NAIT

Edmonton, Alberta

Smart Doorbell

As a submission to

Mr. Kevin Moore, Instructor

Computer Engineering Department

&

Mr. Kelly Shepherd, Instructor

English and Communications Department

Submitted by

Sibat Hassan, Student

Cindy Garcia, Student

Jonie Nicodemus, Student

CMPE2965

Computer Engineering Technology

April 19, 2022

NAIT Office of the Registrar
11762 106 Street NW
Room T409
Edmonton, AB T5G 3H1

April 12, 2022

Mr. Kevin Moore, Mr. Kelly Shepherd
Instructors
NAIT
11762 – 106 Street NW
Edmonton, AB T5G 2R1

Dear Mr. Kevin Moore and Mr. Kelly Shepherd,

We are submitting this evaluation report based on our Smart Doorbell project as a requirement of CMPE2965.

A detailed overview of design and implementation of the Smart Doorbell is discussed in this report. This report breaks down the major components of this project and the challenges encountered while building the prototype. The hardware and software technologies required to complete this project and discussed from a technical standpoint as well as the choices made in selecting these technologies are explained.

We would like to thank our instructors at NAIT for their continuous support and mentoring during this program. Their passion to teach their field of interest is a great source of inspiration for us.

Sincerely,

Sibat Hassan, Cindy Garcia & Jonie Nicodemus
CNT Student

Table of Contents

List of Figures and Tables

Abstract

The Smart Doorbell project was an opportunity to explore remote monitoring and controlling of devices. The Doorbell hardware was interfaced with the Raspberry Pi and the Pi was connected to a Mobile App for remote monitoring and access control. The server script was written in Python to stream video and audio to the Mobile App as well as receive audio from the App. The sensor connected to the Pi was monitored for any motion and notification was sent to the App if any motion was detected. The script also allowed the App to control the door lock.

The expected outcomes of the project were verified by transmitting video and audio from the Pi and successfully receiving both video and audio on the Mobile App. The full duplex audio communication was established between the Pi and the Mobile App. And it was verified by initiating a simultaneous two-way audio communication between the Pi and the App. The door lock operation was tested by remotely controlling it from the App. The motion sensor was tested by triggering it and the notifications were received on the App. Overall, the Smart Doorbell was working without any delays and the entire system was very responsive.

1.0 Introduction

The remote monitoring and controlling of devices are common applications of computer and network engineering. The devices connected across the internet can be used to exchange data over the network and make decisions based on the data.

This report includes a detailed roadmap of the Smart Doorbell project. It discusses the different phases of project, the challenges encountered and the decisions made to overcome these challenges. The report starts with an overview of the project. Then the Doorbell Hardware is discussed from a design and implementation standpoint. After that, the Raspberry Pi server is explained as a central point of communication between the Doorbell and the Mobile App. Then the Mobile App development phase is discussed along with the challenges faced while developing the App. Some enhancements are suggested to improve this project. And to conclude, the project results are discussed by highlighting the system limitations and the desired outcomes.

The objective of Smart Doorbell was to allow remote monitoring and access control. The network communication was limited to a local network to focus on the client-server application development for video and audio streaming. The purpose of the project was to gain a deeper understanding of the real time data communication across the network for monitoring and controlling interconnected devices.

2.0 Smart Doorbell Overview

The Smart Doorbell project consisted of three different parts. The first part was related to the hardware components of the doorbell, the second part was the Raspberry Pi server and the third part was the Mobile App used to monitor and control the doorbell remotely.

The hardware part was designed to detect motion and notify the Mobile App user and receive control signal from the Mobile App to lock/unlock the door. A button was also added to ring the doorbell, pressing the doorbell would turn on the buzzer. An LCD was incorporated to display text messages to the user.

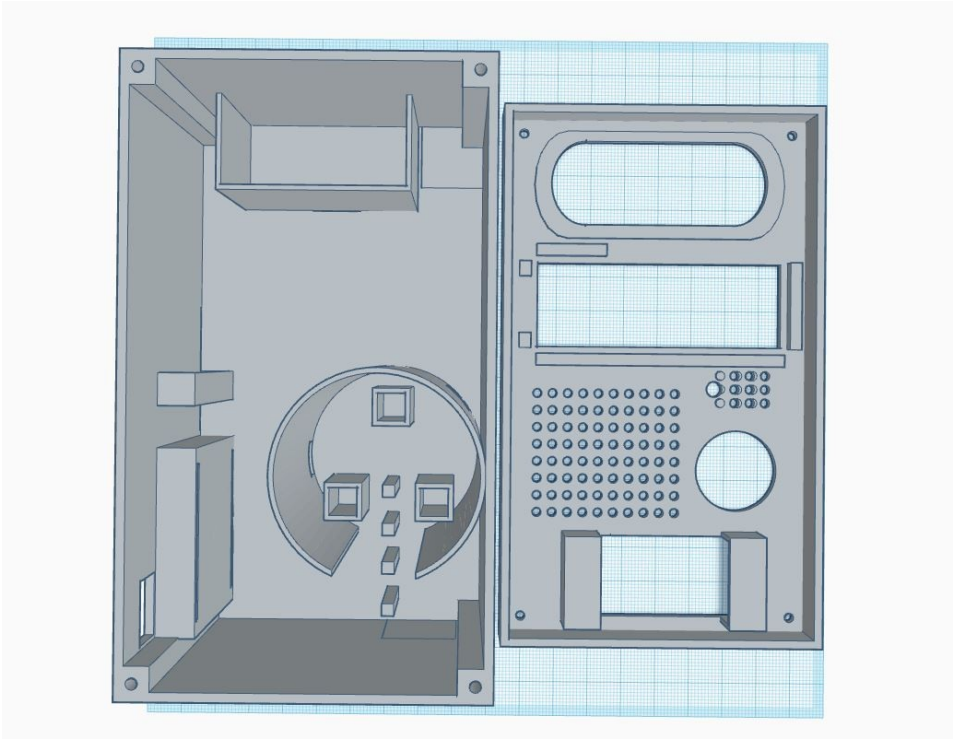
The Raspberry Pi server part was designed as a central point of communication between the Doorbell hardware and the Mobile App. The Pi would live stream video and audio to the Mobile App as well as receive audio from the App. Upon detecting motion, a notification would be sent to the App. The App could in return lock or unlock the door by controlling the solenoid interfaced with the Pi.

The Mobile App part was designed to monitor the video and audio feed from the Pi as well as control the door lock. The App could also stream audio to the Pi. Any detected motion would trigger the notification send event and the App could be used for remote monitoring and access control.

3.0 Hardware Design

The hardware components used of the doorbell are as following: A camera with built-in microphone, Bluetooth speaker, 16 x 2 LCD Screen, PIR motion detection sensor, 5VDC buzzer, and a solenoid circuit for turning on/off the of the solenoid.

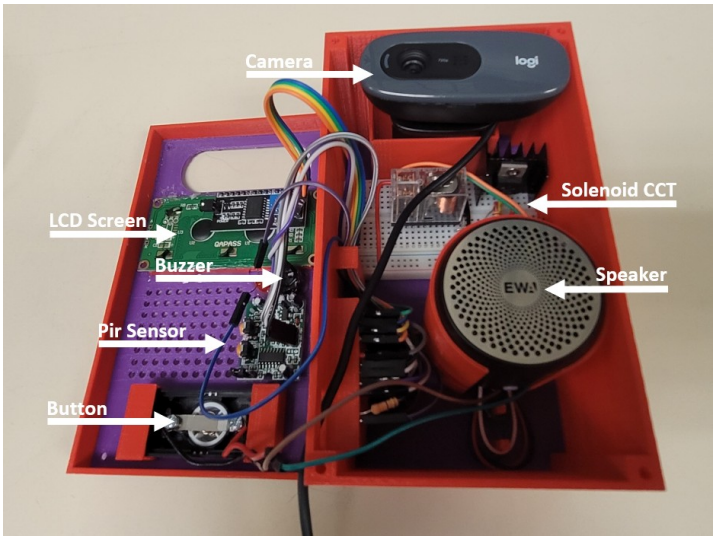
Figure 1. Doorbell Case



The housing made for the Smart Doorbell was 3D-printed. As shown in the above figure, the Doorbell case was meant to encapsulate all the doorbell's module inside it. Protrusions were added such that when installing the modules, they would align better and sit evenly when installing them. Holes were also added on the left side and underneath the case to minimize the cable clutter when attaching the connectors of each module into the breadboard. When

assembled, the box with lid design features a dimension of 154 x 91 x 67 mm with a minimum 2 mm of thickness to the walls.

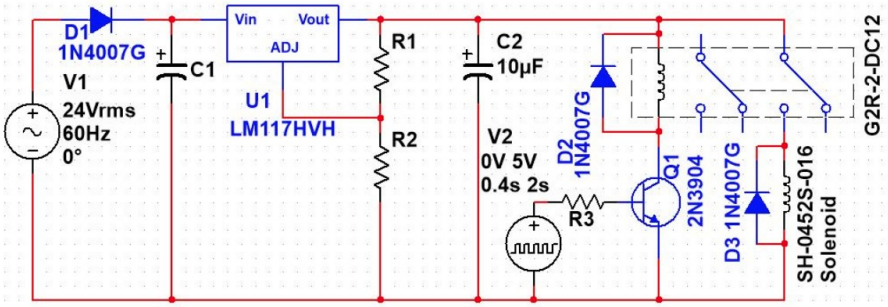
Figure 2. Doorbell Assembly



Detecting motion was achieved by using a PIR sensor. The sensor was fitted inside the Doorbell case as showed above. As explained by Ada & DiCola (2022), this module has two built-in slots which are sensitive to infrared. When the first sensor detects a warm body, it causes a positive differential change between the two halves. When the warm body leaves the sensing area, a negative differential change is then generated which produces the other half of the pulse that detects motion (“The PIR Sensor” and “Lenses” sections).

A LCD was used to display instruction to the user. The LCD was interfaced with the Pi by using the guide by Campbell (n.d.). The doorbell utilized the SDA and SCL pins of the Raspberry Pi to send and receive data from the LCD. With those two pins along with the screen connected to 5V VCC and ground, the Raspberry Pi was able to display/delete texts by using the I2C library (“INSTALLING THE LIBRARY” section).

Figure 3. Solenoid Control Circuit



Source: Taylor, 2021.

The door lock was controlled by using the solenoid circuit borrowed from the CNT Practical Electronics course. As explained by Taylor (2021), the voltage required for the solenoid to operate is 12VDC. Using the LM317 regulator, the 24VAC power supply used to power the circuit below is then converted to match the operating voltage to prevent damaging the solenoid. When the relay contacts are closed, the regulator will reach a temperature close to 60°C after 10 seconds ("Introduction" and "Safety Warnings" sections). To dissipate the heat faster, a heatsink coupled with thermal paste was added to the back metal portion of the regulator. The solenoid circuit input was controlled by the Pi by using a simple command in Python.

4.0 Raspberry Pi Server

The Pi acts as the central point for the communication between the Mobile App and the Doorbell. It relies on the network sockets for the transmission and reception of data over the Wi-Fi network. The hardware interfaced with the Pi is controller by running a script written in Python. The Python script is also handling the live video and audio stream. So, the code written in Python is the main processing unit of this project.

4.1 Raspberry Pi Configuration

Raspberry Pi uses a lightweight Linux based operating system known as Raspbian. In terms of hardware specifications, Pi is not a high-end system. Therefore, Raspbian comes with a minimum set of pre-installed packages to maintain high performance. But video and audio streaming meant that some additional packages had to be installed.

Since Raspberry Pi 400 does not come with the camera module port, a USB web camera had to be interfaced with it. The fswebcam package is one of the utilities intended for USB web camera interfacing with the Pi. The fswebcam package installation and configuration instruction provide by Jolles (2021) enabled the Pi to successfully interface with the USB web camera ("Setting up and using a USB webcam" section). VLC media player video capture mode was used to test the video captured by the USB web camera.

The USB web camera interfaced with the Pi includes a built-in microphone, but it was not recognized by the Pi. Some configuration was needed to make it work. The configuration steps by Emmet (2020) made it possible to configure and test the microphone ("Configuring Alsa on your Raspberry Pi for your Microphone" section).

The Raspberry Pi 400 also lacks the 3.5mm audio jack and by default there was no package installed for managing Bluetooth devices. Using a Bluetooth speaker with the Pi required a couple of packages. The set of instructions provided by Emmet (2019) were used for the interfacing of Bluetooth speaker with the Pi ("Setting up Bluetooth on the Raspberry Pi" section). The speaker audio was tested by playing a .mp3 file.

Initially, interfacing the Pi with the web camera, microphone and speaker were enough to move forward with the video and audio streaming requirement of the project.

4.2 Technologies Explored For Video And Audio Streaming

While trying to figure out a reliable way to stream video and audio in real time, different packages were explored as a proof of concept. The Pi was able to successfully stream video and audio over the internet and a simple web page was designed to view live feed from the Pi.

4.2.1 Motion

Motion is one of the packages available on Raspbian that can capture video and stream it on a network port. By following the user guide on the Raspberry Valley website (*Streaming Video with Motion*, n.d.), Motion was configured to stream video on one of the network ports ("Setup streaming" section). The video streamed by the Pi was successfully received by a computer on the same network.

4.2.2 NGROK

The next step was to stream video over the internet and receive it on a different device from an external network. Exposing the Pi's IP address to any external network required port forwarding, but it required network administrative rights. Therefore, the decision was made to use NGROK instead. NGROK is a tunnelling software that exposes local server ports to the Internet by routing the data through one of its servers. NGROK online documents (*Getting Started with ngrok*, n.d.) were used to enable video streaming over the internet ("Getting

Started with ngrok." Step 1 to Step 4). A cellphone connected to an external network was used to receive video streamed by the Pi.

4.2.3 Video Streaming Website

A simple website was hosted on NAIT's Thor server to test the video streamed by the Pi. The video stream was very slow and there was a significant lag as the video was being routed through an external NGROK server. Based on the video lag, the decision was made to restrict further development over a local network.

4.2.4 WebRTC

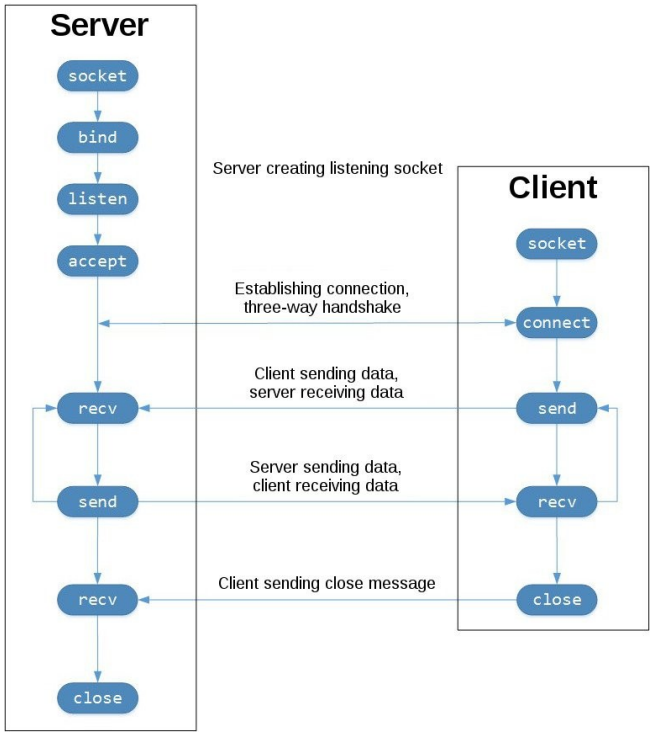
After testing video streaming, it was time to start exploring audio streaming form the Pi. Instead of streaming audio only, WebRTC was used to stream both video and audio simultaneously. WebRTC provides a web user interface to control stream form a server. The instructions by Anantharam (2021) were used to configure Pi for video and audio streaming ("Live Streaming Video + Audio on Raspberry Pi using USB Camera and Microphone", Step 1 to Step 3). A computer over the same network was used to control video and audio stream from the Pi. The Pi was able to stream both video and audio at the same time without any lags.

4.3 Python Socket Programming

After testing video and audio streaming, the decision was made to use a Python socket module for communication between the Pi and the Mobile App. Sockets allow any client-server

application to exchange data over the internet. And writing Python code for client-server application required a fundamental understanding of Transmission Control Protocol (TCP).

Figure 4. TCP Sockets



Source: Jennings, 2022.

The image above shows the workflow for the TCP based socket connection. As explained by Jennings (2022), the server socket is setup to listen for the client connection. The client socket can start the connection with the server by initiating a three-way handshake. During a three-way handshake, the client requests the server for a connection, the listening server responds to the client request by accepting it and finally the client acknowledges the server response. After a successful connection is established, the client and server can exchange data over the network. Once done, both the client and server can close their respective sockets to end the communication (“TCP Sockets” section).

The introductory socket programming guide written by Jennings (2022) was used to code a simple client-server application for data communication between the Pi and a Computer connected over the same network. This application served as a starting point for the Pi and the Mobile App communication interface part of the Smart Doorbell project.

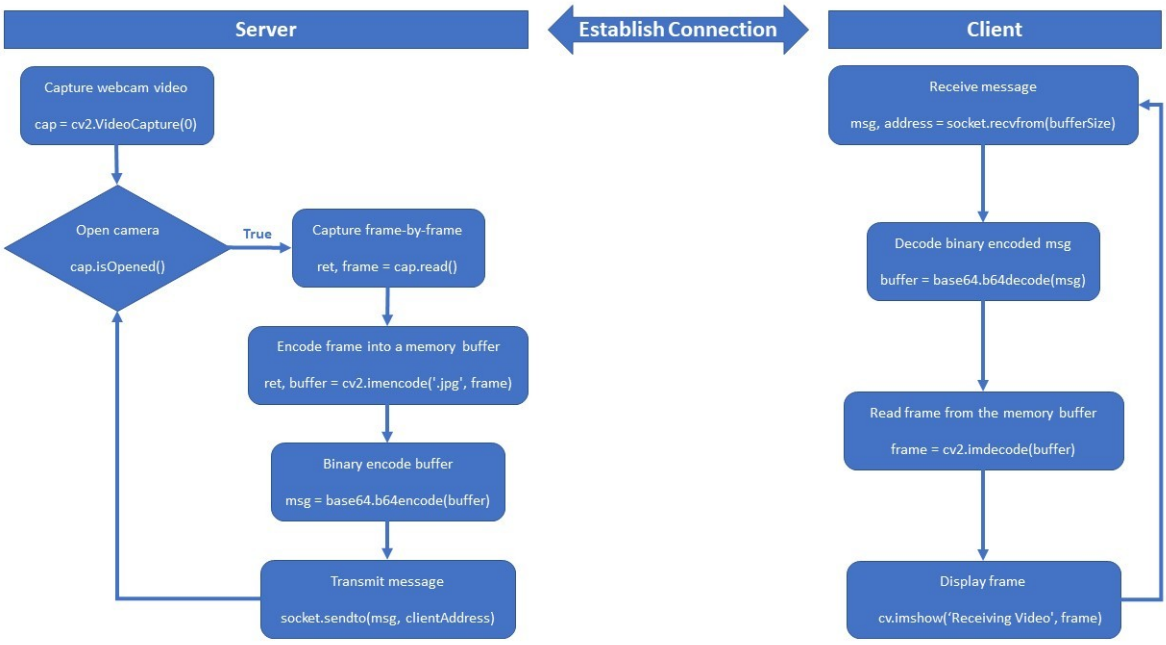
4.4 Video And Audio Communication

OpenCV and PyAudio are the main libraries used for video and audio streaming. Both libraries are open source and well documented. Although both libraries are packed full of video and audio processing functionality, only a tiny fraction of functions were used for this project.

4.4.1 OpenCV

OpenCV is a computer vision library that includes several algorithms for image processing. But for this project only the webcam video capture functionality of OpenCV was used.

Figure 5. Video Stream



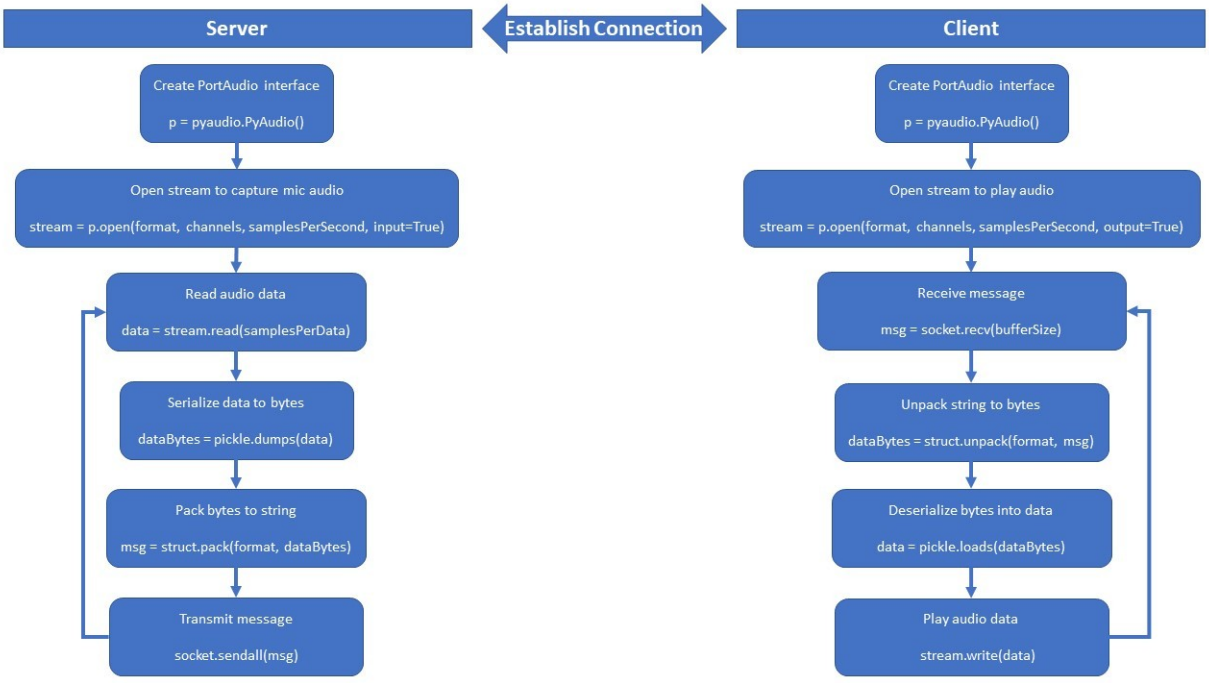
The above workflow describes the implementation details of video streaming code written for the Pi. As discussed in the OpenCV tutorial (*Getting Started with Videos*, n.d.), the two main function used for video capture and display are 'read()' and 'imshow()' ("Capture Video from Camera" section). After a successful connection is established between the server and the client, the video streaming can start over the connected sockets. To transmit video, the first step is to create the video capture object and then start the video transmission if the camera is open. Video is transmitted one frame at a time by reading a single frame, storing it in a memory buffer, converting the buffer into bytes and then sending it to the receiver. To receive the video, the same steps are carried out in reverse order. First receive the bytes, convert received bytes to memory buffer, load frame from the buffer and then show the frame.

The frame rate is dependent on the network speed and other factors like server processing speed and webcam maximum frame rate. But for the Pi, a consistent 20 fps was achieved over the same network.

4.4.2 PyAudio

PyAudio is a cross-platform audio processing library that comes with all sorts of audio signal analysis functionality. But for this project, it was only used to capture microphone audio and to play audio through a speaker.

Figure 6. Audio Stream



The above workflow details the algorithm used for audio streaming. The tutorial by Langen (n.d.) was used for capturing and playing audio. The audio communication starts after the client-server connection is established. To transmit audio, an interface to the audio port is created and then an audio stream is created for capturing the microphone audio. The audio data is captured as

samples, it is then serialized and packed to bytes of string. Once ready, the audio data is transmitted. The receiver also starts by creating an interface to the audio port, then an audio stream is created for playing the audio. The data is received and unpacked for deserialization. After deserializing the data, the audio is played through a speaker.

Simultaneous two-way audio communication was achieved for this project. Audio transmitting and receiving algorithms were coded on both the Pi and the Mobile App.

4.5 Multi-Threading

Because the Pi lacks the processing power to handle multiple scripts efficiently, there was a significant lag in the audio transmission and reception. Also, the video frame rate was as low a seven frames per second. The decision was made to write a single server script and run each process in a separate thread to increase the performance of video and audio streaming.

4.5.1 Video Frame Rate

The Pi struggled to maintain a frame rate of even 10 fps, when other scripts were also running. Threading was an obvious solution to this problem. The video transmitting part of the code was rewritten to run the frame capturing and transmitting process in a separate thread. The resulting frame rate was a significant improvement over the non-threaded code. By using threading, a consistent 20 fps was achieved even though the Pi was also running other processes at the same time.

4.5.2 Audio

The audio lag was as much as 3 seconds, mainly because of the low processing power of the Pi. The server script was refactored to include separate threads for both audio transmission and reception. The audio processing threads were designed to continuously run the audio transmit and receive process separate from each other. The lag was completely removed by using threading and the audio streaming was consistently lag free.

4.6 Two-way Data Communication

As part of the interface between the Mobile App and the Doorbell, the Pi was responsible for controlling the door lock and sending App notifications. A two-way data communication was established between the Pi and the Mobile App for this purpose. Same network socket was used for controlling the door lock and sending notifications.

4.6.1 Controlling The Door Lock

Since the door lock could be locked and unlocked any time by the Mobile App, the server script was written such that the incoming data from the Mobile App was continuously received. After receiving the data, the message was checked against the predefined control values. Based on the message value the door was either locked or unlocked. The message was continuously received from the Mobile App in a separate thread every second. The door lock was very responsive and there was no delay in locking and unlocking the door.

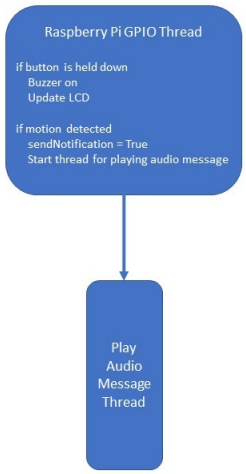
4.6.2 Mobile Notifications

Whenever the motion was detected, the Mobile App was sent a notification by the Pi. The Pi general purpose input and output pins were continuously monitored for any signal from the motion sensor. And as soon as any motion was detected, a predefined notification message was transmitted to the Mobile App. The notification transmission process was run in a separate thread to avoid any delays and the sensor output was checked every second. The Mobile App was able to receive the notifications without any delay.

4.7 Hardware Interfacing

The Doorbell hardware was interfaced with the Pi by using general purpose input and output (GPIO) interface of the Pi. The GPIO controlling code was written in Python and the entire process was executed in a separate thread. The thread would continuously poll the GPIO pins for any input change and actively control the Doorbell.

Figure 7. Doorbell Hardware Interface



The illustration above shows the algorithmic steps coded in Python. The GPIO thread checks for the predefined conditions. Based on the input pins state, the Pi activates the buzzer and sends notifications. If the Doorbell button is pressed the Pi activates the buzzer and updates the LCD message, informing the user that the Doorbell is ringing. If the motion is detected, the Pi sends a notification message and starts a new thread for playing the audio message. The audio message is played in a separate thread to avoid freezing the GPIO thread while the audio message is being played.

By checking the GPIO pins in a separate thread, the Doorbell was very responsive and there were no delays in any of the operations performed by the Doorbell.

5.0 Mobile App

A Mobile App was built to monitor the Doorbell surroundings and control the door lock. Different technologies were explored to build the Mobile App. While each technology presented its own set of challenges, ultimately a Mobile App was successfully developed for this project.

5.1 Xamarin

To build the Mobile App, there were various environments to choose for the project. Initially, Microsoft's Xamarin.Forms UI framework to develop the app came first. According to Microsoft (*What is Xamarin.Forms?,* 2021), "Xamarin Forms allows developers to create user interfaces in XAML with code-behind in C#". Prior experience in C# made this environment a reasonable choice once the project was underway. Setting up Visual Studio 2019 to work with

Xamarin.Forms only required an additional workload installed (*Installing Xamarin in Visual Studio 2019*, 2021). Once that was complete, development of the app began.

Issues arose when researching how to build the app interface. Various tutorials online were either outdated or were referencing similarly named frameworks that used a different code structure not applicable to Xamarin.Forms. This made going forward with the development difficult, and ultimately a switch to a different environment for the app was done, in order to not to fall behind in the project development timeline.

5.2 Kotlin

The next choice for development of the Mobile App was to use Android Studio. Since cross-platform development was never a major consideration, only developing for one mobile operating system was not a concern. As the official development environment for the Android operating system, resources were plentiful compared to Xamarin.Forms, and more up to date.

Android Studio utilizes the Kotlin programming language, so the basics of the language was learned, as provided through Google's Android Developers Kotlin section (*Learn the Kotlin programming language*, 2020). After going through these tutorials, the app reached the point of simulation via Android Studio's built-in virtual Android device emulator. Functionality such as receiving notifications within the 'phone', and communicating to a basic server written in Python to simulate the door locking mechanism reached functional status, again following Google's official documentation for the former (*Create a Notification*, 2022).

When it came to the point of getting ready to set up the video and audio streaming, an issue arose. Trying to reconcile the Python server code and the Android Studio side proved difficult and time consuming. Although both ends were using TCP to communicate with each other, the Python server part was using a library, known as PyAudio, for the audio communication aspect; it was discovered that Android Studio, and therefore Android development as a whole, did not support this library as of the time of this written report. Due to time constraints, a switch to another environment that would work better with the server part was required.

5.3 Kivy

Since the project deadline was approaching and the Mobile App part of the project was still incomplete, the decision was made to use one of the Python packages for Mobile App development. The main reason for using a Python based package was that the client scripts were already written in Python to test video and audio streaming form the Pi. Kivy was chosen for developing the App since it is one of the major Python Mobile App development packages.

5.3.1 App Development

A virtual environment was setup on a computer for App development. Kivy package was installed on the virtual environment along with all of the dependencies. After setting up the development environment, the UI was designed by following the tutorial by Driscoll (n.d.). Widgets were added for displaying video, muting/unmuting audio and locking/unlocking the door. Events were added for audio and door lock buttons control (“Creating a Kivy Application” section).

Since the video and audio were being received continuously, the UI was freezing and the widgets were not very responsive. To stop the UI from freezing, background threads were used for video and audio reception. A separate thread was also used for transmitting audio from the App. The threading structure used for the App was very similar to the one used for the server script. The Python scripts written for testing the video and audio transmission from the Pi were used for the App.

5.3.2 App Deployment

Once the development of the Mobile App was complete, it was time to package the App for Android devices. The App packaging environment was setup in Ubuntu as explained by Driscoll (n.d.). To test the App packaging environment, a sample calculator code was deployed on Android and transferred to a mobile device. It worked without any issues. But while packaging the Doorbell App, the environment failed. After some trouble shooting it was discovered that the audio library (PyAudio) used for the App was not supported by the Android platform. An attempt was made to deploy the App on the iOS platform. But soon it was discovered that while PyAudio was supported by iOS, a paid subscription-based membership was required by Apple to package Apps for iOS devices ("Packaging Your App for Android" and "Packaging Your App for iOS" sections). Therefore, the idea of deploying the App on iOS was abandoned.

So, the Mobile App was fully functional in a virtual environment and the only way to deploy it on an Android device was to change the audio library. But with the project deadline approaching, it was decided to focus on the interfacing part of the project and leave the App running in the virtual environment.

6.0 Further Enhancements

While the basic requirements of the project were completed, the proposed optional features were not added. This project can be enhanced in a few ways discussed below.

1. Voice command feature can be added to activate the doorbell instead of pushing the button. A Speech Recognition Package can be used with the Python server script to enable voice commands recognition.

2. Video and audio message recording feature can be added to the Pi. If the Mobile App user does not respond to the notification, the Pi can record messages on behalf of the user. The libraries used for video and audio processing (OpenCV and PyAudio) include the required functionality for recording video and audio.

3. The OpenCV can be used for face detection to ensure that the motion sensor is triggered by a person. False detection and notifications can be drastically reduced by this approach.

4. Internet device can be configured to make the IP address of the Pi accessible from anywhere across the internet.

5. A different audio package can be used for the Mobile App for deploying it on the Android platform. So that the App can be run on a mobile device instead of the virtual environment.

The above suggestions are not exhaustive, as there are other enhancements that can be explored. As an example, the OpenCV also supports face recognition functionality. The doorbell can be trained to recognize certain people for automated access to the premises.

7.0 Project Results

The initial attempts to stream video and audio at a decent rate were not encouraging but after some optimization, the video frame rate and audio sampling rate were improved.

1. For video, a 20-fps frame rate was achieved at a resolution of 864 x 480.

2. For audio, a 44100-fs sampling rate was achieved without any lag.

The solenoid circuit was exhibiting overheating, when left on for extended periods of time. By adding heatsink with thermal paste, the overheating was reduced but not eliminated. But the door lock remained responsive even with the reduced overheating issue.

The motion sensor was extremely sensitive and was triggering excessively even to the slightest of motion. A delay was injected in the server script to turn off the sensor while the audio message was playing. But even with the delay, the sensitivity of the sensor remained an unresolved issue.

Because the speaker and microphone were housed within the same doorbell case, there was audio feedback. As the doorbell case was not designed to isolation the speaker and the microphone from each other, the audio feedback remained a consistent problem.

The audio library (PyAudio) was not supported by Android. Therefore, the Mobile App was not able to run on a mobile device. The issue was traced back to the PortAudio interface used by the PyAudio. The interface library was unable to compile in the App packaging environment.

8.0 Conclusion

The Smart Doorbell can be used to monitor any premises remotely. It can also be used as a remote access control system. By using this system, the activity around the access point of any premises can be tracked.

This report covers the development of this project from start to finish as well as the challenges encountered along the way and the decisions made to overcome these challenges. After a brief overview of the system, the hardware design and implementation aspect of this project is discussed in detail. The motion sensing and the solenoid door lock design are explained. Then the Raspberry Pi server is covered as an interface between the Doorbell and the Mobile App. The video and audio streaming processes are explained along with the algorithms used. Finally, the Mobile App development phase of the project is explained and the challenges related to it are also discussed.

The key aspect of this project is the high volume of data communication taking place among different device in real time. The algorithms used for streaming video and audio can be used in any system for communication. The devices connected to a network for data communication demonstrates the skills required for developing any client-server application. Interconnecting Doorbell hardware, Pi server and Mobile device demonstrated that by designing a proper

interface, different devices can communicate with each other. The concepts applied to this project can be used for connecting any hardware device to a network and remotely controlling it.

References

Ada, L. & DiCola, T. (2022, April 19). *How PIRs Work*. Adafruit. https://learn.adafruit.com/pir-passive-infrared-proximity-motion-sensor/how-pirs-work

Anantharam, P. (2021, February 16). Live Streaming Video + Audio on Raspberry Pi using USB Camera and Microphone. Medium. https://pramod-atre.medium.com/live-streaming-video-audio-on-raspberry-pi-using-usb-camera-and-microphone-d19ece13eff0

Campbell, S. (n.d.). How To Setup An I2C LCD On The Raspberry Pi. Circuit Basics. https://www.circuitbasics.com/raspberry-pi-i2c-lcd-set-up-and-programming/

Create a Notification. (2022, April 7). Android Developers. https://developer.android.com/training/notify-user/build-notification

Driscoll, M. (n.d.). Build a Mobile Application With the Kivy Python Framework. Real Python. https://realpython.com/mobile-app-kivy-python/

Emmet. (2019, December 15). Bluetooth on the Raspberry Pi. Pi My Life Up. https://pimylifeup.com/raspberry-pi-bluetooth/

Emmet. (2020, April 15). Using a Microphone with a Raspberry Pi. Pi My Life Up. https://pimylifeup.com/raspberrypi-microphone/

Getting Started with ngrok. (n.d.). ngrok Docs. https://ngrok.com/docs/getting-started

Getting Started with Videos. (n.d.). OpenCV-Python Tutorials. https://docs.opencv.org/4.x/dd/d43/tutorial_py_video_display.html

Installing Xamarin in Visual Studio 2019. (2021, July 8). https://docs.microsoft.com/en-us/xamarin/get-started/installation/windows

Jennings, N. (2022, February 21). Socket Programming in Python (Guide). Real Python. https://realpython.com/python-sockets/

Jolles, J. (2021). Working with USB webcams on your Raspberry Pi. The Raspberry Pi Guide for scientists and anyone else! https://raspberrypi-guide.github.io/electronics/using-usb-webcams

Langen, J. D. (n.d.). Playing and Recording Sound in Python. Real Python. https://realpython.com/playing-and-recording-sound-python

Learn the Kotlin programming language. (2020, September 8). Android Developers. https://developer.android.com/kotlin/learn

Streaming Video with Motion. (n.d.). Raspberry Valley. https://raspberry-valley.azurewebsites.net/Streaming-Video-with-Motion/

Taylor, R. (2021, October 11). Project 06: Relay Control of a Solenoid Actuator [Moodle notes]. Northern Alberta Institute of Technology CMPE2150 Practical Electronics https://moodle.nait.ca/pluginfile.php/13326451/mod_resource/content/1/Project%2006%20Relay%20Control%20of%20a%20Solenoid%20Actuator.pdf

What is Xamarin.Forms? (2021, July 8). Microsoft Docs. https://docs.microsoft.com/en-us/xamarin/get-started/what-is-xamarin-forms