Junior Design Project: Noise Detection Camera System

Junior Design Project

CSE 398

M002

Samsondeen Batula, Ibrahima Diallo, Elizabeth Fatade, Matthew Leight

May 9th, 2022

Table of Contents

Introduction

According to FBI and Safewise stats, You are more likely to be a victim of property crime than any other type of crime. Prevention is your #1 option to reduce your risk of victimization. Property Crime makes up about 80% of all crime in the US whether it be burglary, larceny, vandalism, or home break ins. To help address the problem of lack of security and high robbery percentages, we are creating a sound activated camera system to model modern home & camera security systems. The camera will be inside a 3D printed box, which is controlled by one Servo Motor. When a sound event occurs, our device will activate, revealing a camera, and take a picture. The picture taken will then be displayed on a host computer in real time. Our project helps target and prevent such attacks. We drew our inspirations for how we will build our project from current real world products such as a Smart sensor Doorbell Camera and Property Security Systems like Home ADT, Link Interactive and Car Dash Cams.

Building our signature list was fairly simple. We implemented the concept of problem-based learning that Dr. Marcy taught us early on in the semester. The concept of problem-based learning will be implemented: a teaching method in which complex real-world problems are used as the vehicle to promote student learning of concepts and principles as opposed to direct presentation of facts and concepts. We can apply problem-based learning to apply our engineering knowledge to create embedded systems that would help us solve these complex real-world problems, rather than applying the traditional approach to learning. As a result, we were prompted to make our signatures for each class day and create points, rather than the professor telling us what exactly to do. Our signatures consisted of: Setup microphone to detect

noise event on the Serial Monitor (Hardware and Software) **(3 pts)**, Receiving a packet from the Arduino from the Rock Pi **(2 pts)**, Take and save picture once packet is Received **(3 pts)**, Setup Client Server Connection and Send Data (Picture) From Rock Pi to Host Computer **(3 pts)**, Show Control of Motor with MRAA PWM pins **(3 pts)**, Move the servo motor to desired direction, Then take picture, save it on Rock Pi, and send it to the Host Computer **(4 pts)**, Create 3d Printed Box, connect Servo Motor and packets, and demo final project **(2 pts)**

Materials, Software & Hardware Layout

Software:

- OpenCV (python image processing library)
- ZMQ (ZeroMQ, python asynchronous messaging library)

Hardware:

- Arduino Microcontroller (Itsy Bitsy M0)
- Camera (Webcam)
- Servo Motor
- Rock Pi 4B
- Microphone (Adafruit Max 4466)
- MacBook (Host Computer)

Hardware Setup:

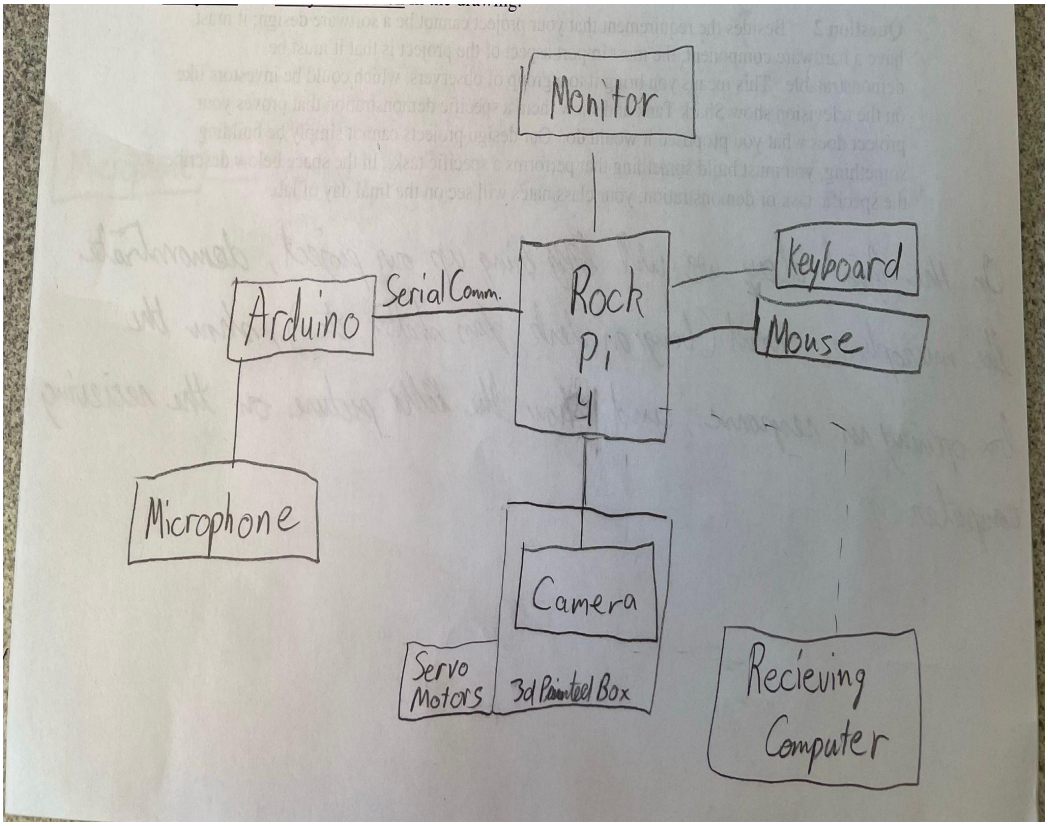


Figure 1: Our device Hardware Setup

<u>Signature 1:</u> Setup microphone to detect noise event on the Serial Monitor (Hardware and Software)

Setup:

The equipment used to start off the project and obtain this signature are:

- Arduino Itsy Bitsy M0 - Microcontroller
- Microphone - Adafruit Max 4466
- Personal Laptop to Code the Arduino Chip
- USB Cables
- HDMI Cable

Description:

For Signature 1, We are detecting a noise event using a microphone and an Arduino Micrcontroller. The Microcontroller we are using is the Itsy Bitsy M0. We used this due to our familiarity with it in previous projects. The microphone we chose was the Adafruit Max 4466 due to it being easily accessible and already in the Lab. Now we need to interface the Itsy Bitsy M0 microcontroller with the Adafruit Max 4466 Microphone. First, we set up the hardware by connecting an analog pin on the Itsy Bitsy M0 to the command pin on the Max 4466. We chose Analog Pin 0 **(A0)** on the Itsy Bitsy to connect to the command pin **(OUT)** on the Max 4466. The microphone can be powered by a max of 3v, so we connected 3v from the Itsy Bitsy M0 to the VCC line in the microphone. We also made sure that both devices are grounded together.

After setting up the hardware we went to the software setup. We used an example code we found online where the microphone detects a high enough voltage and displays that in the Serial monitor. With a little bit of tweaking to the code, we configured the code to print “Mic Event Detected!!” whenever a high enough voltage is detected. We can create high enough voltages by clapping, playing a song into the mic, or simply tapping on the microphone.

Our code for the first signature can be found in full detail **HERE**

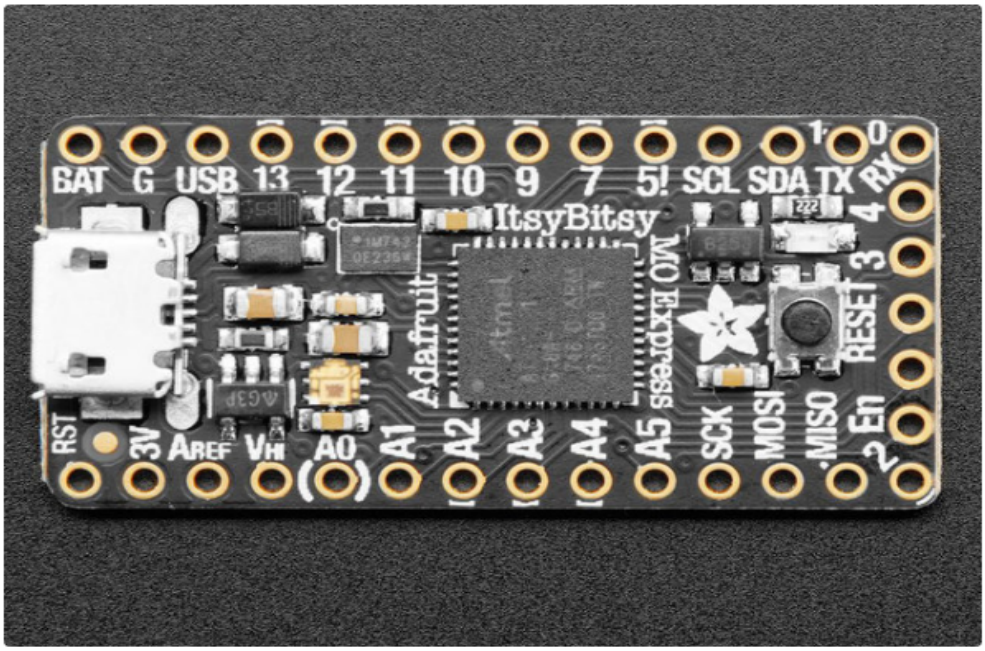


Figure 2: Arduino Itsy Bitsy M0 PINOUT/SCHEMATIC

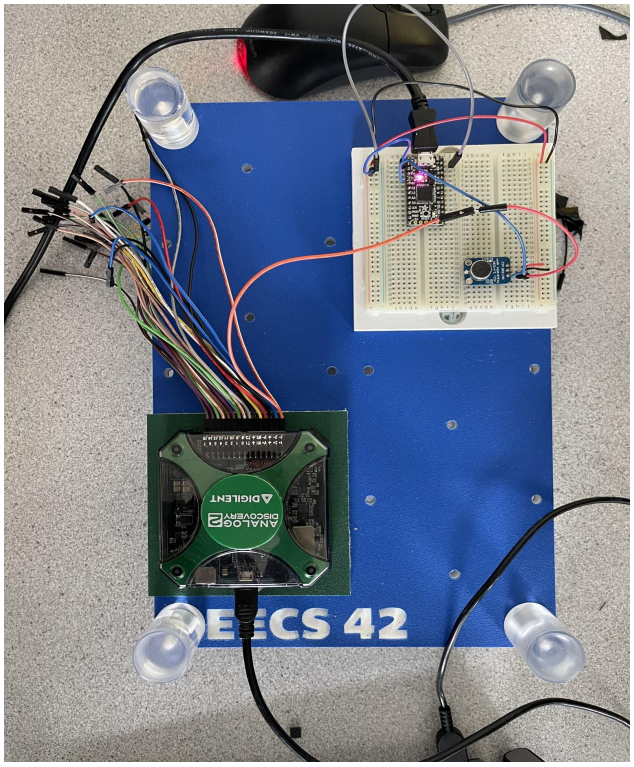


Figure 3: Picture of our Hardware Setup in Signature 1

<u>Signature 2:</u> Receiving a packet from the Arduino from the Rock Pi

Setup:

The equipment used to obtain this signature are:

- Rock Pi 4B
- Keyboard (For Rock Pi)
- Mouse (For Rock Pi)
- VGA Cable
- Monitor
- Power source
- HDMI Cable
- SD card (Debian Family)
- SD card reader
- Analog Discovery
- Everything else is the same as the previous signature

Description:

For Signature 2, We are now using the Rock Pi 4B to build the remainder of our project. We don't need to etch the Debian Family Imager on the SD card because we already did that for our previous project. We are sending packets of data that hold a header and 21 float points of data

from the Arduino Itsy Bitsy M0 to the Rock Pi 4B. Our reason behind sending 21 points of data is for the first 10 to represent the data of the noise levels before the spike sound, the middle number to represent the noise data that cause the spike and the last 10 to represent the noise data after the spike. An advantage of getting data like this is the fact that it can later on be graphed to showcase the change in different noise levels. This informations will be displayed on the terminal on the Rock Pi. To do we made use of the Serial print feature on the Arduino to send over the header and data points over the serial bus when the Mic event occurs and then use the UART to read the data. Through testing we were able to discover that the whole packet was about 50 bytes and created a counter variable in our Rock Pi code toncount up to 50 before stopping the read function with the UART. There are a number of steps needed to enable UART in order to make use of the read function. Once the mic event happens on the Arduino side, a Packet is sent via serial port to the Rock Pi. Once received the packet will be displayed on the terminal on the rock pi.

First, we need to install minicom. This is how we are able to set which teletype serial port **(TTYS)** we are sending our packets through. We are using ttyS4. ***Note it isn't advised to use ttyS2 due to the fact that the serial port is defaulted to already having data running through it.***

- To install minicom type: **sudo apt-get install minicom**
- To enter minicom terminal type: **minicom -s**
- From there enter Serial Port Setup and edit the Serial Device to whichever TTYS you want to send data through.

Now that our Minicom was set, we are able to start coding our Arduino code to send the packet and create a python file to code the Rock Pi to receive the packets. Hardware implementations

we added were interfacing the TX and RX pins on the Rock Pi with the RX and TX pins on the Arduino Itsy Bitsy M0. On the Rock Pi, RX is Pin 10 and TX is Pin 8. On the Itsy Bitsy M0, it simply says where TX and RX are. We also made sure to ground everything on the Rock Pi 4B to match everything grounded on the Itsy Bitsy M0. We also added 3V power from the Rock Pi 4B to the Arduino Itsy Bitsy M0.

Our code for Sending Packets from the Arduino microcontroller to the Rock Pi 4B can be found **HERE**

Our code for Receiving Packets on the Rock Pi 4B from the Arduino microcontroller can be found **HERE**

GPIO number	Function2	Function1	GPIO	Pin#	Pin#	GPIO	Function1	Function2	GPIO number
		+3.3V		1	2		+5.0V		
71		I2C7_SDA	GPIO2_A7	3	4		+5.0V		
72		I2C7_SCL	GPIO2_B0	5	6		GND		
75		SPI2_CLK	GPIO2_B3	7	8	GPIO4_C4	UART2_TXD		148
		GND		9	10	GPIO4_C3	UART2_RXD		147
146		PWM0	GPIO4_C2	11	12	GPIO4_A3	I2S1_SCLK		131
150		PWM1	GPIO4_C6	13	14		GND		
149		SPDIF_TX	GPIO4_C5	15	16	GPIO4_D2			154
		+3.3V		17	18	GPIO4_D4			156
40	UART4_TXD	SPI1_TXD	GPIO1_B0	19	20		GND		
39	UART4_RXD	SPI1_RXD	GPIO1_A7	21	22	GPIO4_D5			157
41		SPI1_CLK	GPIO1_B1	23	24	GPIO1_B2	SPI1_CSn		42
		GND		25	26		ADC_IN0		
64		I2C2_SDA	GPIO2_A0	27	28	GPIO2_A1	I2C2_CLK		65
74	I2C6_SCL	SPI2_TXD	GPIO2_B2	29	30		GND		
73	I2C6_SDA	SPI2_RXD	GPIO2_B1	31	32	GPIO3_C0	SPDIF_TX	UART3_CTSn	112
76		SPI2_CSn	GPIO2_B4	33	34		GND		
133		I2S1_LRCK_TX	GPIO4_A5	35	36	GPIO4_A4	I2S1_LRCK_RX		132
158			GPIO4_D6	37	38	GPIO4_A6	I2S1_SDI		134
		GND		39	40	GPIO4_A7	I2S1_SDO		135

Figure 4: Rock Pi 4B PINOUT

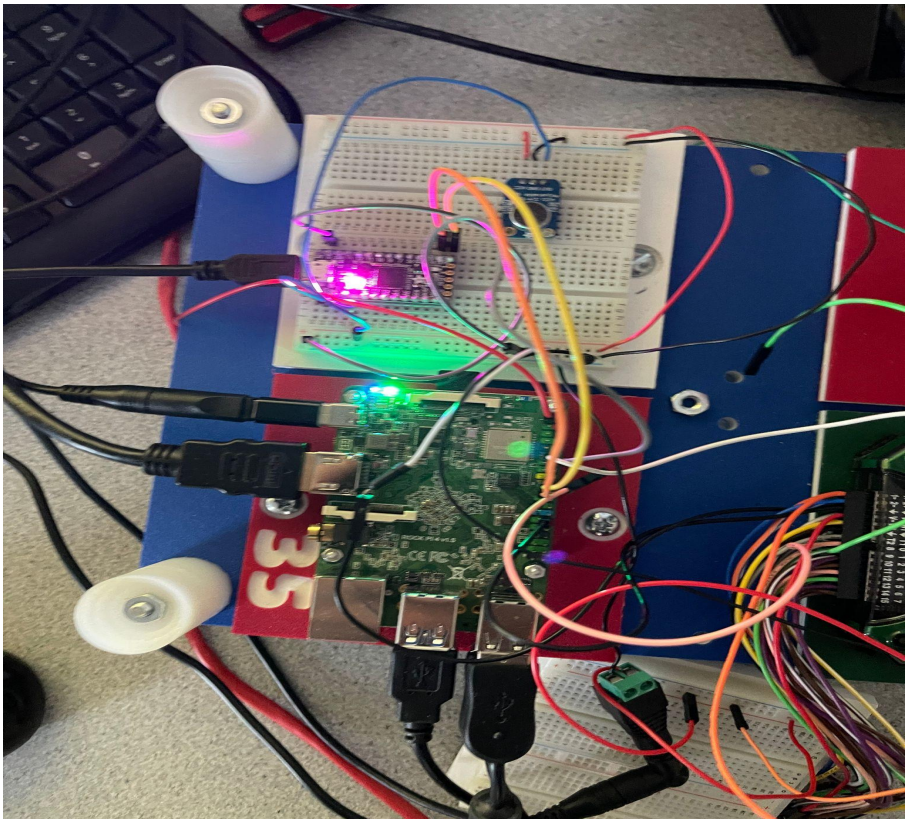


Figure 5: Updated Picture of our hardware now that the Rock Pi 4B is implemented

Note, We grounded everything to the same pin on the Rock Pi to keep everything we used consistent

Note, TX on the Arduino goes to RX on the Rock Pi 4B and RX on the Arduino goes to TX on the Rock Pi 4B.

<u>Signature 3:</u> Take and save picture once packet is Received

Setup:

The equipment used to obtain this signature are:

- USB enabled Webcam
- Everything else is the same as the previous signatures

Description:

Now that the Packet is received and displayed on the Terminal, we need to configure the code to take a picture and save it on the Rock Pi. Instead of writing the code for the camera to take a picture inside the main file, we created a Camera file and wrote the code there. The rest of the Junior Design project was out of order. We wanted to make sure our Servo Motor worked, so we went ahead and set up the hardware and code for that. Going back to creating the camera code, We then implemented an exec(open(“ “)) function. This simply embeds any code we want to run from a different file in the main file. To run the Camera code we added this exec(open(“ “)) in the Servo Motor Code. We did this because we wanted the Camera to take a picture once the Servo Motor reached a certain position. If we added this function in the Main Rock Pi code, the camera would’ve just taken a picture at any given moment. We also used the exec(open(“ “)) function for the Servo Motor Code as well. Adding this code into the Main code makes use of

both the Servo Code and the Camera code. Meaning, once the Servo Motor reaches the position we set it to reach, the camera code is then executed and takes the picture and saves it.

- The exec function to execute the Camera Code is: **exec(open("cameraTest.py").read())**
- The exec function to execute the Servo Motor Code is: **exec(open("servo.py").read())**

Note because of us adding the camera code into the Servo code, we are adding both codes here.

Our code for taking a picture and saving it can be found **HERE** ←- (ADD HYPERLINK HERE)

Our code for the Servo Motor can be found **HERE** ←- (ADD HYPERLINK HERE)

<u>Signature 4:</u> Setup Client Server Connection and Send Data (Picture) From Rock Pi to Host Computer

Setup:

- USB enabled Webcam
- Macbook Pro (M1 Chip)
- Everything else is the same as the previous signatures

Description:

After setting up the Rock Pi 4B and camera to take a picture once a packet has been received we wanted to send the captured picture to a host computer. The host computer used for this project was a Macbook Pro with an M1 Chip. To begin, we set up two python files, one on the Rock Pi (send_image.py) and the other on the host computer (imageReceiver.py). The Rock Pi was to behave as the client and the host computer as the server. In previous lab projects, we had used the python imagezmq module to set up communication between two computers so we decided it would be best to use it again for this project. An additional benefit of imagezmq we discovered is that, as part of the ZMQ library, it has high throughput (updates frames quickly) and low latency (distributes image quickly to connected computers). A lot of code used for both python files were received from this github link - https://github.com/jeffbass/imagezmq/blob/master/README.rst

send_image.py:

The code for this file was built on the Camera file that utilized OpenCv to take a picture once a packet had been received. In order to make sure that our picture would be able to get sent over,

we first had to connect to the host computer. We initially attempted to establish a tcp connection between the Rock Pi and the host computer using the host computer's name of "Macbook-Pro-2". However, that did not work and was not able to connect to the computer at all. After testing different methods, we concluded that over the current internet connection (iot_lab) we would not be able to use hostname of the server but rather the IP address. Once that had been discovered we started working on getting the IP address of the server computer. Since the host computer was a Macbook, we had to use the "netstat -r" command to display the IP routing table for the computer.

```
(base) elizabethfatade@MacBook-Pro-2 ~ % netstat -r
Routing tables

Internet:
Destination        Gateway            Flags           Netif Expire
default            100.64.0.1         UGScg             en0
100.64/20          link#11            UCS               en0      !
100.64.0.1/32      link#11            UCS               en0      !
100.64.0.1         f8:f2:1e:ad:90:61  UHLWIir           en0   1137
100.64.0.169       da:7c:cf:c6:bb:b7  UHLWI             en0    696
100.64.3.47        8c:85:90:6b:ad:7a  UHLWI             en0      !
100.64.3.99        4:54:53:10:76:69   UHLWI             en0      !
```

Figure 6: An example of an IP routing table after running "netstat -r"

We ran this command after connecting to the lab room's internet. After getting the IP address of the server computer, we were able to establish a tcp connection between the Rock Pi and the host computer. The Rock Pi waits for a second before sending the picture to the host computer.

Note that due to the nature of the way the iot_lab internet assigns IP addresses to devices (everytime the device reconnects it assigns it a new and different IP address), we had to go back into the code and change the IP address each time the Macbook Pro reconnected

imageReceiver.py:

In order to receive the captured image, this code had to be running before the send_image.py file. This is to ensure that the file was working and waiting to receive a picture.

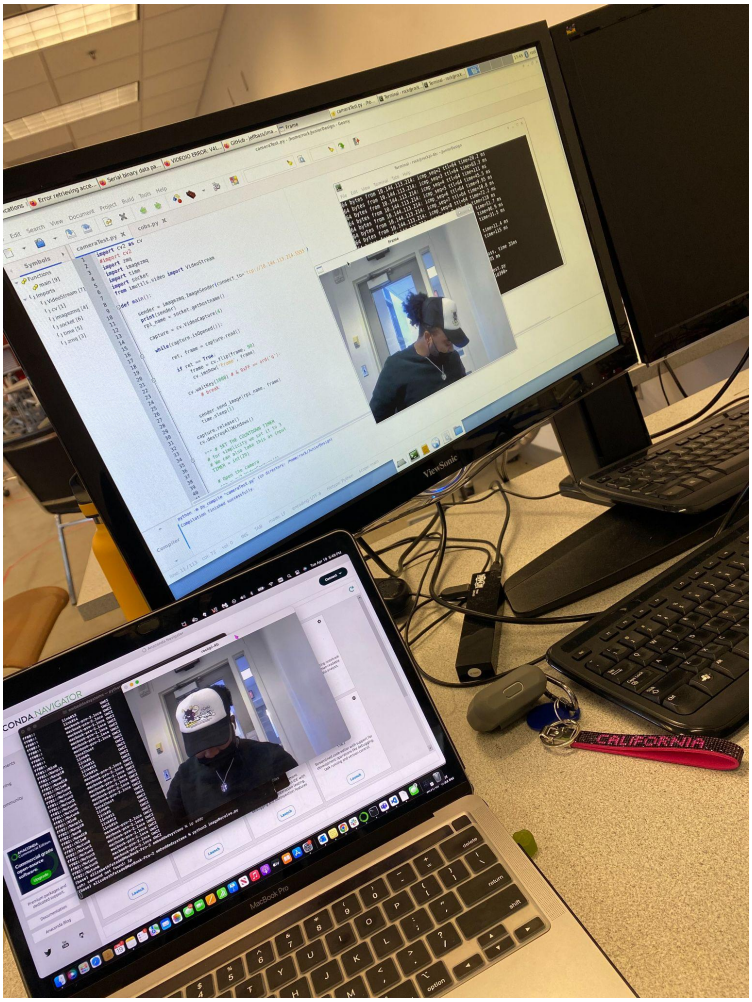


Figure 7: Sending captured picture from Rock Pi to Macbook Pro

Note that there is a bit of a delay when displaying the picture on the Rock Pi and then on the host computer. The Rock Pi updates first, then the host computer

Send_image.py code can be found **HERE**

imageReceiver.py code can be found **HERE**

<u>Signature 5:</u> Show Control of Motor with MRAA PWM pins

Setup:

The equipment used to obtain this signature are:

- Servo Motor (1x)
- Power Brick (Set to 5V)
- Everything else is the same as the previous signatures

Description:

Setting up the Servo Motor was pretty simple. We need to enable the PWM we are using, which is PWM0.

- The GPIO pin for this on the Rock Pi is pin 11. Command wire on the Servo motor (white) is plugged into pin 11 to control the pan on the servo motor.

We already had code from earlier on in the semester that sped up the setup process. The code was taken directly from project 3. In project 3 we needed to use 2 servo motors, one for pan (moving horizontally) and one for tilt (moving vertically). For our Junior Design Project, we are only taking advantage of the tilt movement so we discarded the pan movement code. Setting up the hardware aspect of the Servo Motor wasn't hard at all either. We just needed to make sure that we are using 5V (volts) to power the Servo. Anything less than 5V and the servo motor will turn

on and off while moving positions. We also grounded the Servo Motor to the same ground line as the rest of the devices being used: Microcontroller, Rock Pi, and Microphone.

Our code for the Servo Motor can be found **HERE**

<u>Signature 6:</u> Move the servo motor to desired direction, Then take picture, save it on Rock Pi, and send it to the Host Computer

Setup:

- Servo Motor (1x)
- Power Brick (Set to 5V)
- USB enabled Webcam
- Macbook Pro (M1 Chip)
- Everything else is the same as the previous signatures

Description:

At this point, different sections of the project had been completed and the code for them was located in different files. A big chunk of the project involved the servo motor and webcam working together. We needed to make sure that when the servo moves in the upwards direction, that there is enough time for the camera to take a picture of the view in front of it and send it to the host computer, before the servo motor moves in the downward direction. In order for this integration to be successful, we had to make sure that Signature 3 was able to work fine. As mentioned in the Signature 3 section above, we utilized the exec command to control each code we had being executed one by one at the right time. One of the reasons we went to look for a command in python that would enable us to execute code from one python file in another is because we had many python files all doing different things. We wanted the execution of these

files to flow one after another without having to add all the code across them into one huge file. We used the “exec” command to execute the servo motor code when a packet has been sent over and in the servo motor code, when the motor has been titled up (we used an if condition to check if it was under a certain number) we executed the camera code.

Our code for the Servo Motor code with the execute function to start the Camera code can be found **HERE**

Send_image.py code can be found **HERE**

<u>Signature 7:</u> Create 3d Printed Box, connect Servo Motor and packets, and demo final project

Setup:

The Software and Hardware can be found below:

- Tinkercad web based 3d printing design
- Makerspace 3d printer and filament
- Nylon string 2ft

Description:

There were two phases to creating the final box design that could be printed out by a 3d printer. I first was dealing with the size of the printing space limit, so I had to make sure my design was small enough to be able to fit completely on the printing surface but also be able to hold the camera we wanted to place there.

The picture below is the first iteration of the box and we were able to get this lower part printed but realized that the walls were too small. There was also another issue where the hinges that were printed were filled in with support filament which is automatically put in by the printer.

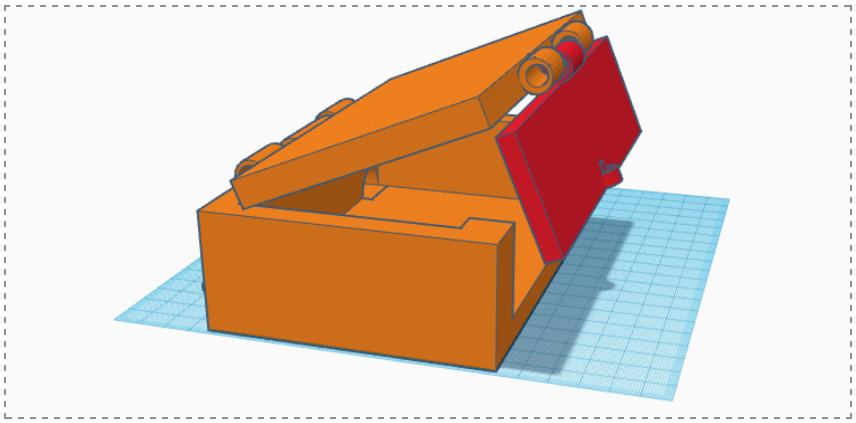


Figure 8: 3D model of box

The second picture is the new lower with old hinges. I say old because we solved the automatic fill issue by making the hinges print vertically instead of horizontally. This lets it print the hinge upright and be able to avoid auto filling as a whole. The weight limit was also a constraint due to the filament being expensive so the maker space staff said it needed to be less than 150 grams per print and less than 200 grams per week printed. I was able to thin all of the walls and doors as well as resize the hinges so they can all be hot glued to the base and doors of the 3d printed box. I had to make a rod to stick into the hinges of the doors so the servo may move the door up and down freely without any type of stress to any part of the box.

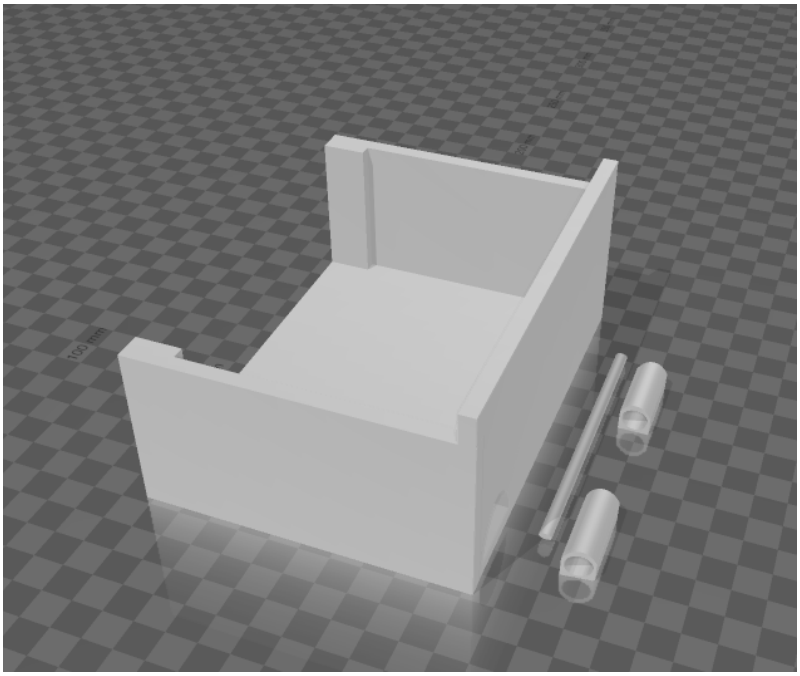


Figure 9: 3D model of box components

Finally assembling the box I only had to add on a ring that was a semicircle shape that would be tied to the rope on the servo we were using to pull the string back. This ring was placed in the hinge between the front and the roof, where the hinge was. This is not well represented in the first picture but this was to make pulling the door easier because it would focus more of the tension upward then behind the box. This was the final piece of the box that was needed to be able to make the box function properly.

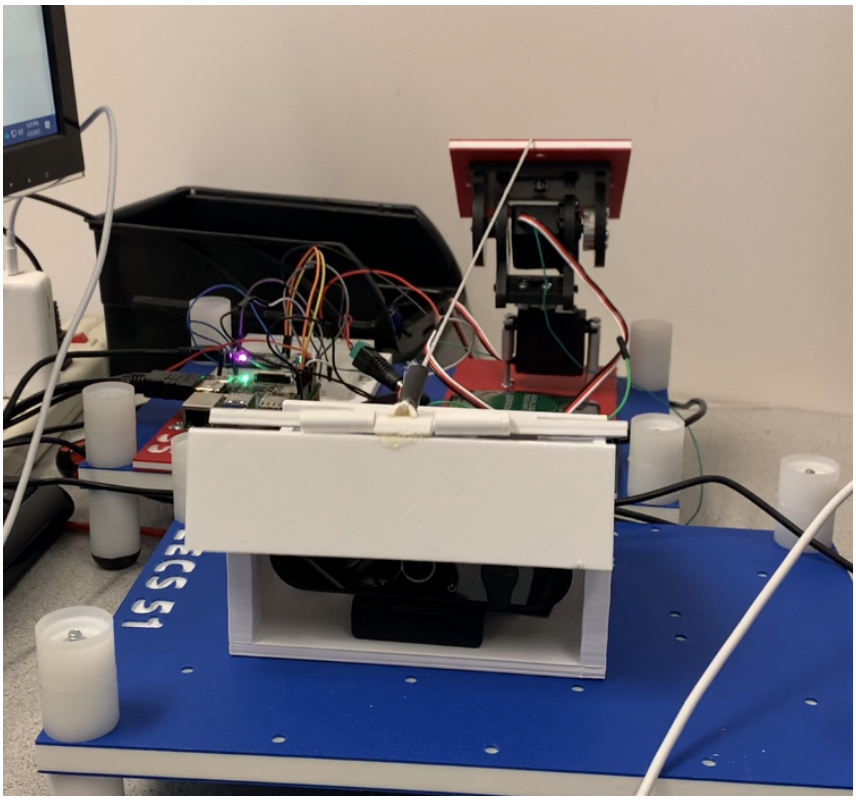


Figure 10: View of Final setup

Our final Main Rock Pi Code to execute all parts of our projects, meaning, the Camera takes a picture and the servo motor moves can be found **HERE**

Our code for the Servo Motor code with the execute function to start the Camera code can be found **HERE**

Ethical Justification of Project

The ethical justification of our project was to ensure that people could have affordable and concealable cameras that were mobile enough to be placed anywhere where valuables could be located. For example, San Francisco has thousands of car robberies. This use case speaks to our commitment to privacy. In the development of our project we were influenced by the Universal Declaration of Human Rights. More specifically, Article 17, where they assert that,

1. Everyone has the right to own property alone as well as in association with others.
2. No one shall be arbitrarily deprived of his property.

If we could have a camera that would be able to take and send a picture to a remote computer there could be a drastic decrease in the amount of break ins.

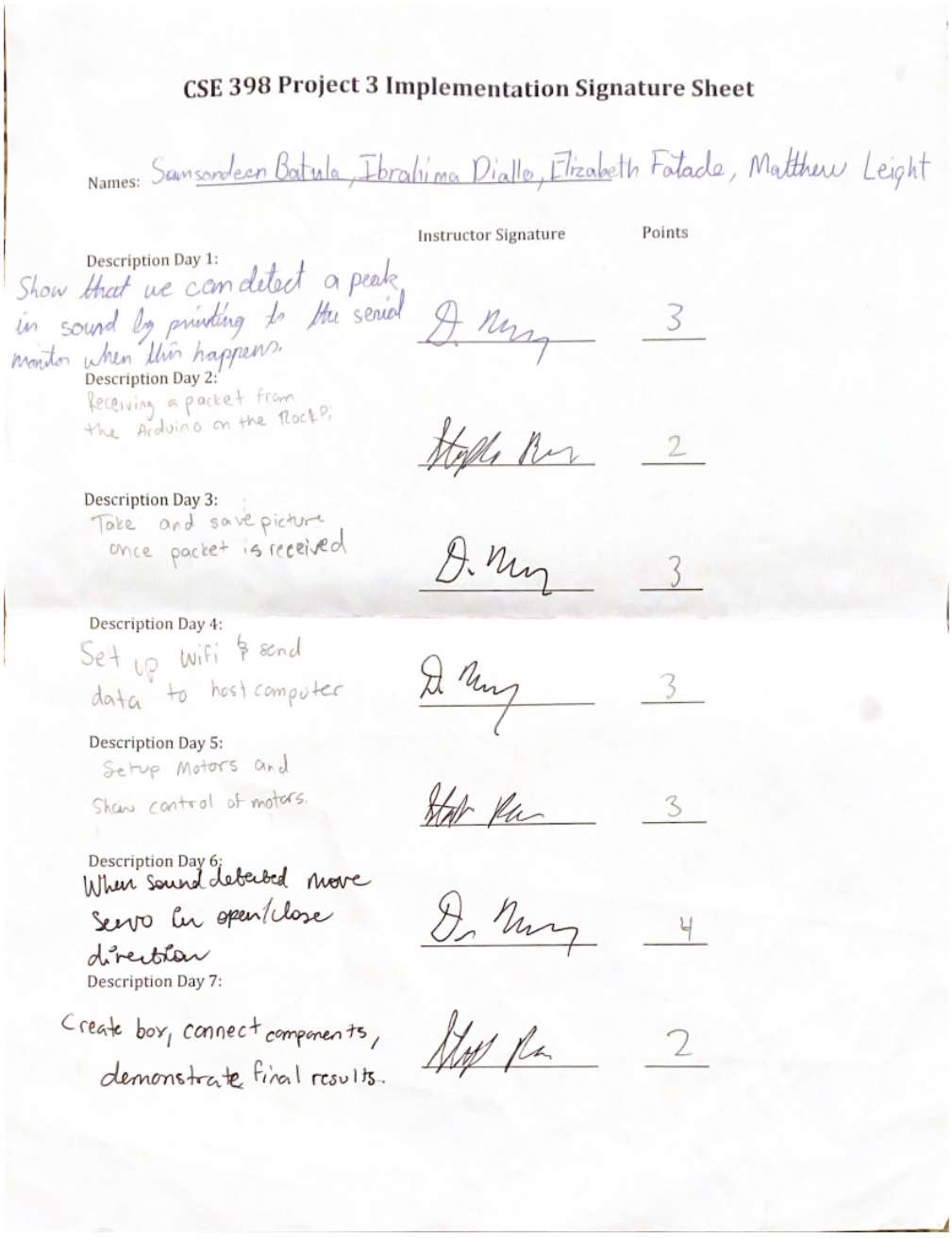
After discussing our project's strengths, the next step was to identify possible ethical weaknesses with our project. One of the first things that came to mind was "What are we going to do with the data that we collect from users?" We could use what we receive from multiple users to inform companies in adjacent industries on who they should advertise to and sell information to them. This would be an example of us violating The Rights Approach outlined in the Markkula Center's Framework for Ethical Decision making. We could also inadvertently violate user privacy by not keeping up with the latest technology and being neglectful with our encryption practices. The use of COBs helps from the Arduino to the Rock Pi but there is another vulnerability from the Rock Pi to the host computer or any other place that you would like to store the report.

We also considered that we not only need to protect the privacy of the user, but the privacy of others that may be involved in its use. Due to the concealability of the camera, there is potential for people to use it for unintended purposes in inappropriate locations like a bedroom, bathroom,

or area that is restricted. There will always be a back and forth battle between guiding our project only towards intended uses and creative people repurposing our device for unintended purposes. These are concerns that future iterations can deal with but it's important we take the time to use The Rights Approach to ensure that users have a degree of privacy.

Conclusion

In our project we were able to successfully create a noise activated and highly mobile camera that is concealed discretely until it would be needed. We had to make design decisions on how to not only complete the whole design, but what hardware to use for each specific section of the project. Each decision determined how much time we put into it in order to get ourselves on track for the final demonstration. During the course of this project, whilst running our code and installing modules and packages, we discovered that if a module or package is installed using the “sudo” command, then the program it is used in has to be run using the “sudo” command as well. We experienced a number of runtime errors because some python packages could only be installed using sudo whilst others



CSE 398 Project 3 Implementation Signature Sheet

Names: Samsondeen Batula, Ibrahima Diallo, Elizabeth Fatade, Matthew Leight

	Instructor Signature	Points
Description Day 1: Show that we can detect a peak in sound by printing to the serial monitor when this happens.		3
Description Day 2: Receiving a packet from the Arduino on the RockPi		2
Description Day 3: Take and save picture once packet is received		3
Description Day 4: Set up WiFi & send data to host computer		3
Description Day 5: Setup Motors and Show control of motors.		3
Description Day 6: When sound detected move servo in open/close direction		4
Description Day 7: Create box, connect components, demonstrate final results.		2

Figure 11: Signed Signature Sheet

Appendix

ByteArray.py (ByteArray Removal Code)

```
#BYTEARRAY REMOVAL CODE
#!/usr/bin/env python
# -*- coding: utf-8 -*-
#
#  bytearrayremoval.py
#
#  Copyright 2022 rock <rock@rockpi-4b>
#
#
encodedBytesString =
"bytearray(b'0.07,0.07,0.07,0.07,0.07,0.07,0.07,0.07,0.11,0.46,4.97,0.11,2.83,0.22,1.06,0.07,0.0
7,0.07,0.07,0.07,0.07,0.07,')"
newEncodedBytesString = encodedBytesString[12:len(encodedBytesString)-3].split(",")
cleanedEncodedBytesString = []
for string in newEncodedBytesString:
cleanedEncodedBytesString.append(float(string))
print(newEncodedBytesString)
print(cleanedEncodedBytesString)
```

CameraPic.py (Camera Picture Code)

```
#cameratest code
import sys
import zmq
import socket
import time
import cv2
from imutils.video import VideoStream
import imagezmq
```

```
sender = imagezmq.ImageSender(connect_to='tcp://10.144.113.180:5555')
rpi_name = socket.gethostname()
capture = cv2.VideoCapture(4)
time.sleep(2)
while capture.isOpened():
ret, frame = capture.read()
if ret == True:
frame = cv2.flip(frame, 90)
cv2.imshow('Frame', frame)
cv2.waitKey(1000)
sender.send_image(rpi_name, frame)
break

time.sleep(1)
capture.release()
cv2.destroyAllWindows()
```

ImageReceive.py (Image Receive Code - receive image on host pc)

```
import cv2
import imagezmq

def main():
    image_hub = imagezmq.ImageHub()

    while True:
        rpi_name, image = image_hub.recv_image()
        cv2.imshow(rpi_name, image)  # 1 window for each RPi
        cv2.waitKey(1)
        image_hub.send_reply(b'OK')

if __name__ == '__main__':
    main()
```

DecodeWrite.py (Decode Write to Waveforms Code)

```
#DECODEDWRITETOWAVEFORMS CODE
import mraa
import time
import sys
import struct
import binascii
```

```
# initialise UART
uart = mraa.Uart("/dev/ttyS4")
#set uart parameters
uart.setBaudRate(9600)
uart.setMode(8, mraa.UART_PARITY_NONE, 1)
uart.setFlowcontrol(False, False)
inputStr = "b'z\x08\x00
\x00\x00\xb4=\x00\x00\x8c=\x00\x00\xa0=\x00\x00\xaa=\x00\x00\x96=\x00\x00\x82=\x00'"
l = len(b'z\x08\x00
\x00\x00\xb4=\x00\x00\x8c=\x00\x00\xa0=\x00\x00\xaa=\x00\x00\x96=\x00\x00\x82=\x00')

charBuffer = []
for i in range(l):
charBuffer.append(inputStr[i])
print("Length output is: ", l)
whatever = uart.write(bytearray(inputStr))
```

Arduino.py (Final Arduino Code)

```
// FINAL CODE (ARDUINO TO ROCK PI)

#include <Wire.h>
#include <PacketSerial.h>
#include <string>
using namespace std;
// Sample window width in mS (50 mS = 20Hz)
const int sampleWindow = 50;
unsigned int sample;
float noiseData[21];
//string noiseData;
string measurements = "";
int eventFlag = 11;
boolean recordFlag = false;

PacketSerial myPacketSerial;
struct send_packet {
  char header;
  String measurements;
};
/************ SETUP ************ SETUP ************ SETUP ************/
void setup(void) {
  delay(100);
  Serial.begin(9600);
```

```
  Serial1.begin(9600);
  Serial.println("Setup called");

  Serial1.println("");
  delay(100);
  myPacketSerial.setStream(&Serial);
myPacketSerial.setStream(&Serial1);  }
/************ LOOP ************ LOOP ************ LOOP ************/
void loop()
{
unsigned long startMillis= millis();  // Start of sample window
unsigned int peakToPeak = 0;   // peak-to-peak level
unsigned int signalMax = 0;
unsigned int signalMin = 1024;
   // collect data for 50 mS
   while (millis() - startMillis < sampleWindow)
   {
      sample = analogRead(0);
      if (sample < 1024)  // toss out spurious readings
      {
         if (sample > signalMax)
         {
            signalMax = sample;  // save just the max levels
         }
         else if (sample < signalMin)
         {
            signalMin = sample;  // save just the min levels
         }
      }
   }
   peakToPeak = signalMax - signalMin;  // max - min = peak-peak amplitude
   double volts = (peakToPeak * 5.0) / 1024;  // convert to volts

   for (int i = 0; i < 20; i++) {
      noiseData[i] = noiseData[i+1];
    }
    noiseData[20] = volts;
   delay(100);

   //Our mic event
   Serial.println(volts);
   if(volts > 1)
   {
    Serial.println("Mic event detected!!");
```

```
    //myPacketSerial.send((uint8_t*)&pkt, sizeof(pkt));
    recordFlag = true;
   }
   //After event happens, start recording
   if(recordFlag == true) { eventFlag--; }

    //Once finished recording
   if(eventFlag == 0)
   {
    send_packet pkt;
    pkt.header = 'z';
    Serial.print("z");
    Serial.print("\n");
    //Add measurements into packet
    /*
    for(int i=0; i<21; i++) {
      pkt.measurements[i] = noiseData[i];
      Serial.print(" Measurement: ");
      Serial.print(noiseData[i]);
      Serial.println(" ");
    }
    */



    //Create string with the measurements
    for(int i=0; i<21; i++) {
      pkt.measurements += noiseData[i];
      pkt.measurements += ",";
    }

    Serial.println(pkt.measurements);
    Serial.println("Sending packet!");
    for (int i = 0; i < pkt.measurements.length(); i++){
      Serial1.print(pkt.measurements.charAt(i));
    }
    //myPacketSerial.send((uint8_t*)&pkt, sizeof(pkt));
    pkt.measurements = "";
    eventFlag = 11;
    recordFlag = false;
   }
}
```

Main.py (Final Main Rock Pi Code)

```
#MAIN CODE FINAL
#!/usr/bin/env python
# -*- coding: utf-8 -*-
#
#  test.py
#
#  Copyright 2022 rock <rock@rockpi-4b>
import mraa
import time
import sys
import struct
import binascii
class DecodeError(Exception):
    pass
def _get_buffer_view(in_bytes):
    mv = memoryview(in_bytes)
    if mv.ndim > 1 or mv.itemsize > 1:
        raise BufferError('object must be a single-dimension buffer of bytes.')
    try:
        mv = mv.cast('c')
    except AttributeError:
        pass
    return mv
def encode(in_bytes):

    """Encode a string using Consistent Overhead Byte Stuffing (COBS).
    Input is any byte string. Output is also a byte string.
    Encoding guarantees no zero bytes in the output. The output
    string will be expanded slightly, by a predictable amount.
    An empty string is encoded to '\\x01'"""

    if isinstance(in_bytes, str):
        raise TypeError('Unicode-objects must be encoded as bytes first')
    in_bytes_mv = _get_buffer_view(in_bytes)
    final_zero = True
    out_bytes = bytearray()
    idx = 0
    search_start_idx = 0
    for in_char in in_bytes_mv:
        if in_char == b'\x00':
            final_zero = True
            out_bytes.append(idx - search_start_idx + 1)
```

```
            out_bytes += in_bytes_mv[search_start_idx:idx]
            search_start_idx = idx + 1
        else:
            if idx - search_start_idx == 0xFD:
                final_zero = False
                out_bytes.append(0xFF)
                out_bytes += in_bytes_mv[search_start_idx:idx+1]
                search_start_idx = idx + 1
        idx += 1
    if idx != search_start_idx or final_zero:
        out_bytes.append(idx - search_start_idx + 1)
        out_bytes += in_bytes_mv[search_start_idx:idx]
    return bytes(out_bytes)
def decode(in_bytes):

    """Decode a string using Consistent Overhead Byte Stuffing (COBS).
    Input should be a byte string that has been COBS encoded. Output
    is also a byte string.
    A cobs.DecodeError exception will be raised if the encoded data
    is invalid."""

    if isinstance(in_bytes, str):
        raise TypeError('Unicode-objects are not supported; byte buffer objects only')
    in_bytes_mv = _get_buffer_view(in_bytes)
    out_bytes = bytearray()
    idx = 0
    if len(in_bytes_mv) > 0:
        while True:
            length = ord(in_bytes_mv[idx])
            if length == 0:
                raise DecodeError("zero byte found in input")
            idx += 1
            end = idx + length - 1
            copy_mv = in_bytes_mv[idx:end]
            if b'\x00' in copy_mv:
                raise DecodeError("zero byte found in input")
            out_bytes += copy_mv
            idx = end
            if idx > len(in_bytes_mv):
                raise DecodeError("not enough input bytes for length code")
            if idx < len(in_bytes_mv):
                if length < 0xFF:
                    out_bytes.append(0)
            else:
```

```
            break
    return bytes(out_bytes)
# initialise UART
uart = mraa.Uart("/dev/ttyS4")
#set uart parameters
uart.setBaudRate(9600)
uart.setMode(8, mraa.UART_PARITY_NONE, 1)
uart.setFlowcontrol(False, False)
flag = 0
while True: #flag == 0:
flag = 1
#if flag == 1:
encodedBytes = bytearray()
count = 0
while True:
if uart.dataAvailable(100):
count += 1
data_byte = uart.read(1)
encodedBytes.extend(data_byte)
print(data_byte)
print(encodedBytes)
if count == 105:
break

encodedBytesString = str(encodedBytes)
print(encodedBytesString)
exec(open("servo.py").read())
```

NoiseDetection.py (NoiseDetection - Signature 1 code Code)

```
/*****************************************
Example Sound Level Sketch for the
Adafruit Microphone Amplifier
*****************************************/
const int sampleWindow = 50; // Sample window width in mS (50 mS = 20Hz)
unsigned int sample;
void setup()
{
  Serial.begin(9600);
}
void loop()
{
```

```
  unsigned long startMillis= millis();  // Start of sample window
  unsigned int peakToPeak = 0;   // peak-to-peak level
  unsigned int signalMax = 0;
  unsigned int signalMin = 1024;

  // collect data for 50 mS
  while (millis() - startMillis < sampleWindow)
  {
    sample = analogRead(0);
    if (sample < 1024)  // toss out spurious readings
    {
      if (sample > signalMax)
      {
        signalMax = sample;  // save just the max levels
      }
      else if (sample < signalMin)
      {
        signalMin = sample;  // save just the min levels
      }
    }
  }
  peakToPeak = signalMax - signalMin;  // max - min = peak-peak amplitude
  double volts = (peakToPeak * 5.0) / 1024;  // convert to volts
  delay(100);
  //Our mic event
  Serial.println(volts);
  if(volts > 3){
   Serial.println("Mic event detected!!");
  }
}
```

ReceivingPackets.py (Receiving packets on rock pi Code)

Code for receiving packets in the rock pi 4 from the arduino

```
#!/usr/bin/env python
# -*- coding: utf-8 -*-a
#
#  test.py
#
#  Copyright 2022 rock <rock@rockpi-4b>
import mraa
import time
```

```
import sys
import struct
import binascii
class DecodeError(Exception):
    pass
def _get_buffer_view(in_bytes):
    mv = memoryview(in_bytes)
    if mv.ndim > 1 or mv.itemsize > 1:
        raise BufferError('object must be a single-dimension buffer of bytes.')
    try:
        mv = mv.cast('c')
    except AttributeError:
        pass
    return mv

def encode(in_bytes):
    """Encode a string using Consistent Overhead Byte Stuffing (COBS).
    Input is any byte string. Output is also a byte string.
    Encoding guarantees no zero bytes in the output. The output
    string will be expanded slightly, by a predictable amount.
    An empty string is encoded to '\\x01'"""

    if isinstance(in_bytes, str):
        raise TypeError('Unicode-objects must be encoded as bytes first')
    in_bytes_mv = _get_buffer_view(in_bytes)
    final_zero = True
    out_bytes = bytearray()
    idx = 0
    search_start_idx = 0
    for in_char in in_bytes_mv:
        if in_char == b'\x00':
            final_zero = True
            out_bytes.append(idx - search_start_idx + 1)
            out_bytes += in_bytes_mv[search_start_idx:idx]
            search_start_idx = idx + 1
        else:
            if idx - search_start_idx == 0xFD:
                final_zero = False
                out_bytes.append(0xFF)
                out_bytes += in_bytes_mv[search_start_idx:idx+1]
                search_start_idx = idx + 1
        idx += 1
    if idx != search_start_idx or final_zero:
        out_bytes.append(idx - search_start_idx + 1)
```

```
        out_bytes += in_bytes_mv[search_start_idx:idx]
    return bytes(out_bytes)

def decode(in_bytes):

    """Decode a string using Consistent Overhead Byte Stuffing (COBS).
    Input should be a byte string that has been COBS encoded. Output
    is also a byte string.
    A cobs.DecodeError exception will be raised if the encoded data
    is invalid."""

    if isinstance(in_bytes, str):
        raise TypeError('Unicode-objects are not supported; byte buffer objects only')
    in_bytes_mv = _get_buffer_view(in_bytes)
    out_bytes = bytearray()
    idx = 0
    if len(in_bytes_mv) > 0:
        while True:
            length = ord(in_bytes_mv[idx])
            if length == 0:
                raise DecodeError("zero byte found in input")
            idx += 1
            end = idx + length - 1
            copy_mv = in_bytes_mv[idx:end]
            if b'\x00' in copy_mv:
                raise DecodeError("zero byte found in input")
            out_bytes += copy_mv
            idx = end
            if idx > len(in_bytes_mv):
                raise DecodeError("not enough input bytes for length code")
            if idx < len(in_bytes_mv):
                if length < 0xFF:
                    out_bytes.append(0)
            else:
                break
    return bytes(out_bytes)

# initialise UART
uart = mraa.Uart("/dev/ttyS4")
#set uart parameters
uart.setBaudRate(9600) uart.setMode(8, mraa.UART_PARITY_NONE, 1)
uart.setFlowcontrol(False, False)

encodedBytes = bytearray()
```

```
count = 0
while True:
if uart.dataAvailable(100):
count += 1
data_byte = uart.read(1)
encodedBytes.extend(data_byte)
print(data_byte)
print(encodedBytes)
if count == 105:
break
```

Servo1.py (Original Servo Motor code Code)

```
#SERVO CODE
import mraa
import time

#Initialize PWM
y = mraa.Pwm(11)

# Set PWM period
y.period_us(20000)


# Enable PWM
y.enable(True)

# Server motor was inverted so in order to get the actual duty cycle of 2.5% we had to subtract it
from 1
value = (1.0 - 0.025)
y.write(value)
time.sleep(1)


count = 0
while count < 50:

# Write PWM value
y.write(value)
time.sleep(0.05)

value = value - 0.001
print("Value: ", value)
```

```
# If the value is gets to a duty cycle of 12.5% then it should spin the other way to the 2.5% duty cycle
if value <= (1.0 - 0.125):
value = (1.0 - 0.025)
y.write(value)
time.sleep(2)

if count == 7:
‘’’ADD EXEC FUNCTION HERE TO START CAMERA CODE’’’
count += 1
```

Servo2.py (Servo Motor code *WITH PICTURE EXEC* Code)

```
#SERVO CODE
import mraa
import time

#Initialize PWM
y = mraa.Pwm(11)

# Set PWM period
y.period_us(20000)


# Enable PWM
y.enable(True)

# Server motor was inverted so in order to get the actual duty cycle of 2.5% we had to subtract it from 1
value = (1.0 - 0.025)
y.write(value)
time.sleep(1)


count = 0
while count < 50:

# Write PWM value
y.write(value)
time.sleep(0.05)

value = value - 0.001
```

```
print("Value: ", value)

# If the value is gets to a duty cycle of 12.5% then it should spin the other way to the 2.5% duty cycle
if value <= (1.0 - 0.125):
value = (1.0 - 0.025)
y.write(value)
time.sleep(2)

if count == 7:
exec(open("cameraTest.py").read())
count += 1
```