Motion Activated Camera
IoT Mini-Project “WatchDog”

Martin Pavlinov Zhelev

CMP408: IoT and Cloud Secure Development

BSc (Hons) Ethical Hacking Year 4

2023/2024

Note that Information contained in this document is for educational purposes.

.

Contents

1 Introduction 1

1.1 Relevance to IoT and Cloud Secure Development 2

1.2 Objectives 2

2 Methodology 3

2.1 Raspberry PI Zero W 3

2.2 Cloud 7

3 Conclusion 10

3.1 Discussion 10

3.2 Future Work 10

4 References 11

5 Appendices 14

5.1 Appendix A – Client-side code 14

5.1.1 WatchDog.py 14

5.1.2 Button_detector.c 18

5.1.3 Button_userspace.c 21

5.1.4 uploadAWS.py 22

5.1.5 processUpload.sh 24

5.2 Appendix B – AWS Code 24

5.2.1 s3-start_step_functions.py 24

5.2.2 rekognition-label_image.py 25

5.2.3 rekognition-examine_labels_result.py 25

5.2.4 ssm-executeUploadOnPI.py 26

5.2.5 ses-send_alert.py 27

5.2.6 s3-move_to_processed.py 29

5.2.7 ses-error_catcher.py 29

5.2.8 stateMachine.json 30

.

1 INTRODUCTION

This project aims to develop a motion activated security camera with human detection. To do this the following hardware and software will be used:

Name	Use
Raspberry Pi Zero W	Used to run the software and to connect the different modules with its General-Purpose Input/Output (GPIO) pins. Also utilised to connect the camera by using its Camera Serial Interface (CSI).
Button	Used in for a LKM module, which activates a userspace script when it detects an interrupt caused by the press of the button.
ZeroCam camera	Utilised for camera recording
Buzzer	Used to signal a file upload to AWS when camera stops detecting motion
TM1638 board	Utilised for menu navigation.
SSD1306 display	Displays simple menu for the user to interact with the script without a terminal session to the raspberry pi.
Sentry-Picam	Software was used to record videos with the PI CAMERA when motion is detected.
AWS Identity Access Management (IAM)	Followed best practices to control access to the utilised AWS resources
AWS Simple Storage Service (S3)	Stores the videos and thumbnails recorded by the camera.
AWS Lambda	Runs code in response to events.
AWS Step Functions	Used to create a workflow that utilises several lambda functions.
AWS Rekognition	Utilised to detect if a human is present in the camera recording.
AWS Systems Manager	Used to perform functions on the Raspberry Pi Zero W remotely by Lambda.
AWS Simple Email Service (SES)	Sends emails to the user containing thumbnail and recording of detected human motion.

Table 1: Used hardware and software.

1.1 Relevance to IoT and Cloud Secure Development

This project is relevant to IoT, because it utilised the Raspberry Pi Zero W's GPIO pins to connect several devices as well as its CSI interface to connect a camera module. To successfully implement the following devices had to be correctly connected and configured:

- Button
- ZeroCam camera
- TM1638 board
- SSD1306 display
- Buzzer

Alongside this, to detect a button press by the user, a Loadable Kernel Module (LKM) was developed that would get loaded at runtime and would detect a button press. When it detects a button press the kernel module would send a signal to a userspace application to launch the python script that is used to control all the connected devices needed for the created Motion Activated Camera IoT device. During all software development best development practices were followed.

In terms of Cloud Secure development, this project utilised several AWS Services to create a workflow that would inspect the thumbnails of videos recorded by the Motion Activated Camera for the presence of humans to inform the user that a human was detected. The AWS cloud environment was correctly configured by following AWS environment best practices using AWS IAM users, roles, and policies.

1.2 Objectives

The project has the following objectives:

- Create a script which launches motion detection camera recording software.
- Add functions to script to implement a TM1648 board and a SSD1306 display.
- Configure secure AWS environment using AWS Identity Access Management.
- Upload footage to AWS Simple Storage Service as soon as it finishes recording.
- Use AWS Lamba and Step Functions to detect file upload and sent footage to AWS Rekognition.
- Determine if human is present using AWS Rekognition
- Send an email to the user informing them of human detection using AWS Simple Email Service.

2 METHODOLOGY

2.1 RASPBERRY PI ZERO W

The begin development the first step the developer took was to examine the different software that would allow him to record videos using the ZeroCam camera he was provided with. The developer determined 3 pieces of Software that could be implemented. Those were "Pikrellcam" (billw2, 2020), "Motion" (Motion, n.d.) and "SentryPicam" (TinkerTurtle, 2022). After conducting some testing and examining the features of all of them, the developer decided to use SentryPicam because the rest of the software was not capable of recording high frame rate video on the Raspberry Pi Zero W, because of the complexity of motion detection being performed.

The first program developed was called "WatchDog.py". It would utilise the "adafruit-ssd1306" library to render a main menu on a ssd1306 display module. Alongside this the "rpi_TM1638" library (thilaire, 2020) was used to allow a seven-segment display board to be included for menu control (Figure 1). When the program was running it would allow the user to start the "SentryPicam" software and change its settings.

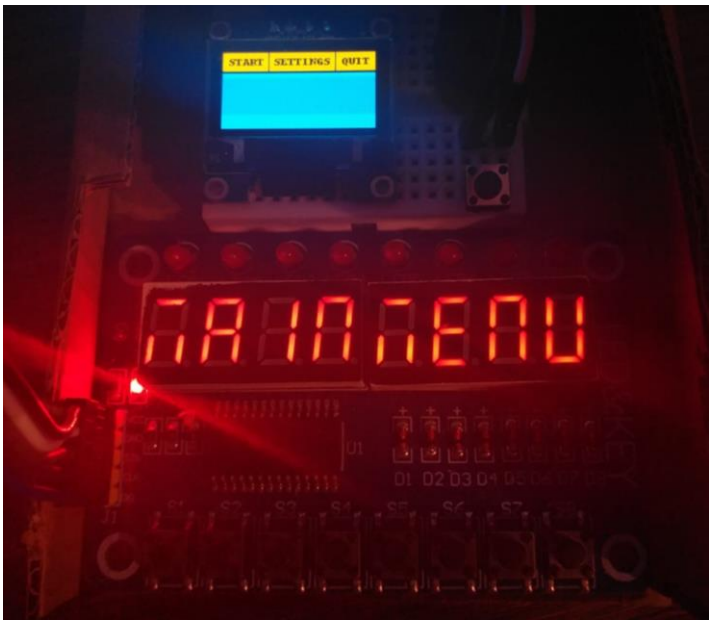

Figure 1: WatchDog.py running.

Following this a LKM module was created called "button_detector.ko". It would run in kernel space and wait for an interrupt caused by a button press to send a signal to separate userspace program called "button_userspace.c" which would execute the "WatchDog.py" script (Figure 2 and Figure 3). The userspace program was used as an intermediary because it is not advisable to execute a python script directly from kernel space, because it might lead to system

instability. To create the LKM module and userspace program a tutorial was followed (4GNU_Linux, 2022) and (Molloy, 2015).

```
[ 3436.712025] button_detector: loading out-of-tree module taints kernel.
[ 3436.712764] BUTTON DETECTOR KERNEL MODULE: INITILIASING
[ 3436.713013] BUTTON DETECTOR KERNEL MODULE: SUCCESSFULLY INITILIASED
[ 3436.713029] GPIO PIN IS MAPPED TO IRQ: 160
[ 3452.203857] INTERRUPT TRIGGERED
martin@raspberryMartin:~/WatchDog $
```

Figure 2: Kernel module dmesg.

```
martin@raspberryMartin:~/WatchDog $ ./button_userspace
BUTTON USERSPACE: WAITING
BUTTON USERSPACE: RECEIVED RUN SIGNALBUTTON USERSPACE: WAITING
```

Figure 3: Userspace program receiving signal.

To execute a file upload to AWS a shell script called “processUpload.sh” was developed. The shell script would get executed after “SentryPicam” finishes detecting motion. It would launch a python script called “uploadAWS.py” that would cause a buzzer to go off and then would locate the thumbnail of the recording made by “SentryPicam” and upload it to AWS S3.

All the code that was developed for the Raspberry Pi Zero W can be found in Appendix A – Client-side code. See Figure 4 below for a photo of the fully assembled device, Figure 5 for a schematic and Figure 6 for a flowchart of the client-side processes.

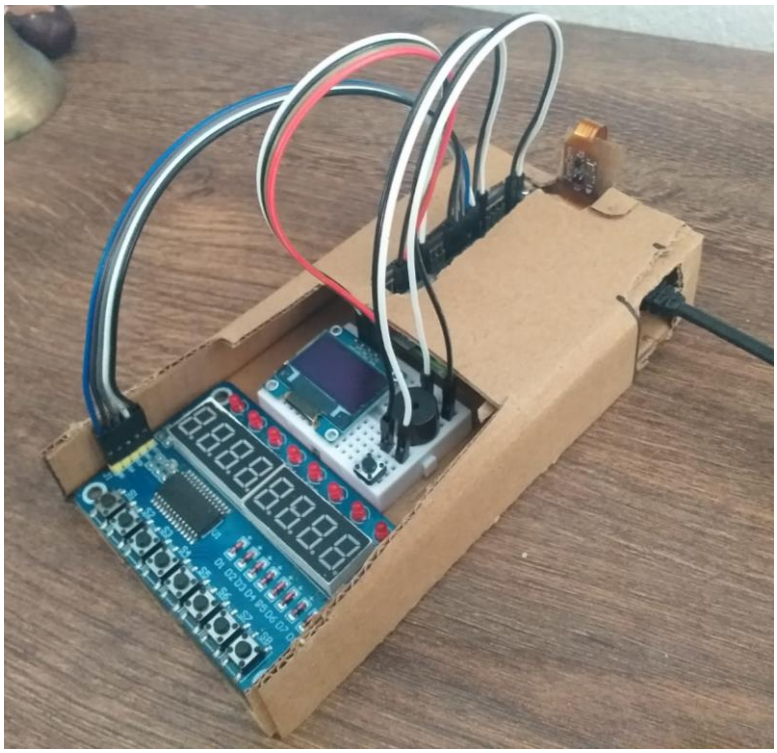

Figure 4: Fully assembled device.

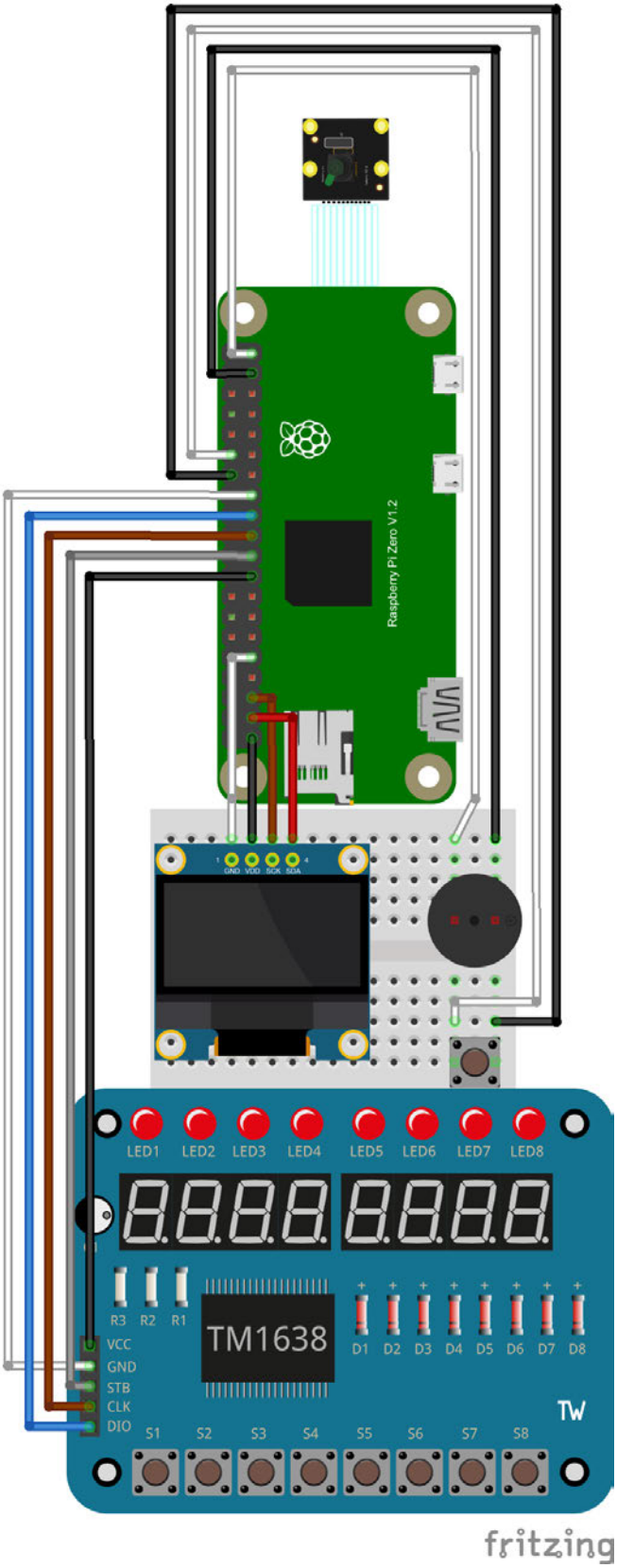

Figure 5: Schematic of device

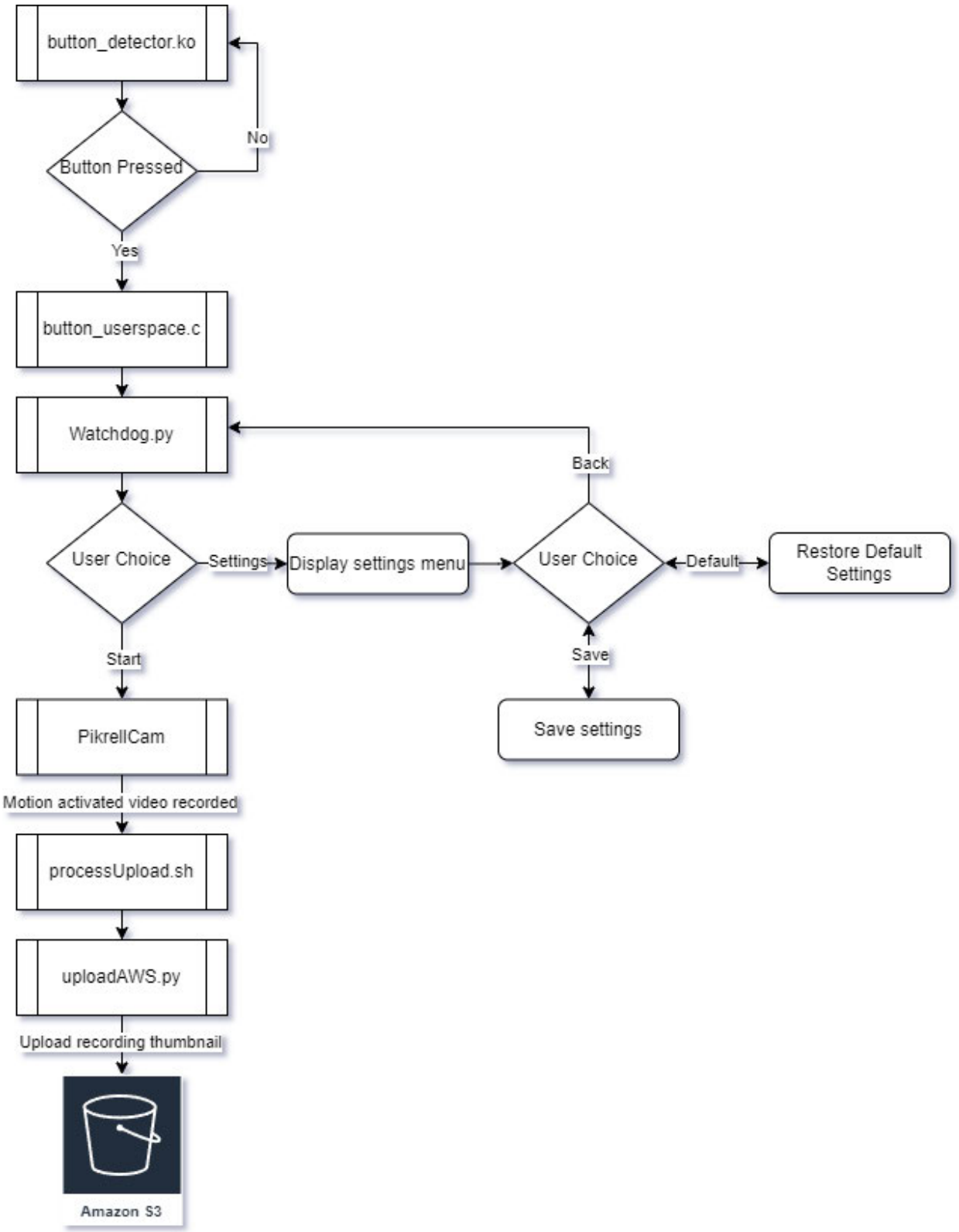

Figure 6: Client side processes

2.2 CLOUD

For the cloud development part of the project 7 python scripts were developed that would execute from AWS Lambda based on certain triggers as well as by utilising AWS Step Function State Machines workflow (Figure 9). All the code that was developed for the Cloud can be found in Appendix B – AWS Code.

The first script in the workflow called “s3-start_step_function.py” would execute when an upload to the “/upload” directory in an AWS S3 bucket called “watchdogstorage” is detected. To detect an upload the trigger functionality of AWS Lambda was utilised. After an upload is detected, this script would start a Step Function State Machine which would be used for the workflow of the rest of the developed AWS Lambda functions.

The second script called “rekognition-label_image” would be executed as soon as the Step Function State Machine is started. It would receive the name of the AWS S3 bucket as well as the name of the file as input and would use those to get the labels describing the objects in the image that were generated by AWS Rekognition for the uploaded file.

After labels are generated, they would be sent to another AWS Lamba script called “rekognition-examine_labels_results” that would examine all the labels for specific keywords and depending on the result would set a boolean called “human” to true to signify that a human was detected or false to signify no human was detected.

Following this the Step Functions State Machine would read the output and depending on the value of the “human” variable would choose which script would be executed next. For the purposes of this report the workflow if a human is detected would be followed so all developed scripts can be explained.

Having confirmed the existence of a human the Step Functions State Machine would execute the “ssm-executeUploadOnPI” AWS Lambda script that would utilise the AWS Systems Manager functionality to execute the “uploadAWS.py” script on the Raspberry Pi Zero W with the “-V” flag which would indicate to the script to upload the “.mp4” video that was recorded by “SentryPicam” to “processed/positive” in the “watchdogstorage” AWS S3 bucket. The AWS Lamba script would wait for the file upload to finish before exiting to ensure the rest of the scripts work correctly.

The next function that would get ran would be “ses-send_alert”. It would utilise AWS SES to send an email to the user that informs them of the detected motion and contains the thumbnail and recording that were created. (Figure 7)

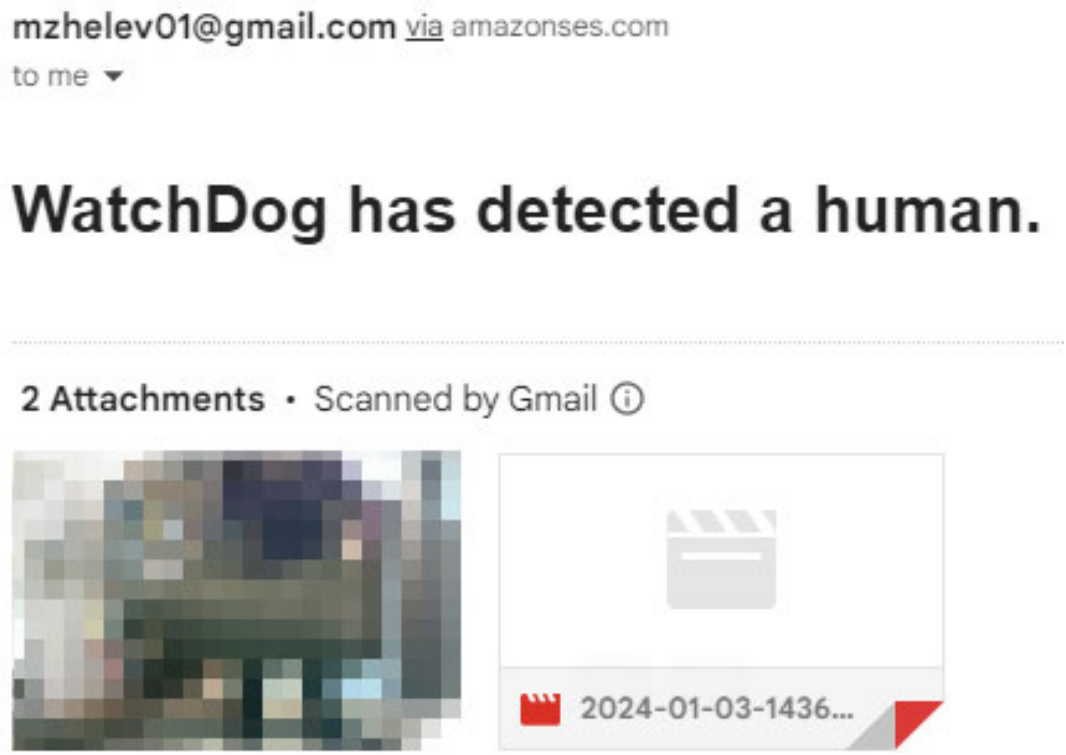

Figure 7. Email informing user of detected human (blurred for privacy)

Finally, the final step in the workflow would be the AWS Lamba script called “s3-move_to_processed” that would move the thumbnail that was uploaded to “upload” in the “watchdogstorage” s3 bucket to “processed/positive” in the same bucket.

During the execution of the workflow if any errors occur, they would be caught by send to an error catcher called “ses-error_catcher” that would send the user an email informing them of the error. (Figure 8)

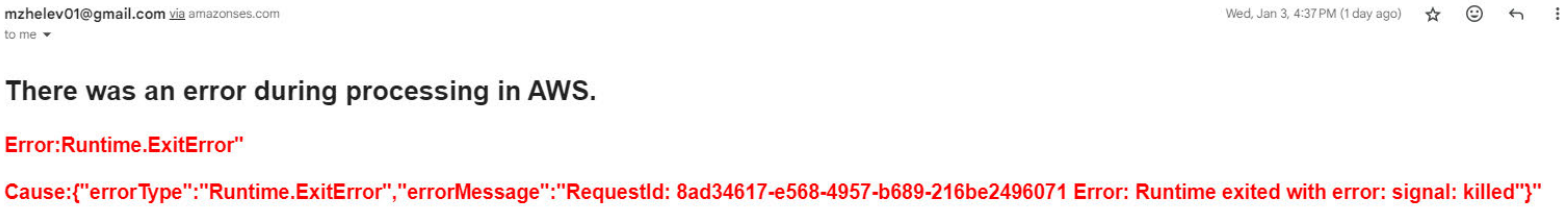

Figure 8: Email informing of error.

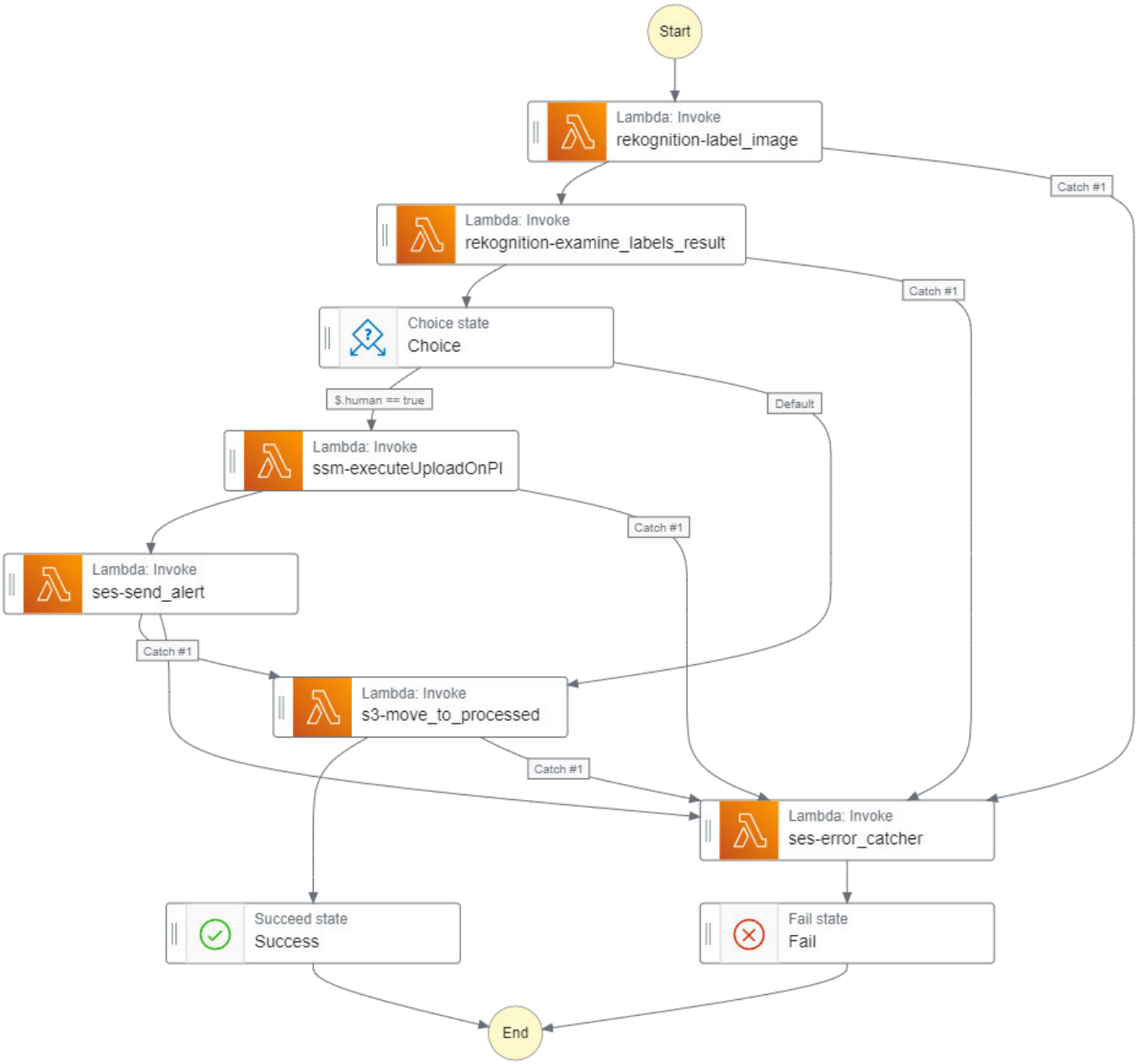

Figure 9: Step function workflow.

3 CONCLUSION

3.1 DISCUSSION

The project successfully demonstrates the use of IoT and Cloud technologies to create a motion-activated security camera that notifies the user if a human is detected.

On the client side this was achieved by utilising a Raspberry Pi Zero W, alongside several peripherals (ZeroCam camera, TM1638 board, SSD1306 display, buzzer, and a button) as well as custom scripts and open-source software called Sentry-Picam to achieve motion detection.

For the cloud part of the project through the user of custom AWS Lambda scripts and several AWS services (AWS IAM, AWS S3, AWS Step Functions, AWS Rekognition, AWS Systems Manager and AWS SES) the developer was able to implement human recognition, video storage and email notifications.

3.2 FUTURE WORK

The IoT device created by this project was fully functional and capable of correctly creating recordings of human motion in most scenarios. Despite that success there are several improvements that could be made if more resources or time is available.

Firstly, the settings menu for the script displayed on the SSD1306 could be further developed to allow the user to change the flags which will be passed to “SentryPicam” for its execution. By adding this feature various settings such as FPS and bitrate of recordings could be changed.

Second, while the developer wanted to implement two LEDs that would indicate that the LKM was active and was sending a signal that was not possible, because neither of the LEDS which he was provided with were functional and as such that was not possible to implement at the current time.

Finally, a script could be developed that would allow the easy installation of all the software needed for the running of the IoT device to increase its ease of installation for novice users.

4 REFERENCES

4GNU_Linux, J., 2022. *Let's code a Linux Driver - 15: Sending a signal from Kernel to Userspace - YouTube.* [Online]
Available at: https://www.youtube.com/watch?v=nt_z07t7qMc
[Accessed 16 12 2023].

ada, l., 2012. *Python Usage | Monochrome OLED Breakouts | Adafruit Learning System.* [Online]
Available at: https://learn.adafruit.com/monochrome-oled-breakouts/python-wiring
[Accessed 3 12 2023].

Anon., 2015. *How to send signal from kernel to user space - Stack Overflow.* [Online]
Available at: https://stackoverflow.com/questions/31646466/how-to-send-signal-from-kernel-to-user-space
[Accessed 15 12 2023].

Anon., 2019. *GitHub - markwest1972/smart-security-camera: A Pi Zero and Motion based webcamera that forwards images to Amazon Web Services for Image Processing.* [Online]
Available at: https://github.com/markwest1972/smart-security-camera
[Accessed 4 12 2023].

AWS, n.d. *Amazon SES examples using SDK for Python (Boto3) - AWS SDK Code Examples.* [Online]
Available at: https://docs.aws.amazon.com/code-library/latest/ug/python_3_ses_code_examples.html
[Accessed 6 12 2023].

AWS, n.d. *Analyzing images with an AWS Lambda function - Rekognition.* [Online]
Available at: https://docs.aws.amazon.com/rekognition/latest/customlabels-dg/ex-lambda.html
[Accessed 6 12 2023].

AWS, n.d. *Detecting labels in an image - Amazon Rekognition.* [Online]
Available at: https://docs.aws.amazon.com/rekognition/latest/dg/labels-detect-labels-image.html
[Accessed 6 12 2023].

AWS, n.d. *Invoke a Lambda function from an Amazon S3 trigger - AWS Lambda.* [Online]
Available at: https://docs.aws.amazon.com/lambda/latest/dg/example_serverless_S3_Lambda_section.html
[Accessed 6 12 2023].

AWS, n.d. *Sending raw email using the Amazon SES API - Amazon Simple Email Service.* [Online]
Available at: https://docs.aws.amazon.com/ses/latest/dg/send-email-raw.html
[Accessed 7 12 2023].

billw2, 2020. *GitHub - billw2/pikrellcam: Raspberry Pi motion vector detection program with OSD web interface..* [Online]
Available at: https://github.com/billw2/pikrellcam?tab=readme-ov-file
[Accessed 4 12 2023].

Molloy, D. D., 2015. *Writing a Linux Loadable Kernel Module (LKM) - Interfacing to GPIOs | derekmolloy.ie.* [Online]
Available at: http://derekmolloy.ie/kernel-gpio-programming-buttons-and-leds#Example_1_Button_Press_LED_Light_LKM
[Accessed 15 12 2023].

Motion, n.d. *Motion.* [Online]
Available at: https://motion-project.github.io/
[Accessed 2 12 2023].

Narkar, S., 2020. *Image Analysis using AWS Rekognition via Lambda Function.* [Online]
Available at: https://hackernoon.com/image-analysis-using-aws-rekognition-via-lambda-function-hvq3uvy
[Accessed 6 12 2023].

Raspberry Pi Foundation, n.d. *Using a buzzer | Physical Computing with Python | Python | Coding projects for kids and teens.* [Online]
Available at: https://projects.raspberrypi.org/en/projects/physical-computing/8
[Accessed 10 12 2023].

Raspberry Pi, 2023. *Raspberry Pi Documentation - Camera software.* [Online]
Available at: https://www.raspberrypi.com/documentation/computers/camera_software.html
[Accessed 3 12 2023].

Sampathkumar, R., 2018. *Manage Raspberry Pi devices using AWS Systems Manager | AWS Cloud Operations & Migrations Blog.* [Online]
Available at: https://aws.amazon.com/blogs/mt/manage-raspberry-pi-devices-using-aws-systems-manager/
[Accessed 15 12 2023].

Singh, R. R., 2020. *Email S3 Objects as Attachment.* [Online]
Available at: https://mrrishisingh.medium.com/email-s3-objects-as-attachment-7a534e78601c
[Accessed 7 12 2023].

ssandbac, n.d. *Raspberry Pi 3 Motion Detection Camera With Live Feed.* [Online]
Available at: https://www.instructables.com/Raspberry-Pi-Motion-Detection-Security-Camera/
[Accessed 3 12 2023].

thilaire, 2020. *GitHub - thilaire/rpi-TM1638: Raspberry Pi driver for chained TM1638 boards.* [Online]
Available at: https://github.com/thilaire/rpi-TM1638
[Accessed 3 12 2023].

TinkerTurtle, 2022. *GitHub - TinkerTurtle/Sentry-Picam: A simple wildlife camera for Raspberry Pis..* [Online]
Available at: https://github.com/TinkerTurtle/Sentry-Picam
[Accessed 4 12 2023].

West, M., 2023. *Smarten up your Pi Zero Web Camera with Image Analysis and Amazon Web Services (Part 2).* [Online]
Available at: https://www.bouvet.no/bouvet-deler/utbrudd/smarten-up-your-pi-zero-web-camera-with-image-analysis-and-amazon-web-services-part-2
[Accessed 2 12 2023].

West, M., n.d. *Smarten up your Pi Zero Web Camera with Image Analysis and Amazon Web Services (Part 1).* [Online]
Available at: https://www.bouvet.no/bouvet-deler/utbrudd/smarten-up-your-pi-zero-web-camera-with-image-analysis-and-amazon-web-services-part-1
[Accessed 2 12 2023].

5 APPENDICES

5.1 APPENDIX A – CLIENT-SIDE CODE

5.1.1 WatchDog.py

```
from time import sleep
from rpi_TM1638 import TMBoards

import board
import busio
import digitalio
from PIL import Image, ImageDraw, ImageFont
import adafruit_ssd1306

import signal
import subprocess
import os

## TM1638 SETTINGS
DIO = 11
CLK = 9
STB = 10

## OLED SETTINGS
WIDTH = 128
HEIGHT = 64
BORDER = 5
SCL = 3
SDA = 2

# Define the Reset Pin
oled_reset = None
# instanciante Oled
# Use for I2C.
i2c = busio.I2C(SCL, SDA)  # uses board.SCL and board.SDA
oled = adafruit_ssd1306.SSD1306_I2C(WIDTH, HEIGHT, i2c, addr=0x3C)
# instanciante TMboard
TM = TMBoards(DIO, CLK, STB, 0)
TM.clearDisplay()
tmOff = False

logo = Image.open("logo.png")
logo_r = logo.resize((WIDTH,HEIGHT), Image.BICUBIC)
```

```
logo_bw = logo_r.convert("1")

# Display image
oled.image(logo_bw)
oled.show()
sleep(1)

image = Image.new("1", (oled.width, oled.height))
draw = ImageDraw.Draw(image)

def clearTM():
    TM.clearDisplay()
    TM.clearDisplay()

def handler(signum, frame):
    res = input("Ctrl-c was pressed. Do you really want to exit? y/n ")
    if res == 'y':
        clearTM()
        oled.fill(0)
        oled.show()
        exit(1)

signal.signal(signal.SIGINT, handler)

def drawText(text: str):

    # Load default font.
    font = ImageFont.load_default()

    # Draw Some Text
    (font_width, font_height) = font.getsize(text)
    draw.text(
        (oled.width // 2 - font_width // 2, oled.height // 2 - font_height // 2),
        text,
        font=font,
        fill=0,
    )

    # Display image
    oled.image(image)
    oled.show()

# Used to display mainMenu
def mainMenu():
    TM.segments[0] = 'MAINMENU'
```

```
    # Draw background
    draw.rectangle((0, 0, oled.width, oled.height), outline=255, fill=255)

    # Load font
    font = ImageFont.load_default()

    draw.text((5,2), "START", font=font, fill=0,)
    draw.text((43,2), "SETTINGS", font=font, fill=0,)
    draw.text((100,2), "QUIT", font=font, fill=0,)

    draw.line([(38, 0), (38,15)], fill=0, width=1)
    draw.line([(94, 0), (94,15)], fill=0, width=1)

    # Display image
    oled.image(image)
    oled.show()

# Used to display mainMenu
def settingsMenu():
    TM.segments[0] = 'SETTINGS'
    draw.rectangle((0, 0, oled.width, oled.height), outline=255, fill=255)

    # Load font
    font = ImageFont.load_default()

    draw.text((5,2), "SAVE", font=font, fill=0,)
    draw.text((43,2), "DEFAULTS", font=font, fill=0,)
    draw.text((100,2), "BACK", font=font, fill=0,)
    draw.text((5,20), "1. Bitrate", font=font, fill=0,)
    draw.text((5,30), "2. FPS", font=font, fill=0,)
    draw.text((5,40), "3. Beep", font=font, fill=0,)
    draw.text((5,50), "4. Port", font=font, fill=0,)
    draw.line([(38, 0), (38,15)], fill=0, width=1)
    draw.line([(94, 0), (94,15)], fill=0, width=1)

    # Display image
    oled.image(image)
    oled.show()

# Clears board and display before quitting
def quit():
    TM.clearDisplay()
    TM.clearDisplay()
    oled.fill(0)
```

```
    oled.show()
    exit(1)

# Turns display off
def screenOFF():
    global tmOff
    tmOff = True
    clearTM()
    TM.leds[0] = True
    TM.leds[1] = True
    TM.leds[2] = True
    TM.leds[3] = True
    TM.leds[4] = True
    TM.segments[0] = "OLED   5"
    for i in range (4, -1, -1):
        sleep(1)
        TM.leds[i] = False
        TM.segments[7] = f"{i}"
    TM.segments[0] = "SCRN OFF"
    oled.write_cmd(0xAE)

# Turns display on
def screenON():
    global tmOff
    tmOff = False
    oled.write_cmd(0xaf)
    TM.segments[0] = "MAINMENU"

def main():
    inMain = True
    inSettings = False
    mainMenu()
    while True:
        if (TM.switches[0] and inMain):
            TM.leds[0] = True
            drawText("STARTED")
            screenOFF()
            os.system("sudo /home/martin/WatchDog/sentry-picam -record -rot 180 -mthreshold 12 -run
/home/martin/WatchDog/scripts/processUpload.sh")
        elif (TM.switches[0] and inSettings):
            TM.leds[0] = True
            drawText("SAVED")
            sleep(1)
            settingsMenu()
            TM.leds[0] = False
```

```
    elif (TM.switches[1] and inMain):
        TM.leds[1] = True
        inMain = False
        inSettings = True
        settingsMenu()
        TM.leds[1] = False
    elif (TM.switches[1] and inSettings):
        TM.leds[1] = True
        drawText("DEFAULTs")
        sleep(1)
        settingsMenu()
        TM.leds[1] = False
    elif (TM.switches[2] and inMain):
        TM.leds[2] = True
        quit()
    elif (TM.switches[2] and inSettings):
        TM.leds[2] = True
        inMain = True
        inSettings = False
        mainMenu()
        TM.leds[2] = False
    elif (TM.switches[7] and tmOff):
        TM.leds[7] = True
        sleep(1)
        screenON()
        TM.leds[7] = False
main()
```

5.1.2 Button_detector.c

```
#include <linux/module.h>
#include <linux/gpio.h>
#include <linux/interrupt.h>
#include <linux/kernel.h>
#include <linux/fs.h>

MODULE_LICENSE("GPL");
MODULE_AUTHOR("Martin Zhelev");
MODULE_DESCRIPTION("Button Press Kernel Module");

// Variables for WatchDog script
#define SIG_RUN 20
#define INIT_WATCHDOG 0x65
#define REM_WATCHDOG 0x66
static struct task_struct *task = NULL;
```

```
#define GPIO_NUMBER 1
#define MAJOR_NUMBER 70

unsigned int gpio_irq;

int err;

// Signal handler used to detect button press
static irq_handler_t button_interrupt_handler(unsigned int irq, void *dev_id, struct pt_regs *regs) {
   printk("INTERRUPT TRIGGERED\n");

   struct siginfo info;

   if(task != NULL) {
      memset(&info, 0, sizeof(info));
      info.si_signo = SIG_RUN;
      info.si_code = SI_QUEUE;

   /* Send the signal */
   err = send_sig_info(SIG_RUN, (struct kernel_siginfo *) &info, task);
   if(err < 0)
      printk("Error sending signal\n");
   }

   return (irq_handler_t) IRQ_HANDLED;
}

// Used to get process id of userspace script
static long int button_ioctl(struct file *file, unsigned cmd, unsigned long arg)
{
   // Get pid of task
   if(cmd == INIT_WATCHDOG) {
      task = get_current();
      printk("USERSPACE APP PID %d REGISTERED \n", task->pid);
   } else if (cmd == REM_WATCHDOG) {
      task = NULL;
      printk("USERSPACE APP DEREGISTERED \n");
   }
   return 0;
}

static int button_release(struct inode *device_file, struct file *instance) {
   if(task != NULL)
      task = NULL;
   return 0;
```

```
}

struct file_operations Fops = {
   .release = button_release,
   .unlocked_ioctl = button_ioctl,
};

static int __init button_detector_init(void) {
   printk(KERN_INFO "BUTTON DETECTOR KERNEL MODULE: INITILIASING\n");

   // Request GPIO pin and configure it as an input with pull-up resistor
   if (!gpio_is_valid(GPIO_NUMBER)) {+
      printk("BUTTON KERNEL MODULE: INVALID GPIO");
   }

   // The return value is zero for success, else a negative errno.
   err = gpio_request(GPIO_NUMBER, "button_gpio");
   if (err < 0) {
      printk(KERN_ERR "BUTTON DETECTOR KERNEL MODULE: GPIO REQUEST FAILED\n");
      return err;
   }

   // The return value is zero for success, else a negative errno.
   err = gpio_direction_input(GPIO_NUMBER);
   if (err < 0) {
      printk(KERN_ERR "BUTTON DETECTOR KERNEL MODULE: GPIO DIRECTION INPUT FAILED\n");
      return err;
   }
   gpio_set_debounce(GPIO_NUMBER, 100); /* Adjust this number to sync with circut */

   // Request the IRQ for the button GPIO pin to setup interrupt
   gpio_irq = gpio_to_irq(GPIO_NUMBER);

   // The return value is zero for success, else a negative errno.
   err = request_irq(gpio_irq, (irq_handler_t) button_interrupt_handler, IRQF_TRIGGER_FALLING,
"BUTTON_IRQ", NULL);
   if (err < 0) {
      printk(KERN_ERR "BUTTON DETECTOR KERNEL MODULE: IRQ REQUEST FAILED\n");
      gpio_free(GPIO_NUMBER);
      return err;
   }

   // The return value is zero for success, else a negative errno.
   err = register_chrdev(MAJOR_NUMBER, "BUTTON KERNEL MODULE IRQ SIGNAL", &Fops);
   if (err < 0){
```

```
        printk(KERN_ALERT "BUTTON DETECTOR KERNEL MODULE: FAILED TO REGISTER\n");
        gpio_free(GPIO_NUMBER);
        free_irq(gpio_irq, NULL);
        return err;
    }

    printk(KERN_INFO "BUTTON DETECTOR KERNEL MODULE: SUCCESSFULLY INITILIASED\n");
    printk("GPIO PIN IS MAPPED TO IRQ: %d\n", gpio_irq);

    return 0;
}

static void __exit button_detector_exit(void) {
    printk(KERN_INFO "BUTTON DETECTOR KERNEL MODULE: TERMINATING\n");
    // Free the GPIO, IRQ and CHRDEV resources
    free_irq(gpio_irq, NULL);
    gpio_free(GPIO_NUMBER);
    unregister_chrdev(MAJOR_NUMBER, "gpio_irq_signal");
    printk(KERN_INFO "BUTTON DETECTOR KERNEL MODULE: TERMINATED\n");
}

module_init(button_detector_init);
module_exit(button_detector_exit);
```

5.1.3 Button_userspace.c

```
#include <unistd.h>
#include <stdio.h>
#include <stdlib.h>
#include <fcntl.h>
#include <signal.h>
#include <sys/ioctl.h>

#define SIG_RUN 20
#define INIT_WATCHDOG 0x65

int bm;
int err;

//Detects incoming signal
void signalHandler(int signal) {
    if (signal == SIG_RUN) {
        printf("BUTTON USERSPACE: RECEIVED RUN SIGNAL");
    }

    // Open the command for reading.
```

```
    FILE *script;
    script = popen("python3 WatchDog.py", "r");

    if (script == NULL) {
        printf("USERSPACE: WATCHDOG COULD NOT BE RAN\n" );
        exit(1);
    }

    err = pclose(script);

    if (err < 0){
        printf("BUTTON USERSPACE: ERROR\n");
    } else {
        printf("BUTTON USERSPACE: WAITING\n");
    }
}

int main(int argc, char *argv[]) {
    signal(SIG_RUN, signalHandler);

    bm = open("/dev/irq_signal", O_RDONLY);
    if (bm < 0) {
        perror("BUTTON USERSPACE: FAILED TO READ DRIVER\n");
        return -1;
    }

    err = ioctl(bm, INIT_WATCHDOG, NULL);
    if(err < 0) {
        perror("BUTTON USERSPACE: ERROR PROVIDING PID\n");
        close(bm);
        return -1;
    }

    // Wait for Signal
    printf("BUTTON USERSPACE: WAITING\n");
    while(1)
        sleep(1);
}
```

5.1.4 uploadAWS.py

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

import boto3
from datetime import date
import argparse
```

```
from gpiozero import Buzzer
from time import sleep

buzzer = Buzzer(26)

AWS_S3_BUCKET_NAME = [redacted]
AWS_REGION = [redacted]
AWS_ACCESS_KEY = [redacted]
AWS_SECRET_KEY = [redacted]

today = date.today()
date = today.strftime("%Y-%m")

parser = argparse.ArgumentParser()
parser.add_argument('filename', action='store', type=str, help="Name of file to upload")
parser.add_argument("-V", action="store_true", help="Use to upload video to AWS")
args = parser.parse_args()

prefix = ""

if args.V:
   fileName = args.filename + ".mp4"
   prefix = "processed/positive/"
else:
   fileName = args.filename + ".jpg"
   prefix = "upload/"

fileLocation = '/home/martin/WatchDog/www/recordings/%s/%s' % (date, fileName)

LOCAL_FILE = fileLocation
NAME_FOR_S3 = fileName

def beep():
   buzzer.on()
   sleep(0.1)
   buzzer.off()

def main():
   beep()
   print('in main method')

   s3_client = boto3.client(
      service_name='s3',
      region_name=AWS_REGION,
      aws_access_key_id=AWS_ACCESS_KEY,
```

```
    aws_secret_access_key=AWS_SECRET_KEY
  )

  response = s3_client.upload_file(LOCAL_FILE, AWS_S3_BUCKET_NAME, prefix + NAME_FOR_S3)

  print(f'upload_log_to_aws response: {response}')
  return 0

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

5.1.5 processUpload.sh

```
#!/bin/bash

file_name="$1"

echo starting s3-upload.sh
echo uploading file $1

python3 /home/martin/WatchDog/scripts/uploadAWS.py $file_name

echo completed s3-upload.sh
```

5.2 Appendix B – AWS Code

5.2.1 s3-start_step_functions.py

```
import json
import boto3

def lambda_handler(event, context):
  bucket = event['Records'][0]['s3']['bucket']['name']
  filename = event['Records'][0]['s3']['object']['key']

  sf = boto3.client('stepfunctions', region_name = 'us-east-1')
  input_dict = {'bucket': bucket, 'filename': filename}
  sf.start_execution(stateMachineArn = [redacted], input = json.dumps(input_dict))

  return {
    'statusCode': 200,
    'Info': "Step Function Started"
  }
```

5.2.2 rekognition-label_image.py

```
import json
import boto3

def get_labels(photo, bucket):
    client = boto3.client('rekognition')
    response = client.detect_labels(Image={'S3Object':{'Bucket':bucket,'Name':photo}}, MaxLabels=10)
    return response

def lambda_handler(event, context):
    bucket = event['bucket']
    filename = event['filename']
    labels = get_labels(filename, bucket)

    return {
        'statusCode': 200,
        'bucket': bucket,
        'filename': filename,
        'Labels': json.dumps(labels)
    }
```

5.2.3 rekognition-examine_labels_result.py

```
import os

def lambda_handler(event, context):
    bucket = event['bucket']
    filename = event['filename']
    human = None
    keywords = ["human", "male", "female", "adult", "child", "people"]
    labels = event["Labels"].lower()

    noPrefixFilename = os.path.basename(filename)

    for keyword in keywords:
        if keyword in labels:
            human = True
            break
        else:
            human = False
            break

    return {
            'statusCode': 200,
            'bucket': bucket,
            'filename': filename,
```

```
        'noPrefixFilename': noPrefixFilename,
        "human": human
    }
```

5.2.4 ssm-executeUploadOnPI.py

```
import json
import boto3
import time

def lambda_handler(event, context):
    bucket = event['bucket']
    filename = event['filename']
    noPrefixFilename = event['noPrefixFilename']
    human = event['human']


    ssm_client = boto3.client('ssm')
    instance_id = [redacted]
    document_name = "AWS-RunShellScript"

    noExtenstionFilename = noPrefixFilename.split('.')[0]

    # Define parameter values
    parameters = {
        "commands": [f"runuser -l martin -c 'python3 /home/martin/WatchDog/scripts/uploadAWS.py
{noExtenstionFilename} -V'"]
    }

    response_ssm = ssm_client.send_command(
        InstanceIds=[instance_id],
        DocumentName=document_name,
        Parameters=parameters
    )

    command_id = response_ssm['Command']['CommandId']
    time.sleep(1)
    ssm_commandresult = ssm_client.get_command_invocation(CommandId=command_id,
InstanceId=instance_id)
    ssm_status = ssm_commandresult['Status']

    while ssm_status != 'Success':
        time.sleep(5)
        ssm_commandresult = ssm_client.get_command_invocation(CommandId=command_id,
InstanceId=instance_id)
        ssm_status = ssm_commandresult['Status']
```

```
    return {
        'statusCode': 200,
        'bucket': bucket,
        'filename': filename,
        'noPrefixFilename': noPrefixFilename,
        'noExtenstionFilename': noExtenstionFilename,
        "human": human
    }
```

5.2.5 ses-send_alert.py

```
import boto3
import os

from email.mime.multipart import MIMEMultipart
from email.mime.text import MIMEText
from email.mime.application import MIMEApplication

def lambda_handler(event, context):
    bucket = event['bucket']
    filename = event['filename']
    noPrefixFilename = event['noPrefixFilename']
    noExtenstionFilename = event['noExtenstionFilename']
    human = event['human']

    jpgFileLocationS3 = filename
    mp4FileLocationS3 = f"processed/positive/{noExtenstionFilename}.mp4"
    tmpJpgLocation = f"/tmp/{noExtenstionFilename}.jpg"
    tmpMp4Location = f"/tmp/{noExtenstionFilename}.mp4"

    SENDER = os.environ['EMAIL_FROM']
    RECIPIENT = os.environ['EMAIL_TO']
    AWS_REGION = [redacted]

    s3 = boto3.client('s3', region_name=AWS_REGION)
    s3.download_file(bucket, jpgFileLocationS3, tmpJpgLocation)
    s3.download_file(bucket, mp4FileLocationS3, tmpMp4Location)

    SUBJECT = "WatchDog HUMAN detected"
    BODY_TEXT = ("WatchDog has detected a human.\n")
    BODY_HTML = """\
    <html>
    <body>
    <h1>WatchDog has detected a human.</h1>
    </body>
    </html>
```

```
    """
    CHARSET = "UTF-8"

    # Create a new SES resource and specify a region.
    ses = boto3.client('ses',region_name=AWS_REGION)

    # Create a multipart/mixed parent container.
    msg = MIMEMultipart('mixed')
    # Add subject, from and to lines.
    msg['Subject'] = SUBJECT
    msg['From'] = SENDER
    msg['To'] = RECIPIENT

    # Create a multipart/alternative child container.
    msg_body = MIMEMultipart('alternative')

    # Encode the text and HTML content and set the character encoding.
    textpart = MIMEText(BODY_TEXT.encode(CHARSET), 'plain', CHARSET)
    htmlpart = MIMEText(BODY_HTML.encode(CHARSET), 'html', CHARSET)

    # Add the text and HTML parts to the child container.
    msg_body.attach(textpart)
    msg_body.attach(htmlpart)

    # Define the attachment part and encode it using MIMEApplication.
    attachmentJpg = MIMEApplication(open(tmpJpgLocation, 'rb').read())
    attachmentMp4 = MIMEApplication(open(tmpMp4Location, 'rb').read())

    # Add a header to tell the email client to treat this part as an attachment,
    # and to give the attachment a name.
    attachmentJpg.add_header('Content-Disposition','attachment',filename=f"{noExtenstionFilename}.jpg")
    attachmentMp4.add_header('Content-
Disposition','attachment',filename=f"{noExtenstionFilename}.mp4")

    # Attach the multipart/alternative child container to the multipart/mixed
    # parent container.
    msg.attach(msg_body)

    # Add the attachment to the parent container.
    msg.attach(attachmentJpg)
    msg.attach(attachmentMp4)

    #Provide the contents of the email.
    ses.send_raw_email(
        Source=SENDER,
```

```
    Destinations=[RECIPIENT],
    RawMessage={'Data':msg.as_string()},
  )

  return {
        'statusCode': 200,
        'bucket': bucket,
        'filename': filename,
        'noPrefixFilename': noPrefixFilename,
        "human": human
  }
```

5.2.6 s3-move_to_processed.py

```
import json
import boto3

def lambda_handler(event, context):
  bucket = event['bucket']
  filename = event['filename']
  noPrefixFilename = event['noPrefixFilename']
  human = event['human']

  s3 = boto3.client('s3')

  if human:
    destination = f"processed/positive/{noPrefixFilename}"
  else:
    destination = f"processed/false-positive/{noPrefixFilename}"

  # Copy object
  source = {'Bucket': bucket, 'Key': filename}
  s3.copy(source, bucket, destination)

  # Delete original object
  s3.delete_object(Bucket=bucket, Key=filename)

  return {
    'statusCode': 200,
    'body': json.dumps(f'File {filename} in {bucket} moved succesfully to {destination} in {bucket}')
  }
```

5.2.7 ses-error_catcher.py

```
import boto3
import os
```

```
def lambda_handler(event, context):
    error = event["Error"]
    cause = event["Cause"]

    SENDER = os.environ['EMAIL_FROM']
    RECIPIENT = os.environ['EMAIL_TO']
    AWS_REGION = [redacted]
    SUBJECT = "WatchDog AWS Error"
    BODY_TEXT = ("There was an arror during processing in AWS.\n")
    BODY_HTML = """<html>
    <body>
      <h1>There was an arror during processing in AWS.</h1>
      <h2 style='color:red'>Error:"""+ error + """</h2>
      <h2 style='color:red'>Cause:"""+ cause + """</h2>
    </body>
    </html>"""
    CHARSET = "UTF-8"

    # Create a new SES resource and specify a region.
    client = boto3.client('ses',region_name=AWS_REGION)

    #Provide the contents of the email.
    client.send_email(Destination={'ToAddresses': [RECIPIENT]},
      Message={
        'Body': {
          'Html': {'Charset': CHARSET, 'Data': BODY_HTML},
          'Text': {'Charset': CHARSET, 'Data': BODY_TEXT,},
        },
        'Subject': {'Charset': CHARSET, 'Data': SUBJECT,},
      },
      Source=SENDER,
    )
    return {
      'statusCode': 200,
    }
```

5.2.8 stateMachine.json

```
{
  "Comment": "A description of my state machine",
  "StartAt": "rekognition-label_image",
  "States": {
    "rekognition-label_image": {
      "Type": "Task",
      "Resource": "arn:aws:states:::lambda:invoke",
      "OutputPath": "$.Payload",
```

```
  "Parameters": {
   "FunctionName": [redacted],
   "Payload.$": "$"
  },
  "Catch": [
   {
    "ErrorEquals": [
     "States.ALL"
    ],
    "Next": "ses-error_catcher"
   }
  ],
  "Next": "rekognition-examine_labels_result"
 },
 "rekognition-examine_labels_result": {
  "Type": "Task",
  "Resource": "arn:aws:states:::lambda:invoke",
  "OutputPath": "$.Payload",
  "Parameters": {
   "Payload.$": "$",
   "FunctionName": [redacted]
  },
  "Catch": [
   {
    "ErrorEquals": [
     "States.ALL"
    ],
    "Next": "ses-error_catcher"
   }
  ],
  "Next": "Choice"
 },
 "Choice": {
  "Type": "Choice",
  "Choices": [
   {
    "Variable": "$.human",
    "BooleanEquals": true,
    "Next": "ssm-executeUploadOnPI"
   }
  ],
  "Default": "s3-move_to_processed"
 },
```

```
"ssm-executeUploadOnPI": {
 "Type": "Task",
 "Resource": "arn:aws:states:::lambda:invoke",
 "OutputPath": "$.Payload",
 "Parameters": {
  "Payload.$": "$",
  "FunctionName": [redacted]
 },
 "Next": "ses-send_alert",
 "Catch": [
  {
   "ErrorEquals": [
    "States.ALL"
   ],
   "Next": "ses-error_catcher"
  }
 ]
},
"ses-send_alert": {
 "Type": "Task",
 "Resource": "arn:aws:states:::lambda:invoke",
 "OutputPath": "$.Payload",
 "Parameters": {
  "Payload.$": "$",
  "FunctionName": [redacted]
 },
 "Next": "s3-move_to_processed",
 "Catch": [
  {
   "ErrorEquals": [
    "States.ALL"
   ],
   "Next": "ses-error_catcher"
  }
 ]
},
"s3-move_to_processed": {
 "Type": "Task",
 "Resource": "arn:aws:states:::lambda:invoke",
 "OutputPath": "$.Payload",
 "Parameters": {
  "Payload.$": "$",
  "FunctionName": [redacted]
```

```
    },
    "Catch": [
     {
      "ErrorEquals": [
       "States.ALL"
      ],
      "Next": "ses-error_catcher"
     }
    ],
    "Next": "Success"
   },
   "Success": {
    "Type": "Succeed"
   },
   "ses-error_catcher": {
    "Type": "Task",
    "Resource": "arn:aws:states:::lambda:invoke",
    "OutputPath": "$.Payload",
    "Parameters": {
     "FunctionName": [redacted],
     "Payload.$": "$"
    },
    "Retry": [
     {
      "ErrorEquals": [
       "Lambda.ServiceException",
       "Lambda.AWSLambdaException",
       "Lambda.SdkClientException",
       "Lambda.TooManyRequestsException"
      ],
      "IntervalSeconds": 1,
      "MaxAttempts": 3,
      "BackoffRate": 2
     }
    ],
    "Next": "Fail"
   },
   "Fail": {
    "Type": "Fail",
    "ErrorPath": "$.Error",
    "CausePath": "$.Cause"
   }
  }
}
```