode) or the external trigger to be generated (PIR mode). The wait functions meet the requirements and during development, interrupt driven approaches interfered with the audio output, probably due to timing conflicts. In TIMER mode, the timer is restarted after either the audio file finishes playing (if the source is FILES) or after a configurable pre-set time (if the source is MICROPHONE).

When triggered, an event handler is called that, depending upon the settings, sets off the TRIGGER_OUT to trigger another prop or device and turns on the LED eyes or other low power device. Then, if the audio source is FILES, it will call `tracks.py`, which will select the next .wav file to be played and call `audio.py`, passing the name of the .wav file to be played. If the audio source is MICROPHONE, `audio.py` is called without passing a file name. When the call to audio.py returns, the event handler turns the LED eyes off and returns.

Audio playback, audio analysis, and servo control are all performed by the `audio.py` module. It defines one class, AUDIO. When the audio.play function is called, it checks on whether the audio source is MICROPHONE or FILES and opens a PyAudio stream appropriately. The stream call runs in a separate thread (this is automatically handled by PyAudio). For each chunk of the input stream, a callback function is called. This callback function is where the audio stream volume is analyzed. The average volume for each chunk is calculated, and the servo is commanded to the appropriate position based on that average volume and the threshold levels that the user has specified in the config file. The wave library is used to read the wave files from storage, and the struct library is used to help deconstruct the wave data to calculate volume and to help to separately analyze the left and right channels for stereo files. The number of levels, the specific thresholds, and whether a bandpass filter is applied before calculating the volume is based on the STYLE setting set by the user in the config file. In addition to the official documentation, I found a slide presentation, Introduction to PyAudio, by Jean Cruypenynck to be very helpful.

If the STYLE is set to 2, then `bandpassFilter.py` is called to process the digital audio stream and return a modified stream with the bandpass filter applied. The program is very short and simple. It uses two functions from the scipy signal processing library to filter out audio input below 500 Hz and over 2500 Hz. No bandpass filter is applied for STYLE 0 or STYLE 1.

When AMBIENT is set to ON, the ambient playback function in `audio.py` must also monitor for triggering events (either the timer or sensor), since it needs to interrupt itself and pass control back to `control.py` when such an event occurs.

The `config.ini` file can either be edited directly or through a GUI program named `controlPanel.py`. If the servo or controller subset of parameters are changed during execution, the changes will be reflected the next time a vocal track is triggered. Other changes will not take effect until after ChatterPi is stopped and then restarted.

`maxVol.py` is a utility program that can be launched from the control panel. It reads and analyzes each wave file in either the vocals or ambient subdirectories and writes them back with the volume levels increased to the maximum possible without clipping or distortion.

Appendix B: Contents of `config.ini` File

This is a sample of the config.ini file that you will find in the chatterpi directory. To modify any of the settings, simply edit and then save the config.ini file. The new settings will take effect the next time you run ChatterPi. Note: Some of the settings shown here may not match the settings in your download.

```
[SERVO]
; Adjust as needed. For my skull, full open is 1100 microseconds,
; full close is 1500
SERVO_MIN = 1100
SERVO_MAX = 1500
; Set the movement range of your servo in degrees
; If closed value is higher than open value, e.g., jaw closed= 90
; and jaw open = 0, then set min angle to the higher value and
; max angle to the lower value
MIN_ANGLE = 90
MAX_ANGLE = 0

[CONTROLLER]
; Styles 0, 1, 2 are single threshold, multi-level, and band-filtered
; multi-level styles for determining when and how much to move the ; ;
; jaw.
STYLE = 2
; Threshold audio volume for style 0
THRESHOLD = 8000
; Threshold audio volumes for style 1
LEVEL1 = 1000
LEVEL2 = 10000
LEVEL3 = 13000
; Threshold audio volumes for style 2
FILTERED_LEVEL1 = 20
FILTERED_LEVEL2 = 500
FILTERED_LEVEL3 = 2000

[AUDIO]
; Pyaudio buffer size - DO NOT CHANGE, LEAVE AT 1024
BUFFER_SIZE = 1024
; Audio source, either FILES or MICROPHONE
SOURCE = FILES
; For stereo .wav files, which channel or channels should be sent
; to the speakers. Valid values are "LEFT" or "BOTH"
; Use LEFT if the right channel is a tones channel to control servo,
; rather than audio to be played.
OUTPUT_CHANNELS = BOTH
; Number of seconds to keep input open when source is MICROPHONE and
; prop trigger is either PIR or TIMER (for FILES, the software knows
; when each file is finished playing
MIC_TIME = 20
```

```
; OFF does not play ambient tracks between triggering, ON does
; (not yet implemented)
AMBIENT = OFF

[PROP]
; Determines what triggers the audio: TIMER, PIR, or START
; PIR mode triggers on external signal (often a PIR sensor)
; When source is FILES, must set to TIMER or PIR. If source is
; MICROPHONE, may also set to START, which starts immediately and
; doesn't turn off
PROP_TRIGGER = TIMER
; Delay between end of one activation and start of another if in TIMER
; mode, in seconds. Recommend a value between 0 and 600 seconds
DELAY = 10
; ON to light eyes when playing vocals, otherwise OFF
EYES = ON
; Control a trigger out to trigger another prop or device when
; ChatterPi is triggered. ON to set trigger_out pin high for
; 0.5 seconds when ChatterPi is triggered.
TRIGGER_OUT = ON

[PINS]
; The GPIO pin assignments for servo control, PIR or other triggering
; input (if used), a pin to light up LED eyes when the prop is
; triggered, and the pin that briefly triggers
; high when the prop is triggered, that can be used to trigger another
; prop or device
JAW_PIN = 18
PIR_PIN = 23
EYES_PIN = 25
TRIGGER_OUT_PIN = 16
```