CyanBot python-ev3dev Documentation

Release 2.0

Author: Sai Tanuj Karavadi

6/1/2023

Chapter 0

Table of Contents

Chapter 1

What is EV3DEV?

ev3dev is an open-source operating system for LEGO MINDSTORMS EV3 robotics kits. It allows users to program and control their EV3 robots using a variety of programming languages, including Python, C++, and Java, and provides access to a wide range of sensors, motors, and other components. ev3dev is designed to be easy to use and flexible, with a large community of users and developers contributing to its ongoing development and

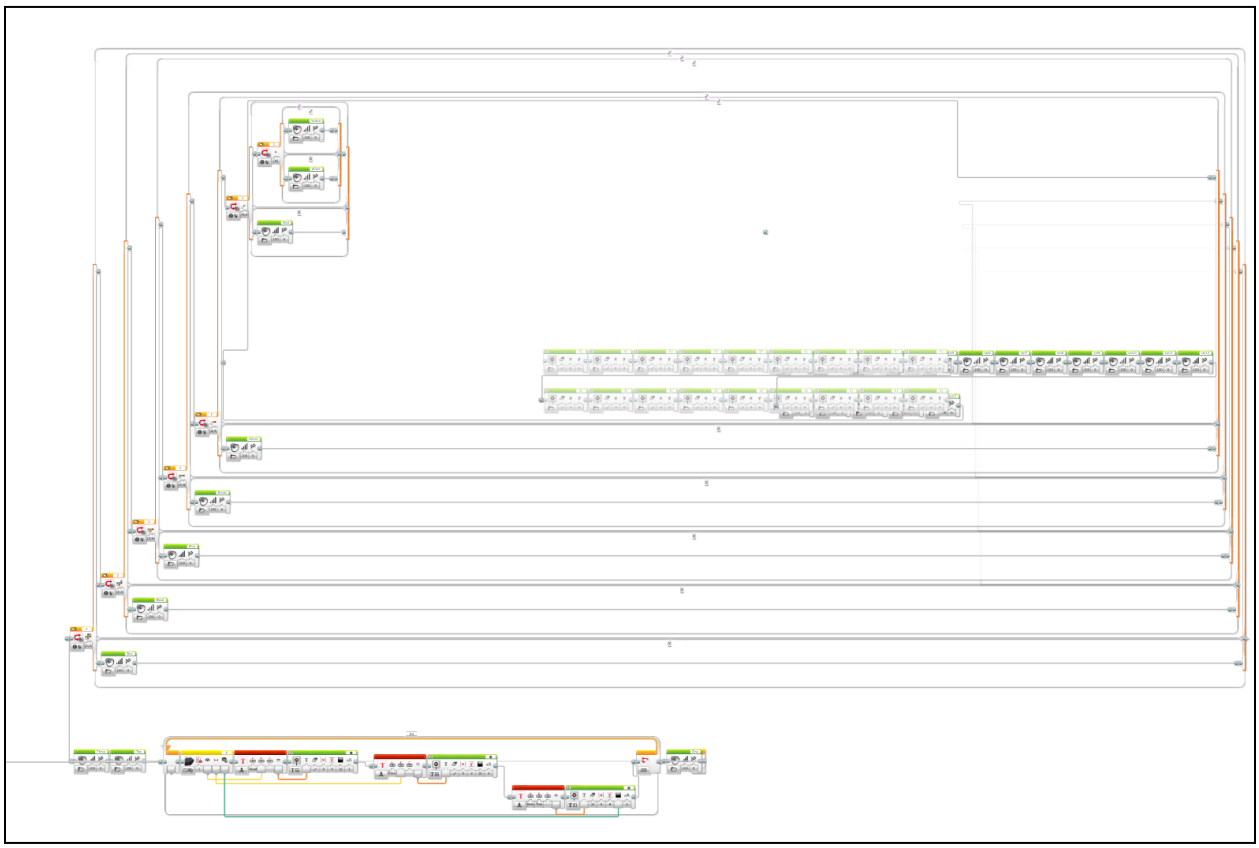
improvement. It also offers advanced features such as remote control, wireless networking, and support for custom hardware, making it a powerful tool for building sophisticated and innovative robotics projects. I am using the python version of ev3dev, running python using Visual Studio Code (VS Code). Using the ev3dev addon and libraries, I am able to program the EV3 brick through my computer.

Chapter 2

Why am I using EV3Dev?

There are several reasons why I am using ev3dev. For one, the default programming language of the Ev3 brick is Mindstorms blocks, which is quite severely limited and lacking in structure. Also, blocks puts severe restrictions on what you can and cannot do. Though ev3dev supports a variety of languages; anything from Lua, C libraries, Java, Go, Rust, and many more

(https://www.ev3dev.org/docs/programming-languages/ for more language options). But I chose python as it was the language I was most familiar with and I was not planning on learning Rust anytime this year. With ev3dev, the possibilities are (almost) endless. I will come back to the problems in another chapter. With ev3dev, I was able to use the webcam, the microphone inside the webcam, display images, use all the sensors and motors, all while not using that much storage inside the ev3 brick. Also, one of the most crucial features of Visual Studio code is saving on Github. With the default Mindstorms programming language, I could not save it on github since it is proprietary and Github does not support it. Here are some images of the default Mindstorms Programming Language:



As you can see, it takes quite a lot of blocks to program something as simple as a color detecting program. In comparison, here is the same program in python using ev3dev on Visual Studio Code:

ev3 > tests > code > ColorName.py > ...

```python
#!/usr/bin/env python3
from ev3dev2.sensor import INPUT_2
import os
from ev3dev2.motor import OUTPUT_B
import time
import ev3dev2.sensor as sensor
import ev3dev2.motor as motor
import ev3dev2.sound as sound
from ev3dev2.sound import Sound
from ev3dev2.sensor.lego import ColorSensor
color_sensor = ColorSensor(INPUT_2)
m = motor.LargeMotor(OUTPUT_B)
s = sound.Sound()
color_names = {
    1: 'Black',
    2: 'Blue',
    3: 'Green',
    4: 'Yellow',
    5: 'Red',
    6: 'White',
    7: 'Brown'
}
while True:

    color_value = color_sensor.color
    if color_value in color_names:
        m.on_for_rotations(-50, 0.25)
        s.speak(color_names[color_value])
        time.sleep(0.15)
        m.on_for_rotations(50, 0.25)


```

It takes much less effort to program on ev3dev python. Not only is it faster and easier to program on ev3dev, but it actually performs better on ev3dev as well. The color detection algorithm I have made here is more reliable than that of the Mindstorms program.

In conclusion, ev3dev is much easier and faster to program on. Once you install ev3dev there is actually a completely different UI (User Interface) as well on the brick meaning it has a more streamlined process.

Chapter 3

What is an EV3?

LEGO Mindstorms EV3 is a robotics kit designed for educational and hobbyist purposes. It consists of a programmable brick, motors, sensors, and LEGO Technic building elements that allow users to build and program their own robots. The EV3 brick is the central component of the kit,

providing a microcontroller, a color LCD screen, and ports for connecting motors, sensors, and other accessories.The kit includes several sensors, such as touch, color, ultrasonic, and gyroscopic sensors, which allow the robot to interact with its environment and perform tasks such as detecting obstacles, following lines, and measuring distances. The motors included in the kit can be used to drive the robot's movement, control arms, and other mechanical components.

One of the key features of the Mindstorms EV3 kit is its programmability. The EV3 brick can be programmed using a variety of programming languages, including a drag-and-drop programming language called EV3-G, a text-based language called RobotC, and popular programming languages like Python, Java, and C++. This flexibility allows users to program their robots at different levels of complexity, from simple behaviors to advanced algorithms.

Another feature of the EV3 kit is its versatility. Users can create a wide range of robots, from simple models to complex machines, by combining different sensors, motors, and building elements. The kit also supports wireless

communication and can be connected to other devices, such as smartphones or computers, for remote control or data exchange.

The Ev3 is a decade old now, so while it was powerful when it released, it has the processing power of a calculator meaning there is quite a bit of limitations to its usage. Thats why many buy a Raspberry Pi and use the ev3 addon to plug in motors, and still be able to program them.

The hardware on the EV3 is amazing and will talk about it in the next chapter.

CHAPTER 4

THE HARDWARE ON CYANBOT.

I am using quite a bit of hardware on the CyanBot. The hardware list is:

These parts are connected to the Lego Mindstorms EV3:

- 1 EV3 Programmable Brick
- EV3 Motors:
    - 2 EV3 Large Motors
    - 1 EV3 Medium Motor
- 1 Logitech C310 Webcam with Microphone
- 1 EV3 Remote Control
- EV3 Sensors:
    - 1 EV3 Color Sensor

    - 1 EV3 Touch Sensor
    - 1 Ev3 Infrared Sensor (IR Sensor)

These parts are not connected to the Lego Mindstorms EV3, but are instead controlled by a remote control, meaning that these are non-programmable:

- 1 Power Functions Battery Box
- 1 Power Funtions Large Motor
- 2 Power Function Medium Motors
- 1 Power Function Small Motor
- 1 Power Function Switch
- 1 Power Function IR Reciever

- 1 Power Function Remote Control

These parts are Pneumatics parts, meaning they are not controlled by motors

(Though the pump is using a small Power Function motor):

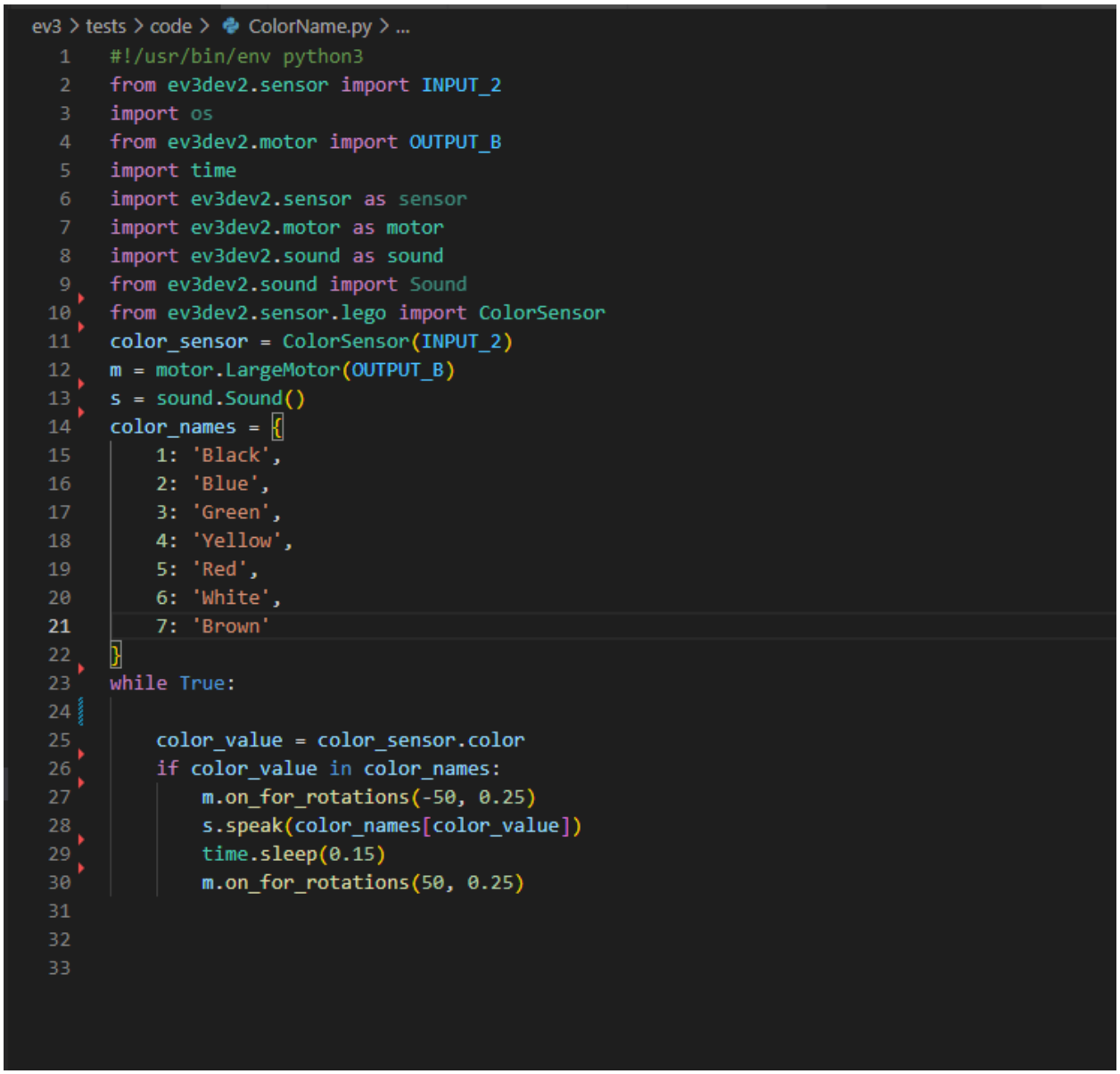
- 1 Medium Pneumatic Cylinder
- 1 Small Pneumatic Pump
- 2 Large Pneumatic Cylinders

Chapter 5

The Color Sensor Program

The Color Sensor Program uses the Color Sensor in the EV3 to detect color. I use the color sensor to measure the color value that is inputted through the sensor, then I use the ev3dev sound class to output the name. I created 2

different programs, one with a touch sensor one without. Here is the program without the touch sensor called ColorName.py:



What this program does is when the program runs, it says the color that it detects. And after that it just loops meaning that the program will not stop until we take the batteries out. The pre-programmed colors in the EV3 are:

```
    1: 'Black',
    2: 'Blue',
    3: 'Green',
    4: 'Yellow',
    5: 'Red',
    6: 'White',
    7: 'Brown'
```

So these colors and another color that is not mentioned here, ‘no color’ are the pre-programmed colors that the EV3 sensor can detect. Theoretically, it can detect 16 million colors though what that requires is a program that outputs RGB values (Red, Green, and Blue) so it would not output a name but rather a number. Also for both programs it moves the mouth so that it seems like it is speaking when it says the color out loud.

Here is the second ColorName program called ColorNameTouch.py:

ev3 > tests > code > ColorNameTouch > ...

```python
#!/usr/bin/env python3
import time
import ev3dev2.sensor as sensor
import ev3dev2.motor as motor
import ev3dev2.sound as sound
from ev3dev2.sound import Sound
from ev3dev2.sensor.lego import ColorSensor
from ev3dev2.motor import OUTPUT_B
from ev3dev2.sensor import INPUT_2
from ev3dev2.sensor import INPUT_4
color_sensor = ColorSensor(INPUT_2)
touch_sensor = sensor.TouchSensor(INPUT_4)
m = motor.LargeMotor(OUTPUT_B)
s = sound.Sound()

color_names = {1: 'Black', 2: 'Blue', 3: 'Green', 4: 'Yellow', 5: 'Red', 6: 'White', 7: 'Brown'}

while not touch_sensor.is_pressed:
    color_value = color_sensor.color
    if color_value in color_names:
        m.on_for_rotations(-50, 0.25)
        s.speak(color_names[color_value])
        time.sleep(0.15)
        m.on_for_rotations(50, 0.25)

```

What this program does is the same as above but instead of looping it will only loop until the touch sensor is pressed, and once the touch sensor is pressed the program will terminate. Meaning that I do not have to take the batteries out.

Chapter 6

The Webcam Programs

There are 3 different Webcam programs that I have created. We will look over each one.

For all 3 of these programs, we use something called Subprocess.run. What Subprocess is, is it runs commands in the SSH terminal using a programming language called Bash.

Here is a Subprocess.call line:

```
import subprocess
subprocess.call(['fswebcam', '-r', '100x100',
'--no-banner', filename])
```

So all 3 of these programs use a linux-based webcam capturing process called fswebcam. Fswebcam is a debian-linux-based software that uses the webcam and sets a resolution, and file save path as well.

Furthermore, subprocess.call can also be used for many other programs, not only for fswebcam. We will show more subprocess.call commands later on.

So keep in mind this line for the 4 webcam programs.

First up is the Webcam.py program, the first one I ever created that actually worked:

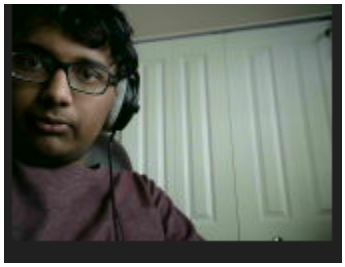
ev3 > tests > code > Webcam.py > ...

```
import ev3dev2.fonts as fonts
#from PIL import Image
#from ev3dev2.display import Display


import subprocess
import time

from ev3dev2.sound import Sound

import ev3dev2.sensor as sensor
import subprocess
sound = Sound()
screen = ev3.Display()
screen.clear()

sound.speak('Hello!')
sound.speak('Click the Button to Capture an Image')
while True:
    screen.clear()
    screen.draw.text((0,0), 'Click the Button', font=fonts.load('ncenI24'))
    screen.draw.text((0,6), 'to capture', font=fonts.load('ncenI24'))
    screen.draw.text((0,12), 'an image', font=fonts.load('ncenI24'))



    ts = TouchSensor(INPUT_4)
    if ts.is_pressed:
        timestamp = time.strftime('%Y%m%d-%H%M%S')
        filename = 'image-{}.jpg'.format(timestamp)
        subprocess.call(['fswebcam', '-r', '100x100', '--no-banner', filename])
        sound.speak('Image Captured!')
    screen.update()
```

What this Webcam.py program does is it uses the touch sensor again. Each time you press the touch sensor, it saves the image as a .jpg formatted image directly onto the ev3 brick. Not only that, but due to the fact that we have to specify a filename to save it as, each time we click the button it would overwrite the original image.jpg that we named it. Instead, a clever solution to that problem was to save each image with the timestamp after the image name. Using:

```
import time
        timestamp =
time.strftime('%Y%m%d-%H%M%S')
```

We are able to import the time and save the image with a slew of numbers after it. Once we terminate the program we can use a computer plugged into the ev3 and get an image out of it. Here is an example of such image:



The next program is called WebcamDisplay.py:

ev3 > tests > code > WebcamDisplay.py > ...

```python
#!/usr/bin/env python3

import time
import ev3dev2.auto as ev3
import ev3dev2.fonts as fonts
import subprocess
from ev3dev2.motor import LargeMotor, OUTPUT_A, OUTPUT_B, SpeedPercent, MoveTank
from ev3dev2.sensor import INPUT_4
from ev3dev2.sensor.lego import TouchSensor
from ev3dev2.led import Leds
from ev3dev2.sound import Sound
from ev3dev2.button import Button
btn = Button()
sound = Sound()
screen = ev3.Display()
screen.clear()

sound.speak('Hello!')
sound.speak('Click the Button to Capture an Image')
screen.clear()
screen.draw.text((0,0), 'Click the Button', font=fonts.load('ncenI24'))
screen.draw.text((0,12), 'to capture', font=fonts.load('ncenI24'))
screen.draw.text((0,24), 'an image', font=fonts.load('ncenI24'))

while not btn.down:
    ts = TouchSensor(INPUT_4)
    if ts.is_pressed:
        timestamp = time.strftime('%Y%m%d-%H%M%S')
        filename = 'image.bmp'
        subprocess.call(['fswebcam', '-r', '160x90', '--no-banner', filename])
        time.sleep(5)
        subprocess.run(['sudo', 'service', 'udev', 'restart'])
        time.sleep(5)
        subprocess.run(['sudo', 'fbi', '-T', '1', '-noverbose', '-a', '/home/robot/image.bmp'])
        time.sleep(10)
#To Run this program, in the ssh do: sudo python3 /home/robot/ev3/ev3/tests/code/WebcamDisplay.py
#Or whatever the path of the pthon program is
```

Now what WebcamDisplay does is pretty cool. Once you hit the touch sensor and capture an image, it refreshes and then displays the image directly onto the ev3 screen!

This uses 2 new subprocess.run lines:

```
import subprocess
        subprocess.run(['sudo', 'service',
'udev', 'restart'])
```

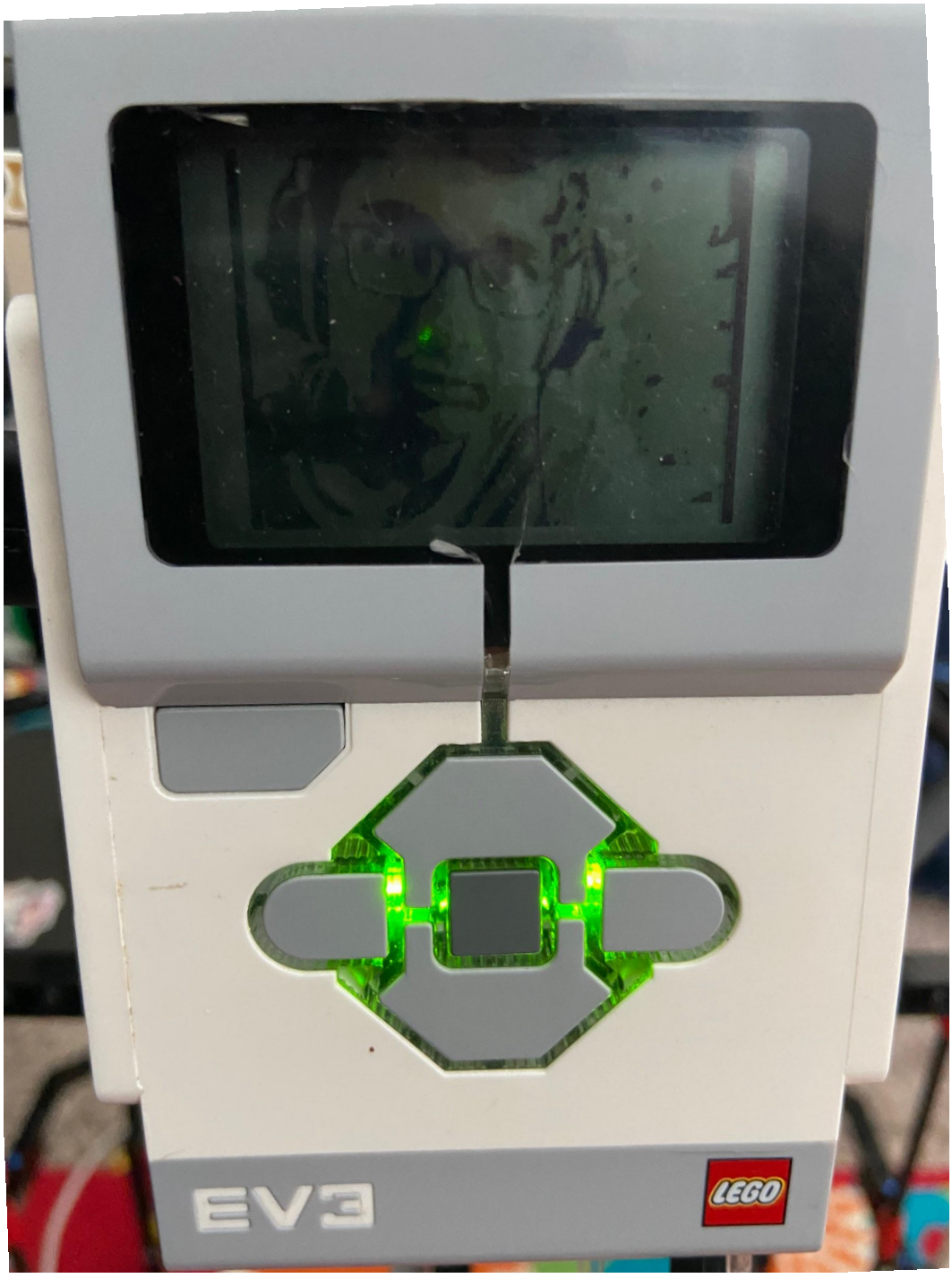
And:

```
import subprocess
        subprocess.run(['sudo', 'fbi', '-T', '1',
'-noverbose', '-a', '/home/robot/image.bmp'])
```

What the first one does is after it captures the image, it refreshes the directory of the ev3 storage so that the program can access the lates image.jpg file.

What the second one does is after it refreshes, it uses the fbi debian-linux-based software to project it onto the ev3 screen. The characters after that is the path of the image, and the projection fittings of the ev3 screen.

Here is what a captured image looks like on the ev3 screen:


EV3
LEGO

As you can see, the captured image is not of the best resolution, Due to the limitations of the ev3 display, it cannot project high quality images.

The default resolution of the ev3 screen is 178x128 pixels, so worse than the lowest resolution that youtube offers.

Not only that, but it is monochrome. You get a few shades of grey, black, and green. That is it. It is basically a non-color graphing calculator display.

Finally, we have the WebcamStream.py program:

ev3 > tests > code > WebcamStream.py > ...

```python
#!/usr/bin/env python3

import time
import ev3dev2.auto as ev3
import ev3dev2.fonts as fonts
import subprocess
from ev3dev2.motor import LargeMotor, OUTPUT_A, OUTPUT_B, SpeedPercent, MoveTank
from ev3dev2.sensor import INPUT_4
from ev3dev2.sensor.lego import TouchSensor
from ev3dev2.led import Leds
from ev3dev2.sound import Sound
from ev3dev2.button import Button
from multiprocessing import Process

btn = Button()
sound = Sound()
screen = ev3.Display()
screen.clear()


sound.speak('Hello! This is a streaming program. This will stream the photos to the ev3 screen.')



def capture_image():
    while not btn.down:
        subprocess.call(['fswebcam', '-r', '50x50', '--no-banner', 'image.bmp'])


def display_image():
    while not btn.down:
        subprocess.Popen(['sudo', 'fbi', '-T', '1', '-noverbose', '-a', '/home/robot/image.bmp'])


if __name__ == '__main__':
    p1 = Process(target=capture_image)
    p2 = Process(target=display_image)
    p1.start()
    p2.start()
    p1.join()
    p2.join()

#To Run this program, in the ssh do: sudo python3 /home/robot/ev3/ev3/tests/code/WebcamDisplay.py
#Or whatever the path of the pthon program is
# Ctrl+C --> Ctrl+Shift+V
# sudo python3 /home/robot/ev3/ev3/tests/code/WebcamStream.py

```

What the WebcamStream.py program does is quite cool as well. It takes an image every 3-4 seconds and displays it on the ev3 screen. Kind of like a live-stream. Due to the limitations of the Ev3 screen and the lack of access to the highly coveted Python PIL (pillow) Python Image Library, there is restrictions. But this basically tries to combine it all using:

```
from multiprocessing import Process
    p1 = Process(target=capture_image)
    p2 = Process(target=display_image)
```

CHAPTER 7

THE RECORDING PROGRAMS

There are 2 different Recording programs that I have created. We will look over each one.

For both of these programs, we use something called Subprocess.run. What Subprocess is, is it runs commands in the SSH terminal using a programming language called Bash.

Here is a Subprocess.call line:

```
import subprocess
        cmd = ['arecord', '-D', 'hw:1,0', '-f',
'S16_LE', '-c', '1', '-r', '44100',
'/home/robot/myvoice.wav']
        recording_process = subprocess.Popen(cmd)
        is_recording = True
```

So both of these programs use a linux-based webcam microphone capturing process called Arecord. Arecord is a debian-linux-based software that uses the webcam microphone and sets a frequency and wavelength to record it in.

Furthermore, subprocess.call can also be used for many other programs, not only for fswebcam. We will show more subprocess.call commands later on.

So keep in mind this line for both the webcam programs.

First up, we have the Recording.py Program:

ev3 > tests > code > Recording.py > ...

```python
#!/usr/bin/env python3
import time
from ev3dev2.motor import LargeMotor, OUTPUT_A, OUTPUT_B, SpeedPercent, MoveTank
from ev3dev2.sensor import INPUT_4
from ev3dev2.sensor.lego import TouchSensor
from ev3dev2.led import Leds
import ev3dev2.auto as ev3
from time import sleep
import ev3dev2.fonts as fonts
import subprocess
import time
from ev3dev2.sound import Sound
import ev3dev2.sensor as sensor
import subprocess
sound = Sound()


ts = TouchSensor(INPUT_4)
is_recording = False

while True:
    if ts.is_pressed and not is_recording:
        sound.speak('Recording started!')
        cmd = ['arecord', '-D', 'hw:1,0', '-f', 'S16_LE', '-c', '1', '-r', '44100', '/home/robot/myvoice.wav']
        recording_process = subprocess.Popen(cmd)
        is_recording = True
    elif ts.is_pressed and is_recording:

        recording_process.terminate()
        sound.speak('Recording stopped!')
        is_recording = False
    sleep(0.1)
#sudo python3 /home/robot/ev3/ev3/tests/code/Recording.py
```

What the Recording.py program does is it uses the touch sensor, and once you press the touch sensor it records audio using the webcam microphone. Once you press the touch sensor again, it stops the recording and saves it directly onto the ev3 brick as a .wav file. You can plug the ev3 to the computer and download the audio file and play it back using VLC or any popular playback software.

The RecordingPlayback.py Program:

ev3 > tests > code > RecordingPlayback.py > ...

```python
#!/usr/bin/env python3
import time
from ev3dev2.motor import LargeMotor, OUTPUT_A, OUTPUT_B, SpeedPercent, MoveTank
from ev3dev2.sensor import INPUT_4
from ev3dev2.sensor.lego import TouchSensor
from ev3dev2.led import Leds
import ev3dev2.auto as ev3
from time import sleep
import ev3dev2.fonts as fonts
import subprocess
import time
from ev3dev2.sound import Sound
import ev3dev2.sensor as sensor
import subprocess
sound = Sound()


ts = TouchSensor(INPUT_4)
is_recording = False

while True:
    if ts.is_pressed and not is_recording:

        cmd = ['arecord', '-D', 'hw:1,0', '-f', 'S16_LE', '-c', '1', '-r', '44100', '/home/robot/myvoice.wav']
        sound.speak('Recording started!')
        recording_process = subprocess.Popen(cmd)
        is_recording = True
    elif ts.is_pressed and is_recording:
        recording_process.terminate()
        sound.speak('Recording stopped!')
        is_recording = False
        break  # exit the loop

    sleep(0.1)

# Play the recorded audio file if the touch sensor is pressed again
if ts.wait_for_pressed():
    subprocess.run(['sudo', 'service', 'udev', 'restart'])
    sound.speak('Playing recorded audio!')
    cmd = ['aplay', '-V', '100', '/home/robot/myvoice.wav']

    subprocess.Popen(cmd).wait()
#sudo python3 /home/robot/ev3/ev3/tests/code/RecordingPlayback.py
```

What the WebcamRecording.py program does is pretty cool as well. It does the same thing as the Recording program:

> What the Recording.py program does is it uses the touch sensor, and once you press the touch sensor it records audio using the webcam microphone. Once you press the touch sensor again, it stops the recording and saves it directly onto the ev3 brick as a .wav file. You can plug the ev3 to the computer and download the audio file and play it back using VLC or any popular playback software.

But once you touch the touch sensor again, it plays it back. The way this is achieved is using something called mpg123.

What mpg123 is a popular command-line audio player and decoder that is available for multiple platforms, including Linux, Windows, and macOS. It is capable of playing a variety of audio file formats, including MP3, Ogg Vorbis, and WAV.

Mpg123 uses a highly optimized audio decoding engine that is designed to run efficiently on a range of hardware, from low-powered embedded devices to high-performance servers. It can decode audio in real-time or pre-decode

audio files for faster playback. In addition to playing audio files from the command line, mpg123 can be used as a library in other software projects. This allows developers to build audio playback functionality into their applications using the highly optimized decoding engine provided by mpg123.

Mpg123 includes a range of command-line options that allow users to customize the audio playback experience. These options include things like setting the playback volume, adjusting the playback speed, and modifying the audio equalizer.

We use the subprocess.run for mpg123 as well:

```
import subprocess
    subprocess.run(['sudo', 'service', 'udev',
'restart'])
    sound.speak('Playing recorded audio!')
    cmd = ['aplay', '-V', '100',
'/home/robot/myvoice.wav']
```

We refresh the directory of the ev3 to access the myvoice.wav file, then play it back using mpg123 on maximum volume.

Chapter 8

The Display Programs

There are a few display programs. These use fbi as well. Some play the BlendS intro while one plays Seisyun Complex. These programs are a heavy

heavy work in progress as I am figuring out how to put music and video at same time. Also ev3 slow so not much video but mostly a slideshow.

SeisyunComplex.py:

```
#!/usr/bin/env python3
import subprocess
import time
import threading
from ev3dev2.button import Button

btn = Button()


mp3_file_path =
"/home/robot/ev3/seisyun_complex.mp3"
subprocess.Popen(["mpg123", "-b", "8192",
mp3_file_path])


        #Ctrl+C --> Ctrl+Shift+V into SSH
Terminal
        # sudo python3
/home/robot/ev3/ev3/tests/code/SeisyunComplex.py
```

Seis.py:

```
import time
import ev3dev2.auto as ev3
import ev3dev2.fonts as fonts
import subprocess
from ev3dev2.motor import LargeMotor, OUTPUT_A,
OUTPUT_B, SpeedPercent, MoveTank
from ev3dev2.sensor import INPUT_4
from ev3dev2.sensor.lego import TouchSensor
from ev3dev2.led import Leds
from ev3dev2.sound import Sound
from ev3dev2.button import Button

btn = Button()
sound = Sound()
screen = ev3.Display()
```

```
screen.clear()

sound.speak('Hello!')
sound.speak('Make a sound to capture an image')
screen.clear()
screen.draw.text((0,0), 'Make a sound',
font=fonts.load('ncenI24'))
screen.draw.text((0,12), 'to capture',
font=fonts.load('ncenI24'))
screen.draw.text((0,24), 'an image',
font=fonts.load('ncenI24'))

while not btn.down:
    # Wait for sound event
    sound.wait()
    sound.speak('capturing image')
    timestamp = time.strftime('%Y%m%d-%H%M%S')
    filename = 'image.bmp'
    subprocess.call(['fswebcam', '-r', '160x90',
'--no-banner', filename])
    time.sleep(5)
    subprocess.run(['sudo', 'service', 'udev',
'restart'])
    time.sleep(5)
```

```
    subprocess.run(['sudo', 'fbi', '-T', '1',
'-noverbose', '-a', '/home/robot/image.bmp'])
    time.sleep(10)

# sudo python3
/home/robot/ev3/ev3/tests/code/Seis.py
```

BlendS,py:

```
#!/usr/bin/env python3
import subprocess



subprocess.run(['sudo', 'fbi', '-T', '1',
'-noverbose', '-nocomments', '-a',
'/home/robot/ev3/mp4_000/mp4_000.jpg'])
```

```
subprocess.run(['sudo', 'fbi', '-T', '1',
'-noverbose', '-nocomments', '-a',
'/home/robot/ev3/mp4_000/mp4_001.jpg'])
subprocess.run(['sudo', 'fbi', '-T', '1',
'-noverbose', '-nocomments', '-a',
'/home/robot/ev3/mp4_000/mp4_002.jpg'])
subprocess.run(['sudo', 'fbi', '-T', '1',
'-noverbose', '-nocomments', '-a',
'/home/robot/ev3/mp4_000/mp4_003.jpg'])
subprocess.run(['sudo', 'fbi', '-T', '1',
'-noverbose', '-nocomments', '-a',
'/home/robot/ev3/mp4_000/mp4_004.jpg'])
subprocess.run(['sudo', 'fbi', '-T', '1',
'-noverbose', '-nocomments', '-a',
'/home/robot/ev3/mp4_000/mp4_005.jpg'])
subprocess.run(['sudo', 'fbi', '-T', '1',
'-noverbose', '-nocomments', '-a',
'/home/robot/ev3/mp4_000/mp4_006.jpg'])
subprocess.run(['sudo', 'fbi', '-T', '1',
'-noverbose', '-nocomments', '-a',
'/home/robot/ev3/mp4_000/mp4_007.jpg'])
subprocess.run(['sudo', 'fbi', '-T', '1',
'-noverbose', '-nocomments', '-a',
'/home/robot/ev3/mp4_000/mp4_008.jpg'])
```

```
subprocess.run(['sudo', 'fbi', '-T', '1',
'-noverbose', '-nocomments', '-a',
'/home/robot/ev3/mp4_000/mp4_009.jpg'])
subprocess.run(['sudo', 'fbi', '-T', '1',
'-noverbose', '-nocomments', '-a',
'/home/robot/ev3/mp4_000/mp4_010.jpg'])
subprocess.run(['sudo', 'fbi', '-T', '1',
'-noverbose', '-nocomments', '-a',
'/home/robot/ev3/mp4_000/mp4_011.jpg'])
subprocess.run(['sudo', 'fbi', '-T', '1',
'-noverbose', '-nocomments', '-a',
'/home/robot/ev3/mp4_000/mp4_012.jpg'])
subprocess.run(['sudo', 'fbi', '-T', '1',
'-noverbose', '-nocomments', '-a',
'/home/robot/ev3/mp4_000/mp4_013.jpg'])
subprocess.run(['sudo', 'fbi', '-T', '1',
'-noverbose', '-nocomments', '-a',
'/home/robot/ev3/mp4_000/mp4_014.jpg'])
subprocess.run(['sudo', 'fbi', '-T', '1',
'-noverbose', '-nocomments', '-a',
'/home/robot/ev3/mp4_000/mp4_015.jpg'])
subprocess.run(['sudo', 'fbi', '-T', '1',
'-noverbose', '-nocomments', '-a',
'/home/robot/ev3/mp4_000/mp4_016.jpg'])
```

```
subprocess.run(['sudo', 'fbi', '-T', '1',
'-noverbose', '-nocomments', '-a',
'/home/robot/ev3/mp4_000/mp4_017.jpg'])
subprocess.run(['sudo', 'fbi', '-T', '1',
'-noverbose', '-nocomments', '-a',
'/home/robot/ev3/mp4_000/mp4_018.jpg'])
subprocess.run(['sudo', 'fbi', '-T', '1',
'-noverbose', '-nocomments', '-a',
'/home/robot/ev3/mp4_000/mp4_019.jpg'])
subprocess.run(['sudo', 'fbi', '-T', '1',
'-noverbose', '-nocomments', '-a',
'/home/robot/ev3/mp4_000/mp4_020.jpg'])
subprocess.run(['sudo', 'fbi', '-T', '1',
'-noverbose', '-nocomments', '-a',
'/home/robot/ev3/mp4_000/mp4_021.jpg'])
subprocess.run(['sudo', 'fbi', '-T', '1',
'-noverbose', '-nocomments', '-a',
'/home/robot/ev3/mp4_000/mp4_022.jpg'])
subprocess.run(['sudo', 'fbi', '-T', '1',
'-noverbose', '-nocomments', '-a',
'/home/robot/ev3/mp4_000/mp4_023.jpg'])
subprocess.run(['sudo', 'fbi', '-T', '1',
'-noverbose', '-nocomments', '-a',
'/home/robot/ev3/mp4_000/mp4_024.jpg'])
```

```
subprocess.run(['sudo', 'fbi', '-T', '1',
'-noverbose', '-nocomments', '-a',
'/home/robot/ev3/mp4_000/mp4_025.jpg'])
subprocess.run(['sudo', 'fbi', '-T', '1',
'-noverbose', '-nocomments', '-a',
'/home/robot/ev3/mp4_000/mp4_026.jpg'])
subprocess.run(['sudo', 'fbi', '-T', '1',
'-noverbose', '-nocomments', '-a',
'/home/robot/ev3/mp4_000/mp4_027.jpg'])
subprocess.run(['sudo', 'fbi', '-T', '1',
'-noverbose', '-nocomments', '-a',
'/home/robot/ev3/mp4_000/mp4_028.jpg'])
subprocess.run(['sudo', 'fbi', '-T', '1',
'-noverbose', '-nocomments', '-a',
'/home/robot/ev3/mp4_000/mp4_029.jpg'])
subprocess.run(['sudo', 'fbi', '-T', '1',
'-noverbose', '-nocomments', '-a',
'/home/robot/ev3/mp4_000/mp4_030.jpg'])
subprocess.run(['sudo', 'fbi', '-T', '1',
'-noverbose', '-nocomments', '-a',
'/home/robot/ev3/mp4_000/mp4_031.jpg'])
subprocess.run(['sudo', 'fbi', '-T', '1',
'-noverbose', '-nocomments', '-a',
'/home/robot/ev3/mp4_000/mp4_032.jpg'])
```

```
subprocess.run(['sudo', 'fbi', '-T', '1',
'-noverbose', '-nocomments', '-a',
'/home/robot/ev3/mp4_000/mp4_033.jpg'])
subprocess.run(['sudo', 'fbi', '-T', '1',
'-noverbose', '-nocomments', '-a',
'/home/robot/ev3/mp4_000/mp4_034.jpg'])
subprocess.run(['sudo', 'fbi', '-T', '1',
'-noverbose', '-nocomments', '-a',
'/home/robot/ev3/mp4_000/mp4_035.jpg'])
        #Ctrl+C --> Ctrl+Shift+V into SSH
Terminal
        # sudo python3
/home/robot/ev3/ev3/tests/code/BlendS.py
```

Blend2.py:

```
#!/usr/bin/env python3
import subprocess
```

```
cmd = ['aplay', '-V', '100',
'/home/robot/ev3/seisyun_complex.mp3']
for i in range(50):
    subprocess.run(['sudo', 'fbi', '-T', '1',
'-noverbose', '-a',
'/home/robot/ev3/SeisyunOpening_000/SeisyunOpenin
g_'+str(i).zfill(3)+'.jpg'])
# sudo python3
/home/robot/ev3/ev3/tests/code/Blend2.py
```

CHAPTER 9

THE VIDEO PROGRAMS

Video programs can do many things. First off, using mplayer, we can play gifs and video files.

But I struggled in making it play videos and music at the same time.

That is when I discovered compression. Compressing a video to play it on the EV3 was a struggle to test many different combinations, but in the end:

- .JPG Format
- Use 178x128 resolution
- 44100 Hertz Audio
- 10-15 Frames Per Second for the Video
- Lowest Video compression
- Mono Audio
- If the video is under 4 Megabytes it should be good

Here is a video of my EV3 Playing Bad Apple:

https://www.youtube.com/watch?v=qsX8IqXMkZ0&t=103s

Here is the code for the Video Programs:

```
# brickrun --
/home/robot/ev3/ev3/tests/code/Radio.py
#brickrun -- mpg123 -l 1 --loop -1 -@
http://icecast.omroep.nl/radio1-bb-mp3 -b 1024
```

```
#brickrun -- mpg123 -@
http://us3.streamingpulse.com:7015/live -b 100000
#brickrun -- mpg123 -@
"http://91.232.4.33:7028/stream?type=http&nocach
e=185776" -l 1
```

```
#brickrun -- mplayer -afm mp3lib -acodec mp3 -bps
128 -srate 44100
"http://91.232.4.33:7028/stream?type=http&nocach
e=185776"
```

```
#brickrun -- mplayer
/home/robot/ev3/Chika-Dance.mp4 -vo
fbdev2:/dev/fb0 -framedrop
#brickrun -- mplayer
/home/robot/ev3/output_file.mp4 -vo
fbdev2:/dev/fb0 -framedrop
```

```
#brickrun -- mplayer
/home/robot/ev3/YourNameScenery.gif -vo
fbdev2:/dev/fb0 -framedrop
#brickrun -- mpg123
/home/robot/ev3/BadAppleSong.mp3
#brickrun -- mplayer
/home/robot/ev3/output_file.mp4 --autosync 5vo
fbdev2:/dev/fb0 -framedrop
#brickrun -- ffmpeg -i
/home/robot/ev3/BadAppleSong.mp3
/home/robot/ev3/BadAppleSong.wav
#brickrun -- mpg123
/home/robot/ev3/BadAppleSong.mp3 -b 10000
#brickrun -- aplay
/home/robot/ev3/BadAppleSong.wav
#brickrun -- mplayer /home/robot/ev3/YLIA.gif
```

```
#brickrun -- mplayer
/home/robot/ev3/BadApple.mp4 -vo
fbdev2:/dev/fb0
#brickrun -- mplayer
/home/robot/ev3/BadApple.mp4 -framedrop -vo
fbdev2:/dev/fb0 -autosync 5
#brickrun -- mplayer
/home/robot/ev3/output_file.mp4 -framedrop -vo
fbdev2:/dev/fb0 -autosync 5 -vfm ffmpeg -lavdopts
lowres=1:fast:skiploopfilter=all -cache 16000
-nocache
#brickrun -- mplayer
/home/robot/ev3/BadApple.mp4 -vo
fbdev2:/dev/fb0 -autosync 5
#brickrun -- mplayer
/home/robot/ev3/BadApple2.mpg -vo
fbdev2:/dev/fb0 -autosync 5
#ffmpeg -i /home/robot/ev3/BadAppleSong.wav -ac 1
-ar 16000 /home/robot/ev3/output.wav
```

```
#ffmpeg -i /home/robot/ev3/BadApple.mp4 -i
/home/robot/ev3/BadAppleSong.wav -c:v copy -c:a
copy /home/robot/ev3/output_file.mp4
#brickrun -- mplayer
/home/robot/ev3/Y2Mate_1.mpg -vo
fbdev2:/dev/fb0 -autosync 5
```

```
#brickrun -- mplayer -vo fbdev2:/dev/fb0 -ao sdl
/home/robot/ev3/YLIA.gif -loop 0 & brickrun --
mplayer -ao sdl /home/robot/ev3/.mp3
#brickrun -- mplayer
/home/robot/ev3/output_file.mp4 -vo
fbdev2:/dev/fb0 -lavdopts lowres=1 -noborder
-nomouseinput -quiet
#brickrun -- ffmpeg -i
/home/robot/ev3/BadApple.mp4 -vf
```

```
"fps=10,scale=178:128:flags=lanczos"
/home/robot/ev3/BadApple.gif
```

```
#brickrun -- mplayer -fps 15 -demuxer lavf -lavfdopts
format=mjpeg -vo fbdev2:/dev/fb0 -quiet
/home/robot/webcam.jpg
```

CHAPTER 10

HEARING SENSOR PROGRAMS

I have also used Hearing sensors in the robot. Using NXT Hearing Sensors, I have implemented a program to use the hearing sensor.

What the program does is that using the head turning motor, the 2 hearing sensors the robot waits for a loud sound near the sensor, and then turns its head towards that sound.

The program uses Decibels mode to compare the decibels heard during the hearing sensor. Also there is a minimum threshold the hearing sensor has to hear so that it does not turn for background noise.

Here is the sensor program:

```
#!/usr/bin/env python3
from ev3dev2.sensor.lego import SoundSensor
from ev3dev2.sensor import INPUT_2, INPUT_1
from ev3dev2.motor import OUTPUT_C, LargeMotor
import time
from time import sleep
import ev3dev2.auto as ev3
from ev3dev2.display import Display
sound_sensor_left = SoundSensor('in2')#left ear is 2
sound_sensor_right = SoundSensor('in1')#right ear is
1
sound_sensor_left.mode = 'DB'
threshold = 92
sound_sensor_right.mode = 'DB'
import ev3dev2.fonts as fonts
motor = LargeMotor(OUTPUT_C)#head turning is C
display = ev3.Display()
MOTOR_SPEED = 50

initial_position = motor.position
target_position = int(360 * .6)#360 is the angle
negtarget_position = int(360 * 1 * -1)#360 is the
angle

while True:
    sound_value_left =
sound_sensor_left.sound_pressure
```

```
    sound_value_right =
sound_sensor_right.sound_pressure
    if sound_value_left > threshold:
        motor.run_to_abs_pos(position_sp= int(360 *
.8), speed_sp= 50)
        time.sleep(3)
        motor.run_to_abs_pos(position_sp= -int(360 *
.8), speed_sp=50)
        motor.wait_until_not_moving()
    elif sound_value_right > threshold:
        motor.run_to_abs_pos(position_sp= -int(360 *
.8), speed_sp=50)
        time.sleep(3)
        motor.run_to_abs_pos(position_sp= int(360 *
.8), speed_sp=50)
        motor.wait_until_not_moving()
    else:
        motor.stop()



#sudo python3
/home/robot/ev3/ev3/tests/code/Hearing.py
```

CHAPTER 11

CONNECTING TO THE INTERNET

Using EV3DEV, you can connect to the internet.

This has many use scenarios, such as:

- Downloading classes into the EV3 environment
- If you cannot download a class, using the internet you can get entire repositories onto the EV3 so that you can download the classes from the repositories.
- You can get information from the internet, such as time, weather, stocks, news, and much more. It can even say it!
- It can play music from the internet.
- It can play an internet-based radio.

To connect to the internet, there are a few steps you have to take:

1. First off, power on the EV3 and your computer.

2. Second, this method only works if the EV3 is connected to the computer via a USB cable.
3. Third, connect the USB cable to the EV3.
4. Fourth, open Control Panel > Hardware and Sound > Devices and Printers > and then Right click your EV3 Device (Remote NDIS Device)
5. Fifth, go to Go to Network Settings > Left tab Network Settings.
6. Sixth, right click your home wifi, then go to the 3rd tab. Click the checkmark on both of them and on the first option make sure it connects to the EV3.
7. Seventh, go to the EV3 > Connections > Wired > Click connect. Wait for a few minutes and it should say online.
8. That is it! You are connected to the internet!

9. If it does not, there is one thing you do. Repeat step 6 by unchecking both options, wait for a few minutes, recheck them and then continue from there. Be patient! It is very old hardware!

That is how you connect to the internet!

CHAPTER 12

USING THE INTERNET TO PLAY THE RADIO

Using EV3DEV, you can connect to the internet. Using the internet, you can play the radio!

Here are a few things. The EV3 does **NOT** have a Radio antenna. So you will have to use a radio link!

Here is an example of a radio link. Beware, it is German haha!

http://icecast.omroep.nl/radio1-bb-mp3

Using this radio link, we can use MPG123 or aplay to play it on the ev3. It does take time to load since it uses the internet and depends on internet speed. Also can lag after some time.

Here is a command for the internet radio:

brickrun -- mpg123 -l 1 --loop -1 -@

http://icecast.omroep.nl/radio1-bb-mp3 -b 10000

That's it!

Here is a radio program:

```
from
ev3dev2.dis
play import
Display
              from ev3dev2 import ev3
              import subprocess


              screen = Display()


              process = subprocess.Popen(
                 ['brickrun', '--', 'mpg123',
              '-@',
              'http://us3.streamingpulse.com:701
              5/live', '-b', '100000'],
                 stdout=subprocess.PIPE,
                 universal_newlines=True
              )
```

```
for line in
iter(process.stdout.readline, ''):
   if 'ICY-NAME' in line:
       station_name =
line.split('ICY-NAME:
')[1].strip()
       screen.draw.text((10, 10),
f"Station:"+station_name)
       screen.update()


   if 'ICY-META: StreamTitle' in
line:
       song_name =
line.split("StreamTitle='")[1].spl
it("';")[0]
       screen.draw.text((10, 50),
f"Song: {song_name}")
       screen.update()


   if ev3.Button().middle:
       break
```

```
# Close the subprocess
process.stdout.close()
process.wait()


#sudo python3
/home/robot/ev3/ev3/tests/code/Rad
io.py
```

Chapter 13

Using Artifical Intelligence to generate responses

This one is a long one. So bear with me.

Introduction

The purpose of this document is to provide an in-depth explanation of the usage of ChatGPT in generating responses based on the provided code snippet. ChatGPT, developed by OpenAI, is a state-of-the-art language

model that utilizes deep learning techniques to generate human-like text responses. The code snippet showcases the integration of ChatGPT with a speech recognition system and an EV3 robot, enabling the creation of an interactive voice-based chatbot. This document will delve into the various aspects of the code and the underlying mechanisms of ChatGPT.

Code Explanation - Recording and Transcribing Audio

The code begins by importing the necessary libraries and initializing the sound and touch sensors. The "record_audio" function is responsible for capturing audio input. It uses the "arecord" command-line utility to record audio from the default microphone and saves it as "voice.wav". The "transcribe_audio" function leverages the Google Cloud Speech-to-Text API for transcribing the recorded audio. It reads the audio file, configures the recognition settings such as encoding, sample rate, language code, and

model, and sends the audio data to the API for transcription. Finally, the function retrieves the transcribed text for further processing.

Code Explanation - Generating Response

The "generate_response" function utilizes the OpenAI API to generate a response based on the provided text prompt. It takes the transcribed text as input and constructs a prompt by formatting the text accordingly. The function then utilizes the OpenAI Completion API to send the prompt and receive a response. The "davinci" engine is specified for generating the response, which represents the most powerful variant of the GPT family. Additionally, the function sets parameters such as the maximum number of tokens, temperature, and the number of responses to generate. The API returns a list of choices, and the function extracts the first choice as the generated response.

Code Explanation - Speaking the Response

The "speak_response" function incorporates the EV3 sound module to audibly communicate the generated response. It takes the response text as input and utilizes the "speak" method of the sound object to play the response as speech output. This functionality enhances the interaction between the EV3 robot and users by enabling the robot to provide spoken responses, thereby creating a more engaging and interactive user experience.

Main Loop and Interaction

The main loop of the code snippet ensures continuous monitoring of the touch sensor's state. When the touch sensor is pressed and recording is not

in progress, the code triggers a beep sound and initiates the audio recording process. Conversely, if the touch sensor is pressed again while recording is in progress, another beep sound is triggered, and the recording is stopped. Subsequently, the transcribed text is printed to provide a visual representation of the user's input. The generate_response function is then called to generate a response based on the transcribed text. The generated response is printed, spoken out using the speak_response function, and the recording state is reset to prepare for future interactions.

ChatGPT - An Overview

ChatGPT is a powerful language model developed by OpenAI. It is trained using a massive amount of text data and employs deep learning techniques, specifically transformer neural networks, to understand and

generate text. The model is designed to engage in natural language conversations and generate coherent and contextually appropriate responses. It has been trained on a wide variety of internet text sources, enabling it to capture the nuances of language and provide human-like interactions.

OpenAI API Integration

To leverage the capabilities of ChatGPT, the code snippet integrates with the OpenAI API. The "openai.api

Page 7 (continued):

_key" is used to set the API key required for authentication. It is recommended to store the API key securely and access it using environment variables for security purposes. The "generate_response" function utilizes the OpenAI Completion.create() method to interact with

the API. It specifies the "davinci" engine for response generation, sets parameters such as maximum tokens, temperature, and the number of responses to generate. The API responds with a list of choices, and the function extracts the first choice as the generated response.

Speech Output and Interaction

The "speak_response" function utilizes the EV3 sound module to convert the generated response into audible speech. It takes the response text as input and invokes the "speak" method of the sound object to play the response as speech output. This capability enables the EV3 robot to communicate with users in a more interactive and human-like manner. By speaking the response, the robot enhances the overall user experience and fosters a more natural and engaging conversation.

Enhancing the User Interface

The code snippet can be extended to incorporate additional user interface elements. For example, a graphical interface could be implemented to display the transcribed text and the generated response in a more user-friendly manner. This could include using text boxes or speech bubbles to visually represent the conversation between the user and the EV3 robot. Such enhancements can make the interaction more intuitive and visually appealing, providing a seamless user experience.

Conclusion

This document has provided a comprehensive explanation of the usage of ChatGPT in generating responses based on the provided code snippet. By integrating ChatGPT with a speech recognition system and an EV3 robot,

users can engage in interactive voice-based communication with the robot. The combination of advanced language modeling and robotic technology opens up exciting possibilities for creating intelligent and conversational applications. Leveraging the power of ChatGPT and the OpenAI API, developers can create chatbots, virtual assistants, and other applications that provide natural and contextually relevant responses. The provided code snippet serves as a starting point for exploring the potential of ChatGPT and its integration with robotics. With further customization and refinement, the application can be enhanced to deliver even more immersive and engaging conversational experiences.

Here is the code for it:

```
#!/usr/bin/env python3
import subprocess
import io
import os
from ev3dev2.sensor import INPUT_4
from google.cloud import speech
from google.cloud.speech import enums
from google.cloud.speech import types
import openai
import time

from ev3dev2.sound import Sound
from ev3dev2.sensor.lego import TouchSensor

sound = Sound()
touch_sensor = TouchSensor(INPUT_4)

def record_audio():
    subprocess.call("arecord -D hw:1,0 -f S16_LE -r
16000 voice.wav", shell=True)

def transcribe_audio():
    client = speech.SpeechClient()
    with io.open("voice.wav", "rb") as audio_file:
        content = audio_file.read()
    audio = types.RecognitionAudio(content=content)
    config = types.RecognitionConfig(
```

```
encoding=enums.RecognitionConfig.AudioEncoding.LINEAR16,
        sample_rate_hertz=16000,
        language_code="en-US",
        model="video"
    )
    operation = client.long_running_recognize(config=config, audio=audio)
    response = operation.result(timeout=90)
    return response.results[0].alternatives[0].transcript

def generate_response(text):
    openai.api_key = os.environ["OPENAI_API_KEY"]
    prompt = "Your prompt here: {}".format(text)
    response = openai.Completion.create(
        engine="davinci",
        prompt=prompt,
        max_tokens=2048,
        temperature=0.5,
        n=1,
        stop=None,
    )
    return response.choices[0].text.strip()
```

```
def speak_response(response):
    sound.speak(response)

recording = False

while True:
    if touch_sensor.is_pressed and not recording:
        sound.beep()
        recording = True
        record_audio()
    elif touch_sensor.is_pressed and recording:
        sound.beep()
        text = transcribe_audio()
        print("You said: {}".format(text))

        response = generate_response(text)
        print("GPT Says: {}".format(response))

        speak_response(response)
        recording = False
    time.sleep(0.1)
#sudo python3
/home/robot/ev3/ev3/tests/code/Talking.py
```

CHAPTER 14

ALL THE SUDO LINES TO EXECUTE THE PROGRAMS

For most of the programs, we need to use the SSH terminal to execute the programs. Here is a convenient place to put all the lines that we need to execute the programs.

Remember to do Ctrl+Shift+C and then Ctrl+Shift+V into the SSH Terminal otherwise it will not paste

ColorName:

```
sudo python3
/home/robot/ev3/ev3/tests/code/ColorName.py
```

ColorNameTouch:

```
sudo python3
/home/robot/ev3/ev3/tests/code/ColorNameTouch.py
```

Webcam:

```
sudo python3
/home/robot/ev3/ev3/tests/code/Webcam.py
```

WebcamDisplay:

```
sudo python3

/home/robot/ev3/ev3/tests/code/WebcamDisplay.py
```

WebcamStream:

```
sudo python3

/home/robot/ev3/ev3/tests/code/WebcamStream.py
```

Recording:

```
sudo python3

/home/robot/ev3/ev3/tests/code/Recording.py
```

RecordingPlayback:

```
sudo python3

/home/robot/ev3/ev3/tests/code/RecordingPlayback.

py
```

Blend2:

```
sudo python3

/home/robot/ev3/ev3/tests/code/Blend2.py
```

BlendS:

```
sudo python3

/home/robot/ev3/ev3/tests/code/BlendS.py
```

Seis:

```
sudo python3

/home/robot/ev3/ev3/tests/code/Seis.py
```

SeisyunComplex:

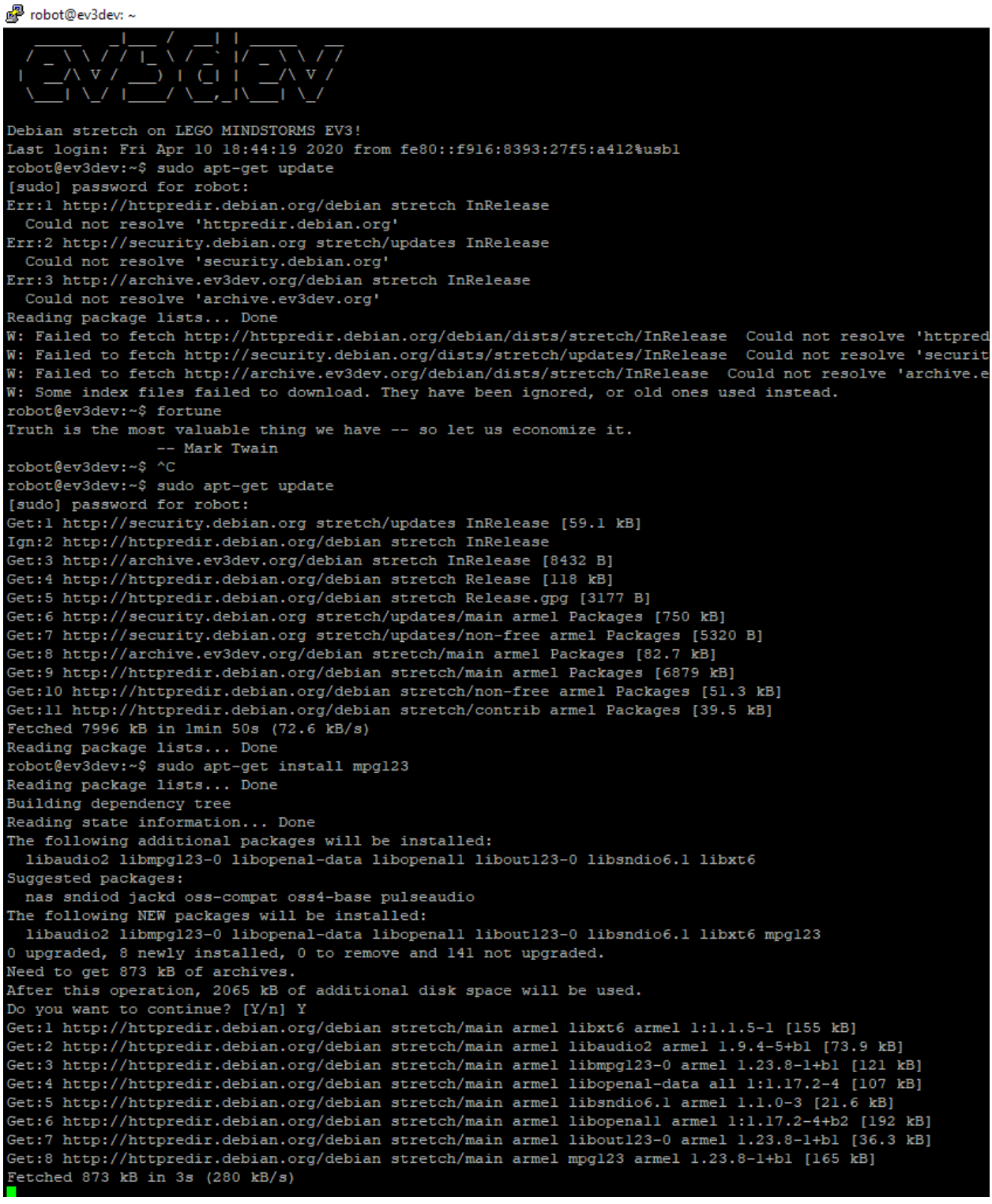
```
sudo python3

/home/robot/ev3/ev3/tests/code/SeisyunComplex.py
```

CHAPTER 15

EXTRA PICTURES AND IMPORTANT INFORMATION

Username: robot

Password: maker



Sudo apt-get install

Github User: CyanCheetah

https://github.com/CyanCheetah/ev3

CHAPTER 16

MORE SUDO LINES FOR REFERENCE

```
#sudo python3
/home/robot/ev3/ev3/tests/code/Radio.py
# brickrun --
/home/robot/ev3/ev3/tests/code/Radio.py
#brickrun -- mpg123 -l 1 --loop -1 -@
http://icecast.omroep.nl/radio1-bb-mp3 -b 1024


#brickrun -- mpg123 -@
http://us3.streamingpulse.com:7015/live -b 100000
#brickrun -- mpg123 -@
"http://91.232.4.33:7028/stream?type=http&nocache=18
5776" -l 1

#brickrun -- mplayer -afm mp3lib -acodec mp3 -bps
128 -srate 44100
"http://91.232.4.33:7028/stream?type=http&nocache=18
5776"


#brickrun -- mplayer /home/robot/ev3/Chika-Dance.mp4
-vo fbdev2:/dev/fb0 -framedrop
#brickrun -- mplayer /home/robot/ev3/output_file.mp4
-vo fbdev2:/dev/fb0 -framedrop
#brickrun -- mplayer
/home/robot/ev3/YourNameScenery.gif -vo
```

```
fbdev2:/dev/fb0 -framedrop
#brickrun -- mpg123 /home/robot/ev3/BadAppleSong.mp3
#brickrun -- mplayer /home/robot/ev3/output_file.mp4
--autosync 5vo fbdev2:/dev/fb0 -framedrop
#brickrun -- ffmpeg -i
/home/robot/ev3/BadAppleSong.mp3
/home/robot/ev3/BadAppleSong.wav
#brickrun -- mpg123 /home/robot/ev3/BadAppleSong.mp3
-b 10000
#brickrun -- aplay /home/robot/ev3/BadAppleSong.wav
#brickrun -- mplayer /home/robot/ev3/YLIA.gif

#brickrun -- mplayer /home/robot/ev3/BadApple.mp4
-vo fbdev2:/dev/fb0
#brickrun -- mplayer /home/robot/ev3/BadApple.mp4
-framedrop -vo fbdev2:/dev/fb0 -autosync 5
#brickrun -- mplayer /home/robot/ev3/output_file.mp4
-framedrop -vo fbdev2:/dev/fb0 -autosync 5 -vfm
ffmpeg -lavdopts lowres=1:fast:skiploopfilter=all
-cache 16000 -nocache
#brickrun -- mplayer /home/robot/ev3/BadApple.mp4
-vo fbdev2:/dev/fb0 -autosync 5
#brickrun -- mplayer /home/robot/ev3/BadApple2.mpg
-vo fbdev2:/dev/fb0 -autosync 5
#ffmpeg -i /home/robot/ev3/BadAppleSong.wav -ac 1
-ar 16000 /home/robot/ev3/output.wav
#ffmpeg -i /home/robot/ev3/BadApple.mp4 -i
```

```
/home/robot/ev3/BadAppleSong.wav -c:v copy -c:a copy
/home/robot/ev3/output_file.mp4
#brickrun -- mplayer /home/robot/ev3/Y2Mate_1.mpg
-vo fbdev2:/dev/fb0 -autosync 5


#brickrun -- mplayer -vo fbdev2:/dev/fb0 -ao sdl
/home/robot/ev3/YLIA.gif -loop 0 & brickrun --
mplayer -ao sdl /home/robot/ev3/.mp3
#brickrun -- mplayer /home/robot/ev3/output_file.mp4
-vo fbdev2:/dev/fb0 -lavdopts lowres=1 -noborder
-nomouseinput -quiet
#brickrun -- ffmpeg -i /home/robot/ev3/BadApple.mp4
-vf "fps=10,scale=178:128:flags=lanczos"
/home/robot/ev3/BadApple.gif

#brickrun -- mplayer -fps 15 -demuxer lavf
-lavfdopts format=mjpeg -vo fbdev2:/dev/fb0 -quiet
/home/robot/webcam.jpg
```

CHAPTER 17

LIST OF THE THINGS I HAVE INSTALLED

- PIL (python image library)
- Mplayer
- Fbi
- Fswebcam
- EV3DEV
- MPG123

- MPG321
- Aplay
- Openai
- Sphynx
- Debian
- And more! Will update as I go along

CHAPTER 18

REFERENCE LINKS

- https://github.com/CyanCheetah
    - My Github Profile
- https://github.com/CyanCheetah/CyanBot
    - This programs Github

- https://www.youtube.com/watch?v=alDYm5ZyAks
    - The Robot Overview Video
- https://www.youtube.com/@cyancheetah7034
    - My Youtube Channel (Subscribe!)
- https://www.youtube.com/watch?v=qsX8IqXMkZ0&t=103s
    - EV3 Plays Bad Apple

CHAPTER 18

WAYS TO CONTACT ME

Discord: CyanCheetah#6013

Email: saitanuj12@gmail.com

Linkedln: https://www.linkedin.com/in/sai-tanuj-karavadi-0b6b54265/

That is about it!

Thanks for reading!