Home Security IoT

Ramzey Ghanaim, Haruto Nakai

rghanaim@scu.edu
hnakai@scu.edu
Santa Clara University
500 El Camino Real, Santa Clara, CA, United States

***Abstract*— While many home security camera options exist, they are all expensive and backed by large companies that collect data in one form or another. Our project's goal is to bypass these flaws by creating a localized home security system a price less than $100.00, and we were able to achieve that with a setup of two Raspberry Pis, one acting as a server and another as a slave with a camera and a distance sensor, which can be replaced with more affordable materials to be able to market to an audience looking for IoT security at a lower cost than contemporary services such as Nest provide.**

I. Introduction

The markets for electronic security devices for personal use is growing. An article by Dan Scalco shows that the need and demand for security has gone higher and higher as the age of technology has progressed, showing estimates that, by 2020, the security market will be worth over $100 billion dollars, with home security systems taking up $47 billion of it, thanks to investments and attention from investors, smart home companies, consumers, and insurance companies. [1] In particular, smart home companies like Nest has stepped into this market, looking to tie security systems with their smart home appliances like thermostats and smart electronics; a basic security bundle that they offer, which includes door detectors and cameras, cost a total of $676[1].

Currently, there are a few companies that handle home security cameras. Nest and Ring are the most popular companies. Nest is backed by Google and Ring is its' own company. Both of these companies' camera products for home security cost above $100.00 and always have to be connected to an online cloud service where consumers' camera data is being stored. Our solution looks to solve these problems by using low cost technologies that consumers can easily acquire. We started by designing a client-server model. The server side is simply a Raspberry Pi 3B, while the client side consists of a Raspberry Pi 2 B, an ultrasonic sensor Longrunner HC-SR04 [2], and an Arducam OV2640 camera[2]. Together these items come to a price of $90.00, with $30 in the server and $60 in the client, which is cheaper than the alternatives from Nest and Ring, which cost more than $150 per device. the physical product of the sensor-camera prototype setup can be seen in Fig. 1.

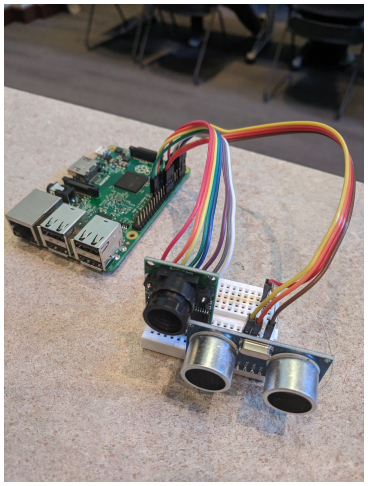

Fig. 1 Raspberry Pi 2, SPI Camera, and Ultrasonic sensor (left to right)

Another shortcoming of existing devices is the fact that all the camera capture is being stored and streamed by a 3rd party. Our solution requires users to sign into their email accounts to be used directly in our source code to authenticate with the customer's email service provider, under TLS encryption, that just about all modern emails work in. For this project, we kept things simple and used Google's Gmail.

Furthermore the Raspberry PI 3 acts as a centralized local hub that handles all the interactions of receiving and sending images to end users over email. We call the Raspberry Pi 3 a gateway. By using the customer's own personal email, they have full control over where camera images are stored and who receives them.

The Results of our project allowed us to determine that it is possible to have a personalized, and cheaper solution to home security without giving up privacy to a large corporation.

II. Design and Implementation

Since the end goal was to have a centralized local service to avoid any given cloud platform to sync and store user data from, we decided to use a Raspberry PI as a central hub. Keeping a localized system with no cloud sync allows users to be sure their data is private.

[1] https://nest.com/home-security-systems/
[2] http://www.arducam.com/arducam-mini-released/

Next we decided on using a low power, low cost microcontroller board with a low cost camera and sensor as our edge device to capture the environment.

Since the Pi's cost was higher than the edge technologies, we needed to design the Pi to be able to work as a central server. As a result, our project was built as a star topology, where microcontrollers and the Pi are communicating through WiFi. This enables the system to be very modular, allowing for one master to communicate with as many client modules as needed to provide security to a whole house.

A. Gateway

When designing the gateway, we first looked at the featuresets we wanted: handle multiple connections and processing at the same time, along with the ability to send emails in a programmable manor. Based on these requirements, we decided we needed Linux running on a Raspberry Pi. Linux on a Pi is lightweight enough to not be a power sore, while being able to handle multiple connections and threads. Lastly, with Linux we would be able to use Python to send emails, which makes the emailing side of the project a whole lot easier than if we wrote email sending in C.

As a centralized server it is the gateway's job to be able to receive pictures captured by the microcontrollers and email them to the end user. To support multiple edge controllers at the same time, the Pi will be running a multi-threading C program to receive and communicate with each microcontroller to grab images and call a python script to send the image as an email. With some testing and research, we discovered that it was best to keep the gateway to not run more than ten active threads at one time. After 10 threads, the Pi became slow, and sometimes crashed, but for the average home user, having 10 sensors capture movement all at the same time is unlikely. Furthermore, each thread takes little time to execute. Our testing resulted in at around one second to execute a thread.

The flow of gateway starts by waiting for incoming connections from clients. during this time the gateway blocks and waits to accept incoming connections. Once a connection is made, as long as less than 10 threads are currently running, a thread is fired. Each thread follows the following communication protocol:

- Gateway receives the file size
- Gateway sends an ACK string
- Client sends the file name of the image captured.
- Gateway creates a new file with the file name provided by the client.
- Gateway receives image file, 100 bytes at a time
- Once complete, the gateway closes the file and calls a python script
- The python script logs into the users email and sends an email to themselves via python's MIME email API.
- Thread exits

A flow diagram of this process can be seen in Fig. 2, and a screenshot of an email sent by the server board is on Fig. 3.

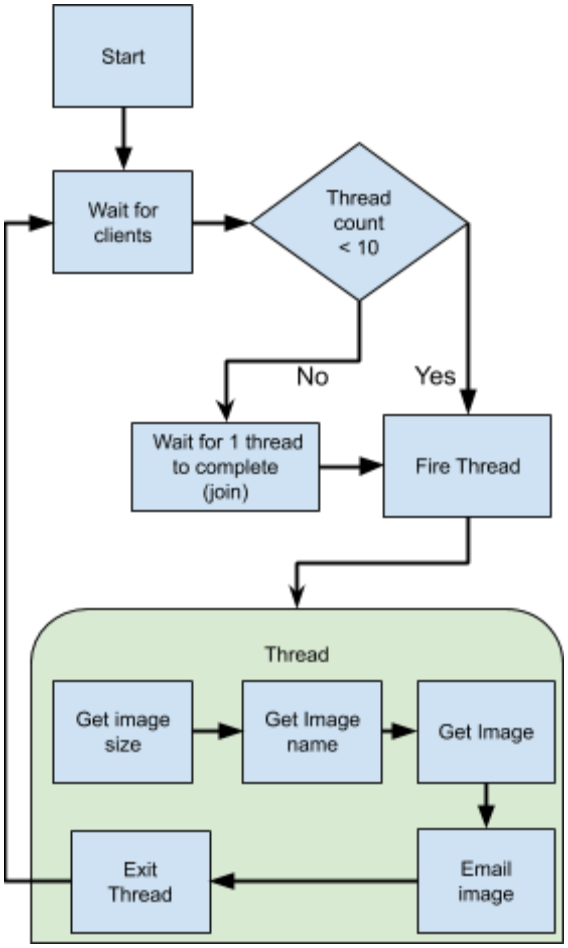

Fig. 2 A flow guide of the Gateway

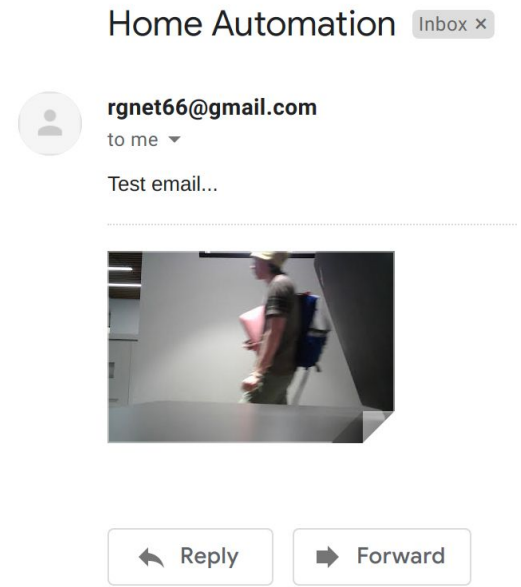

Fig. 3. Screenshot of an email sent by server

B. Client

The client side of this program involves using an ultrasonic distance sensor to detect nearby objects, and then take a picture with a camera, followed by sending the picture to the gateway.

The client code is entirely written in C using two APIs: one being the aforementioned API that accesses the camera using SPI and I2C, and the other being pigpio[3], a Raspberry Pi GPIO API which simplifies accessing the GPIO used for the ultrasonic sensor. The pin configurations are shown in Table 1.

Broadcom Pin #	Assigned Device	Pin Label
23	Sensor	Trig
24	Sensor	Echo
2	Camera	SDA
3	Camera	SCL
17	Camera	CS
9	Camera	MOSI
10	Camera	MISO
11	Camera	SCK

Table 1: Client Raspberry Pi pin assignments

A flow diagram of this process can be seen in Fig. 4. When the program is run, it first initializes each of the APIs, setting the GPIO pins for the sensor and applying settings for the camera. Specifically, the pigpio driver sets two of the GPIO pins as usable, one for input and one for output, and the I2C ports are used to initialize the pins and then the camera itself, then settings are applied so that it is using the correct configurations for the model and the size of the saved picture is set to the maximum resolution. The sensor is activated once in this initialization phase to set the expected distance between it and the wall without any obstructions, and the threshold distance for detecting a reading as obstruction as three-fourths of that. The distance between the sensor and the object can be found using the formula

$$d = (t_{end} - t_{start}) \times v_{sound}$$

, where t_{start} and t_{end} are the times of when the sensor's trigger pin starts and ends emitting a signal rounded to the nearest second, respectively, and v_{sound} is the speed of sound, which is approximately 34,300 cm/second. This is done by looking at the times at the beginning and the end of the pulse from Echo in the timing diagram as seen on Figure 5. Then, the main loop begins. The distance sensor is used to measure the distance in the method outlined above. If the distance that the sensor gets is more than the threshold, the program will determine that there was no obstruction, and will go to sleep for a second before returning to the start of the loop. If the distance is shorter, then it will branch off into the camera code, seeing that there is an obstruction, such as a passing human being. The camera is activated using the *single_capture* method from the API, which takes the picture with the specified parameter and sends the compressed JPEG image file to the board through the SPI interface, which the board will write to the local disk. Then it activates a separate program that establishes a socket connection with the server, and sends the name of the file, the size of the file, and then the file itself. Then it will sleep for five seconds as a cooldown to prevent the client from spamming the server, and therefore the user's email address, with the same pictures, before returning to the loop. As mentioned before, because all of the instructions are sequential, and because the code does not need to be run alongside anything else, no threading is required. This will allow for a wider range of compatibility on the hardware.

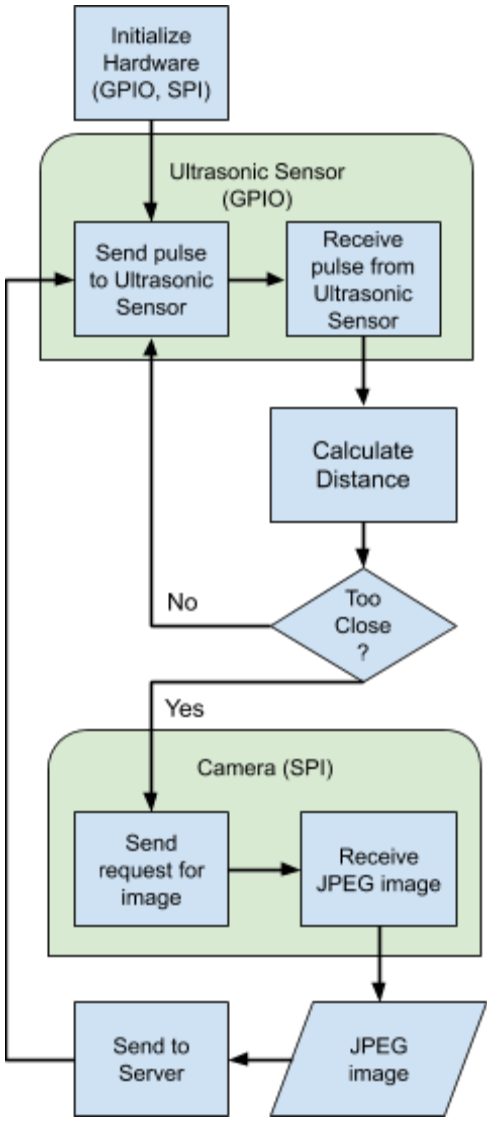

Fig 4. A flow guide of the Client

[3] https://abyz.me.uk/rpi/pigpio/

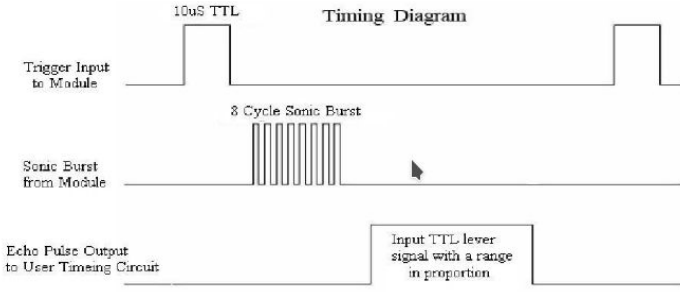

Fig. 5: Timing diagram for the Ultrasonic Sensor [2]

III. Results and Analysis

With our limited resources, we were able to have a completed working prototype: A sensor that detects objects and talks informs a microcontroller, followed by the microcontroller taking pictures, sending this picture to a gateway through C's networking library. All of this, followed by having the multi-threaded gateway receiving the picture and generating an email with Python APIs for the user.

While existing solutions can do similar things, we were able to achieve this with both supplies under $100.00, and while not relying on a large company who may be using your data without user consent by hosting the whole system under personally owned media, such as personal emails and software.

When timing our design, it only took less then a 3 seconds from when an object is in front of the sensor to the gateway sending the picture as an e-mail. From there the time to get the email entirely relies on the user's email client. We feel that 3 seconds delay is reasonable for this particular, non-life threatening Internet of Things application.

The sensor client code allows for low power usage, as the sensor is polled every second, which takes only a nanosecond of output and 1.16618e-5 seconds of listening, assuming the client is monitoring a 4 meter wide hallway. The most power consuming part of the code is using the camera and sending data over the Wi-Fi connection to the server, which does not happen very often. This allows the system to go to sleep at idle times, greatly reducing the amount of power used, leading to a much longer battery lifetime of the device, given that the monitoring client will sit outside powered by a battery.

IV. Conclusion and Future Work

Our developed solution to home automation shows it is possible to create a localized system for home security at a lower cost than existing solutions and arguably more secure due to the lack of any company backing the project. But since our time and resources on our research were limited, we did think of some ways to improve this project in the future.

There are many improvements that could go into lowering the costs of each client module. The Raspberry Pi 2B was simply chosen to be used for our prototype because it was one we had on hand and because there were high amounts of documentation on it. If we were to select a board to be used for production, any board that supports SPI, I2C additional GPIO controls, and Wi-Fi connection; a Raspberry Pi Zero W, worth $10, would do a fine job, but we can simplify the hardware further by using a chip with the minimal requirements and either a simple kernel or no OS, due to the fact that the code does not use any interrupts. There are other cameras that could be considered for actual production as well. If we planned not to convert to JPEG format on the camera and do so on the board, we can consider using a MIPI CSI camera, which would send the uncompressed image over multiple lines. This would negate the need for a JPEG compressor and an SPI FPGA on the camera, lowering the camera costs to $5. These two choices, with the $5 cost of the sensor added, would bring down the hardware cost of each client module to be $20, a third of the prototype. Even if marketed at a markup of 50%, or $30, the client module is extremely cheap and affordable compared to the competitors we have today like Nest's systems, and should be a boon to households that do not wish to purchase and manage an expensive full smart home system.

An issue with the camera that may harm its usability is motion blur. Because the target that the camera is trying to capture is moving as it is passing the ultrasonic sensor, there is bound to be motion blur on the picture relative to the background, as seen in Fig. 5. If this was to be used as a home defense system that allows the user to identify the intruder for coordination with law enforcement, not being able to identify the cultprit's face clearly will be a problem. To remedy this, the code could be modified to take multiple pictures and select the picture with the least blur using graphical utilities such as OpenCV and its Laplacian filter [3], or take a short video instead of a single picture.

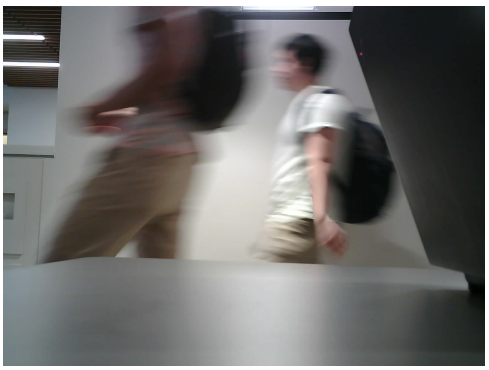

Fig. 5: Picture captured with camera, blurred

On the gateway side, as it stands, the email sender logs into an email and sends an email from that particular

email address to that particular address, essentially sending an email to yourself. The problem arises how the credentials are hardcoded into the python program. So this can be a major risk. Future work would have us store the credentials encrypted on the Pi and have the email sender decrypt and use the credentials on the fly. We tried experimenting storing the credentials in an ansible-vault, but could not complete it in our limited time span. But after looking into the process, using ansible-vault to encrypt and decrypt the credentials is a feasible metric to achieve to secure the users login credentials.

Another email issue, revolves around how the company behind the email client may spying on you. So, we would need to make our own local email server running on the gateway in a future expansion.

Nevertheless, we solidified our concepts and were able to have a successfully working design by the end of this project, with great ideas to expand for the future.

REFERENCES

[1] Scalco, D. (2019). Why Home Security Systems Are on Track to Be a Multi-Billion-Dollar Market. [online] Inc.com. Available at: https://www.inc.com/dan-scalco/why-home-security-systems-are-on-track-to-be-a-multi-billion-dollar-market.html [Accessed 12 Jun. 2019].

[2] Freaks, Elec. "Ultrasonic ranging module hc-sr04." HC-SR04 datasheet (2016).

[3] Culjak, I., Abram, D., Pribanic, T., Dzapo, H., & Cifrek, M. (2012, May). A brief introduction to OpenCV. In 2012 proceedings of the 35th international convention MIPRO (pp. 1725-1730). IEEE