Jack Quarm, jack.quarm@colostate.edu

Declan McLaughlin, declan.mclaughlin@colostate.edu

Denzel Hartshorn, denzel.hartshorn@colostate.edu

Raspberry Pi Term Project Report

Introduction:

Our Raspberry Pi Project is a culmination of many of the main concepts we have learned this semester in CS370 as well as the implementation of some of the networking concepts we learned in CS250. Our project is designed to use 3 different devices, one Raspberry Pi, one laptop, and one camera, all working together to have a fully functional security camera that can recognize faces and send images of the face that was detected to another device and to be posted onto the social media platform Twitter/X. We will get more into the details of how we we're able to implement this using the Raspberry Pi later in the report.

Problem Characterization:

The problem we were seeking to solve with this project was that many security camera companies like Ring are expensive to buy initially and often require subscriptions to run. We wanted to create something using the Raspberry Pi that cut out the middleman of a paid service and used a social media platform like Twitter/X to store our data and notify us. Many of these security camera services have a motion detection sensor that sends a message to the user with an image of what was detected. In this project, we use a webcam and some prebuilt OpenCV modules to detect a face and start recording. Once the camera stops detecting a face, it then sends

a frame of that video from the laptop to the Pi where it is then posted to a private Twitter account where we can get the notifications. We had many issues when attempting to implement this idea using our Raspberry Pi, firstly with the webcam but mainly with sending the data back and forth between the Pi and the laptop. In addition to this, we had plenty of issues initially with connecting our laptop/the CS machines to the Twitter API. All of these different problems we've written about are just a few of the technical errors we experienced throughout this process. We spent plenty of time troubleshooting and tweaking the programs to slowly inch closer and closer to our main and final goal. It was beneficial for us to be working in a group of three because we were able to divy up responsibilities amongst eachother, allowing us to get more done in a shorter amount of time. From our experience, this was the most efficient way we found to be able to take on such a large project.

Proposed Solution and Implementation Strategy:

We were able to accomplish this task by breaking up a large project into many smaller parts and python program files, all in hopes of reducing redundancy, increasing modularity, and keeping our implementation more readable, traceable, and manageable. Our first task was to get the Twitter bot up and running. Initially, we tried to create the Twitter bot using Selenium, but there were many different issues with the different browser drivers, the xpath parsing, and giving input into the login screen. Eventually we scrapped the selenium idea and went to straight to Twitter's developer webpage. There, we followed a YouTube tutorial[1] and we were able to create a free dev account and get access to the API tokens we needed. We created a file called *"oauth2.py"* that would allow us to set up and use all of the correct API credentials using the new oAuth2.0 endpoint that Twitter has so that we could connect our Python program to Twitter

itself. We used a library called *"tweepy"* that had specific functions within the library that allowed to easily setup our API client. There are several constant variables declared in this class that the user would have to manually update before running the program, these variables are the access tokens that change based on what Twitter account you plan to log in to. After some tinkering around with tweepy, we successfully created some functions in our *"twitter_bot.py"* class that allowed us to both send a tweet with text, and a tweet with text and media. Now that we had gotten the API working with our program, we decided to take a step back and work on implementing the camera's functionality.

After creating the *"webcam.py"* file and following a few OpenCV tutorials on YouTube[2], we were able to successfully get the camera working. The first working model was in a method called *"get_vid()"* and it simply turned on the camera, and used a prebuilt openCV model to detect a generic face or body. Once detected, a timestamp was noted and the program started writing to a .mp4 file with that time. While being detected, the program would draw a box around the subject's face and display it in the "Camera" window that was open, and after a set amount of time after detection ended (we found two seconds to be reasonable), the camera would turn off and the video file would be saved in the *./videos* folder that was created. The next plan of action was to extrapolate a single frame from this video since the free version of Twitter's API only allowed for images to be posted and not videos. So, a method called *"get_img(filename)"* was created. The filename is the name of the video that was just recorded, this method would be called after detection stopped and the video file was not being written on anymore in *get_vid()*. This method essentially had the same detection methods as *get_vid()* where a box is drawn around a face as it goes frame by frame. It creates a folder named *./frames*, that store each individual frame of the video. After each frame from was stored in the *./frames* folder, we take

the current time and send a formatted string and the total number of frames minus the number of frames after detection stopped divided by two to the twitter bot to post: *((current_frame - remove_frame)/2)*. We are sending the frame in the middle to ensure a good view of the person being detected. Once that returns, *get_vid()* calls a method named *"deleteAll()"* which uses the *shutil* library to recursively delete every frame and video from the *./video* and *./frames* folders, as well as the folders to keep the file directory nice and tidy. After creating a simple *"Main.py"* file and making a basic program execution loop, the first big part of this project was complete. We had the ability to determine a max number of posts, and time in between scans to then call our *webcam.get_vid()* method and run the loop until completion. After getting this program running, we thought it would be a good idea to see how it would run on the Pi, since we planned on having the webcam running off of it. What should have been a simple five minutes of setting up the pi, running the program, and calling it a day, turned into several hours of trying to figure out why the webcam could not be detected. After reflashing the Pi's OS and downloading every possible webcam module known to man, we finally got the webcam to turn on. Then we hit another wall, our imports were not working. For whatever reason the Pi really did not like the cv2 module imported from openCV, even though it was installed and up to date. This gave us an interesting dilemna, how were we going to make this work now? Already used to pivoting our project at this point, we quickly came up with the only solution that seemed to make sense: the Pi would become the facilitator and poster, while the laptop took care of the main program loop using the webcam. Now that we had a plan of action, it was time to tackle what would be the hardest part of the project: networking.

We started off by creating two files: *"laptop.py"* and *"pi.py"*. The first goal was to be able to have the Pi send over two variables to the laptop: *"max_posts"* and

"interval_between_scans". These two variables would be essential to the entire program loop as they determined how it was going to execute. After skimming forums and watching a some tutorials on YouTube[3], we figured it would be easiest to send data from a client, and receive data as a server. So, the two devices end up doing a bit of a networking tango so to speak. Alternating between the Pi and laptop being the server or the client. The laptop starts as the server and the Pi connects to it using sockets on a predetermined port, and already knowing the laptop's IP address. Once connected to each other, the Pi proceeds to send the two variables one at a time, 2048 bytes at a time, until it receives an acknowledgement from the laptop. Once this is complete, the Pi disconnects from the laptop's server socket and initializes one of it's own. This part went fairly smoothly, being that the internet had lots of resources detailing how to set up and send a string over a TCP connection. Now is where we started having trouble. The laptop would execute the webcam loop with no issue, but now we had to rewrite part of the webcam code, since it was posting to Twitter straight from the *webcam.py* file. Instead of overwriting what we had already created, we made two new methods: *"get_vid_net()"* and *"get_img_net(filename)"*. The original methods were left so that the program could be run locally on one machine, and these new ones were slightly tweaked to work in the context of this project. *get_vid_net()* would be called from laptop.py's *main()* method and would return the frame that *get_img_net(filename)* returns. That way, the laptop could send the image back over to the Pi using *"send_image(filename)"*. Now, the laptop would connect to the Pi's server socket and start sending the image chunk data over in chunks of 1024 bytes. The Pi would receive this in *"receive_image()"* and write the chunk data to a file named *"image_to_tweet.jpg"*. This *.jpg* file would stay on the Pi and get overwritten each time a new photo was received, thus keeping it's file directory clean and tidy as well. Once the image was received, the Pi would call

send_tweet(filename) which would get the current date and time, and then call *twitter_bot.tweet_text_and_media(text, media)* to post to Twitter. To our great surprise it almost worked first try. However, it was posting about half of the image, always the top half. After some digging around through the code, we figured that the Pi was simply just deciding it had received enough of the image to post and was uploading an unfinished file. It was quite interesting to see how the image was being written in bytes from top to bottom, but we could not leave it as unfinished as it currently was. A quick and easy solution to this issue was to send over the file size from the laptop to the Pi first, that way the Pi would know when it has received all of the image chunk data. Sure enough, receiving the data inside of a simple *"while received_data < file_size:"* did the trick and the images were being fully sent over the TCP connection to the Pi before posting the tweet. After this accomplishment, the main objective of the program was done, and all that was really left was to format the terminal's output so as to make sense of what is happening in the program as it executes. With many formatted print statements containing IP addresses and sockets, variable names and sizes, and general information about where the program was at, both the laptop and Pi would now be able to understand what was happening on the other end.

With that, this project was essentially complete, but we wanted to add a few final touches in order to be able to share this with our CS friends and really go above and beyond to make this something we are extra proud of. So, we combed through every line of code, working to reduce redundancy, better name our methods and variables, and commented like madmen throughout every file to help those from the outside understand how everything works. We needed a profile picture and banner for the Twitter page, so we photoshopped Professor Pallickara onto a goat, because it's pretty obvious he is the G.O.A.T. We created a requirements.txt file using

"pip freeze > requirements.txt" and temporarily got rid of any sensitive API keys. After maticulously writing a *README.md* file with everything one should need to be able to set up and run this themselves, we now had a fully fleshed out project that could work with another device over a network, or simply all run on one machine locally. The project was now officially complete after many hours of research, writing code, creating more bugs than we fixed, and eventually reaching the point we are at today.

Conclusion:

Now that we've finished our term project and are more than happy with how it turned out, we can take this time to reflect on the process as a whole. For starters we wish we would've been able to have a better plan when we started. When we first started working on this project our original idea was far too ambitious. We spent several weeks of meeting in the CS Lab trying to get our original idea off the ground and having to pivot over and over again. We did not anticipate just how many different issues we were going to run into until we eventually scrapped our initial idea entirely and came up with the project we have today. Although the challenges we ran into were difficult and confusing at first, they were not impossible. We're very proud of ourselves in regards to how we were able to adapt to the problems we faced and think critically and creatively in finding solutions to them. This was one of the most difficult and time-consuming computer science projects we have worked on but it was such a rewarding experience. We were able to bring together concepts from different classes we had taken throughout our careers at CSU and use concepts we were learning from Professor Pallickara and implement them ourselves in a project. We are all walking away from this experience with newfound knowledge of how API's work, a better understanding of the connection of devices

over a network, and how to not give up when fixing one bug creates ten more. Please check out our project on GitHub, we are very proud of it: Raspberry Pi: Security Camera - Twitter Bot.

Works Cited

1. “Create Your Own Twitter Bot in Python 3.10 Tutorial (2022 Edition).” *YouTube*, YouTube, 29 July 2022, https://www.youtube.com/watch?v=2UBcRiddwAo&t=60s.
2. “OpenCV Python Tutorials.” *YouTube*, YouTube, https://www.youtube.com/playlist?list=PLzMcBGfZo4-lUA8uGjeXhBUUzPYc6vZRn.
3. “Socket Programming in Python (Guide).” *Real Python*, Real Python, 16 Nov. 2023, realpython.com/python-sockets/.