Sensor Deployment in Smart City IoT Lab

Li Wei Xing
Ruston Smart City IoT Lab
Ruston, United States
sli@jlisbz.com

***Abstract*—The Ruston smart city IoT Lab has been able to use Raspberry Pis to deploy sensors all over the Louisiana Tech University Campus and output the results. The Raspberry Pis can be accessed remotely through the terminal.**

Keywords—Raspberry Pi, sensors, mote, Python

I. Introduction

The concept of smart cities has been gradually integrated into our society. As people become more interconnected and dependant on technology, the goal of smart cities is to provide citizens with a better quality of life. To connect everything together, smart cities utilize the Internet of Things where IOTs are connected together the internet. An IoT is comprised of many different things, for example, cellular devices, laptops, and Raspberry Pis. Smart city technology will allow people to communicate not only with each other but can communicate with the city's infrastructure and can be informed on what is happening around them.

Sensors are connected to an IoT to collect data and then use the information found to create efficiency within the city and provide safety. These sensors can range from detecting the different levels of gas in the air, which parking spots are available, and severe weather. Another advantage of having a smart city is that all the information gathered would be in real-time; therefore, time-sensitive information will never be overlooked or missed. Another upside to smart cities is that it can be used for economic gain. Cities can tell where energy consumption is at its high and what steps they can take to reduce energy consumption and lower the cost of operations. Moreover, cities can be informed on where and when traffic occurs on the road and come up with ways to efficiently reduce the amount of traffic and carbon released in the atmosphere.

For our experimental purposes, we have decided to use the Raspberry Pi 3. This Raspberry Pi was chosen because it was affordable, easily portable, and the computation ability of the Raspberry Pi is sufficient for our experiment. There were various sensors bought from Amazon and Adafruit used to attach to the Raspberry Pi. Some sensors include the BME 680, Lux sensor, and PIR motion sensor. Moreover, Python was the programming language of choice as it is compatible with the Raspberry Pi. Some challenges we have encountered is how we can send the data received efficiently and effectively from the Raspberry Pi to our database. As well, the question of if fog/edge computing would be useful for our particular research.

In this paper, we will focus on how to set up a Raspberry Pi, how to ssh into the Raspberry Pi, how to set up the sensors, and how to read and get the information gained from the sensors. This paper is organized as follows. Section II talks about the background and the related works of implementing the Raspberry Pi, fog/edge computing, and the transfer of data to the database. Section III talks about how the Raspberry Pi was set up. Section IV talks about the deployment of the Raspberry Pis and their locations. Section V talks about future work and discussions.

II. Background

A. Raspberry Pi and Sensors

Raspberry Pis were created in the United Kingdom in 2012 and are small single-board computers [1]. It is essentially a cheap computer that runs on Linux and has different GPIO ports to allow you to control electronic components or explore the Internet of Things [2]. The goal of the raspberry pi was to create something small, portable, and accessible all around the world with the same computational power as a computer does. Over time, different models of the Raspberry Pi were made, each with their own features and better computational abilities. Here is a summary of the five Raspberry Pis made to date.

1) Raspberry Pi 1: There were four models made, B, A, B+, and A+. The B and B+ models had ethernet ports while the A and A+ didn't. The B, A, B+ models were standard size (85.60*56.5mm) while the A+ model was a compact size (65*56.6mm). The Raspberry Pi 1's were not made to be wireless. Moreover, models B and A had 26 GPIO pins while the B+ and A+ had 40.

2) Raspberry Pi 2: There was only one model made, B. This model was standard size with an ethernet port but did not have the wireless feature. As well, it had 40 GPIO pins.

3) Raspberry Pi Zero: There were two models made, Zero and W/WH. These did not come with ethernet ports and were 65*30mm big. Moreover, they both had 40 GPIO pins and the Zero was wireless where the W/WH was not.

4) Raspberry Pi 3: There were three models made, B, A+, and B+. The B and B+ were standard sizes where the A+ was the compact size. All three were wireless; however, only the B and B+ had ethernet ports. Moreover, All three models had 40 GPIO pins.

5) Raspberry Pi 4: There were three models made, B(1 GiB), B(2GiB), and B(4GiB). All three models are standard size, have ethernet ports, 40 GPIO pins, and were wireless.

Having so many options to choose from, we found that the Raspberry Pi 3 was most suitable for this paper. That is because it was cost-efficient, had ethernet ports, was wireless, and had a sufficient amount of GPIO ports for us to attach sensors too. The sensors themselves were all bought online, mostly using the company Adafruit. Adafruit was used because their sensors were relatively easy to use and can output accurate data. As well, the cost of most of the Adafruit sensors was reasonable, allowing us to buy a lot to be attached onto our Raspberry Pis.

B. Related Work

There has been a lot of research done on implementing sensors within a smart city. Researchers have been using Raspberry Pis to do so as it is cost-efficient and can provide the researcher with a good amount of data to analyze before investing a lot of money. There have been many research papers about the application of Raspberry Pis with different sensors implemented into the smart city for things such as smart waste management, smart traffic light scheduling, and LDR bulb applications. Moreover, the concept of transferring the information gained to the database has been researched a lot; whether one should use fog/edge computing and what kind of communication protocols are used to transfer the information.

Even though containers are a new topic, they are already heavily used and researched. A paper by Classsen describes the use of containers, the networking performance, and the scalability of the kernel modules. It is said that containers are used to provide network connectivity among containers and also create virtual network devices. After, these network modules were put to the test to see the performance and they found that UDP had the lowest performance.

A paper by Ahmed and Pierre was about how we can implement fog/edge computing infrastructures into smart cities. They confirmed that Raspberry Pis offer amazing performance, cost, and energy ratios for use in research. The research was to detect the deployment time, network activities, disk throughput, and CPU usage to measure container deployment performance.

III. Raspberry Pi and Sensor Setup

The Raspberry Pis were first given a disk image through the software balena etcher. The disk image was downloaded onto the USB that the Raspberry Pi provided and then, the USB that contained the disk image was inserted into the Raspberry Pi. After that, the Raspberry Pi was connected to a monitor to start the setup. We enabled the I2C ports to provide access to the I2C bus ports, we enabled the camera just in case we wanted to attach a camera to the Raspberry Pi, we enabled the SSH to remotely access the Raspberry Pi, and as well, we enabled the GPIO ports to use the sensors. In order to ssh into the Raspberry Pi, we had to be on the Louisiana Tech University's wifi. That is why we used Global Protect to get into the La Tech WPA from wherever we were.

We added duckdns to each Raspberry Pi therefore we didn't need the IP address of each Pi. Instead, we had a specific name to identify the different Pis and be able to access the one we wanted. We used www.duckdns.org to set up the different duckdns domain names. Next, we changed the user interface to the command line and removed the Pi from the monitor. After connecting an ethernet port to the Pi, we sshed into the Raspberry Pi from our laptops and continued to work on it through there. Lastly, a breakout board was attached to the Raspberry Pi to get the Pi ready for the sensors.

We mainly experimented with four sensors, the BME680, Lux sensor, the PIR motion sensor, and the I2S MEMS Microphone. The purpose of this experiment was to be able to attach the sensors to the Pi, be able to generate a code in Python to run the sensors, and be able to get readings from the sensors.

1) BME680 – The BME680 sensor was bought from Adafruit. Figure 1 represents the Python computer wiring[3].

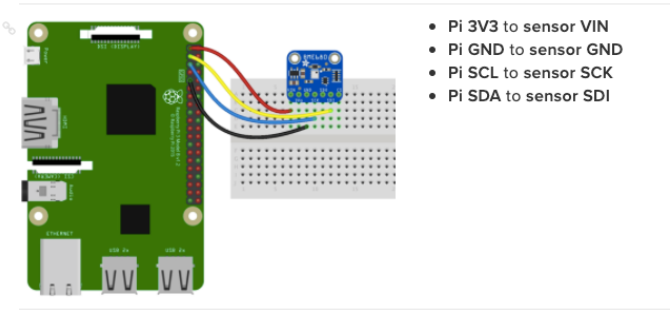

Figure 1: Python Computer Wiring for a BME680 sensor

First, the BME680 sensor requires the Circuit board to be running the latest version of Adafruit CircuitPython. Then, the BME680 sensor requires the user to install the Adafruit CircuitPython BME680 library onto the CircuitPython board. The adafruit_bme680.mpy and adafruit_bus_device must be manually installed, separate from the library already installed. After the python code is created, we were able to get the temperature (measured in Celsius), gas (measured in Ohms), humidity (measured in percent humidity), pressure (measured in hPa), and altitude (measured in meters). This python code was done in a while loop; therefore, it was continually running once the program had started.

2) Lux Sensor – the Lux sensor we used was the TSL2561. The TSL2561 sensor was bought from Adafruit. Figure 2 represents the Pythong computer wiring.[4]

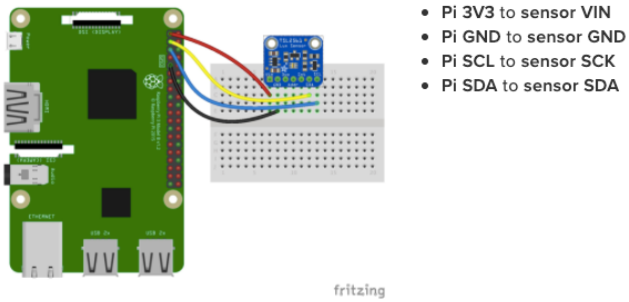

Figure 2:Python Computer Wiring for a TSL2561 sensor

First, the TSL2561 sensor requires the Circuit board to be running the latest version of Adafruit CircuitPython. Then, the TSL2561 sensor requires the user to install the Adafruit CircuitPython TSL2561 library onto the CircuitPython board. The adafruit_tsl2561.mpy and adafruit_bus_device must be manually installed, separate from the library already installed. After the python code was created, the sensor would be able to detect lux (The computed light lux value measured by the sensor), broadband (measures the broadband channel value), infrared (measures the infrared channel value), and the luminosity (measures A 2-tuple of broadband and infrared channel values).

3) PIR motion sensor - PIR sensors, often referred to as, "Passive Infrared" or "IR motion" sensors, enable you to sense motion[5]. To connect the PIR motion sensor to the break out board, we first plugged three of the male to female jumper wires into the three pins on the PIR sensor. The red pin represents the PIR-VCC, the brown pin represents the PIR-OUT, and the black pin represents the PIR-GND. The red pin is plugged into the positive rail of the bread board. Then, the brown pin is plugged into the black rail while the black pin is connected to the negative rail of the break out board. After that, a red jumper wire is connected to the GPIO 5V on the Pi to a positive rail of the break out board. Then, a GPIO pin is selected as an input to sense when the PIR sensor will detect motion. The last step is to connect the GPIO pin to the same rail as the PIR-OUT. Figure 3 represents the wiring done to attach a PIR motion sensor to the Raspberry Pi.

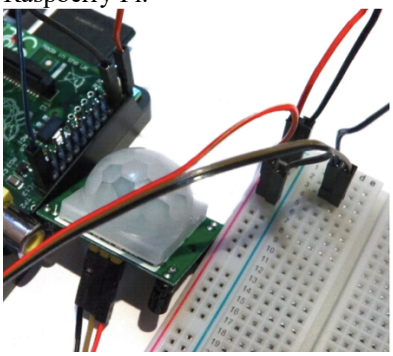

Figure 3:Python Computer Wiring for a PIR motion sensor

The PIR motion sensor also comes with sensitivity options. There is an orange time knob where if you turn it to the right, it will increase the time delay of detecting motion. If you turn the orange knob to the left, this will decrease the time delay of detecting motion. There is another knob that will increase or decrease the sensitivity of detecting motion. In order to write the Python code for the PIR motion sensor, the Python GPIO library was imported. Then, the time library was imported and as well, the GPIO pin numbering was set. That GPIO pin was the pin that allowed the PIR motion sensor to send information to the Raspberry Pi. After that was set up, we were able to write the Python code to run the PIR motion sensor. The Python code was set up to where if there was motion detected, it would be able to calculate how long the motion was detected for and output the results.

4) I2S MEMS Microphone – This microphone was bought from Adafruit. This microphone can detect sound and convert it to voltage with a range of about 50 Hz – 15KHz. The main use of this sensor is to record and detect general audio[6]. There are three digital pins on the microphone; clock, data, and word-select clock. When it is connected to the Raspberry Pi, the I2S will drive the clock and word-select pins at a high frequency and read the data from the microphone.

The microphone was installed as followed. The chip was first soldered onto the break out board to ensure connectivity. Then, the ground pin on the microphone was connected to the ground on the break out board. Next, the 3.3V pin and SEL was connected to the 3.3V on the break out board. The BCLK on the mic was connected to the BCM 18 on the Pi, the DOUT on the mic was connected to the BCM 20 on the Pi, and the LRCL on the mic was connected to the BCM 19 on the Pi. Figure 4 is a diagram of the Python computer wiring for the I2C MEMS microphone.

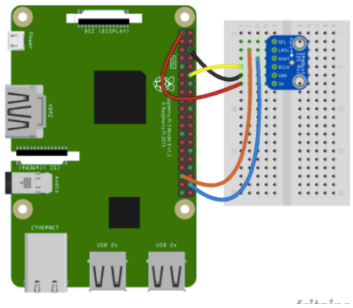

Figure 4: Python Computer Wiring for a I2C microphone

We were lucky enough to find a Python script already done for the I2C microphone; thus, all we had to do was download the file onto the Pi. After we made sure we have the correct input device, we tested the code by running:

arecord -D plughw:0 -c1 -r 48000 -f S32_LE -t wav -V mono -v file.wav

This code will start the microphone to record the input on the file.wav file. After that, we ran:

aplay file.wav

This code will play the file back directly on the Raspberry Pi if there are speakers connected to it. If not, we ran this code:

scp pi@raspberrypi:file.wav ~/Desktop/file.wav

This code will copy the file onto the user's Desktop; therefore, if the Raspberry Pi does not have a speaker, the file can be ran on the computer where we could hear the audio recording.

IV. Deployment of the Raspberry Pi

In total, we deployed 8 Raspberry Pis which we called motes. Some motes had sensors on them and some did not.

Our primary goal was to be able to access the motes from our laptops without being present in the same room. Moreover, our additional goal was to be able to run the Raspberry Pis and get the data outputted onto the terminal. Table 1 is a summary of each of the motes deployed and what sensors are attached to each mote.

Table 1: Summary of all the motes and sensors deployed

Mote Name	**Sensors Attached**
Robby	N/A
Sophie	N/A
Stella	N/A
Blue	N/A
Larry	BME680, TSL2591, PIR motion sensor
Rocket	BME680, TSL2591, PIR motion sensor, low noise sensor
Winston	BME280, TSL2591, kuman soil moisture, camera

All the motes were successfully accessed through the terminal and are active. Moreover, all the sensors attached were able to provide data from the room they are in.

V. Discussion and Future work

Our goal in the future is to be able to write a research paper about our findings. The paper will consist of seeing the performance from using just the Pi to get, store, and send information; compared to using Docker containers to do so. After our findings, if the Docker containers do not show significant performance differences, then we would be able to virtually deploy thousands of virtual motes using the Docker containers. The purpose of deploying many motes is to create a smart city effect of how the system can handle many devices sending information to the database. If the system crashes, we can then alter the algorithm to suit it for a smart campus. Moreover, we can analyze the data and find correlations with the sensor readings to the environment surrounding it. Our goal is to be able to provide efficiency and safety for the people living within the smart campus.

The use of fog/edge computing was discussed with this smart city project. The thought of implementing fog computing can be useful when creating a smart city as the central database will not be so busy trying to do all the computations and outputs. With fog computing, we can divide the work into different areas; therefore, provide a faster response time, lower latency, and more accurate information based on the area the person is located.

References

[1] "Raspberry Pi." *Wikipedia*, Wikimedia Foundation, 4 May 2020, en.wikipedia.org/wiki/Raspberry_Pi.

[2] "What Is a Raspberry Pi?" *Opensource.com*, opensource.com/resources/raspberry-pi.

[3] Ada, Lady. "Adafruit BME680." *Adafruit Learning System*, learn.adafruit.com/adafruit-bme680-humidity-temperature-barometic-pressure-voc-gas/python-circuitpython.

[4] DiCola, Tony. "TSL2561 Luminosity Sensor." *Adafruit Learning System*, learn.adafruit.com/tsl2561/python-circuitpython.

[5] Hut, Pi. "Raspberry Pi GPIO Sensing: Motion Detection." *The Pi Hut*, The Pi Hut, 16 Nov. 2017, thepihut.com/blogs/raspberry-pi-tutorials/raspberry-pi-gpio-sensing-motion-detection.

[6] Adafruit Industries. "Adafruit I2S MEMS Microphone Breakout - SPH0645LM4H." *Adafruit Industries Blog RSS*, www.adafruit.com/product/3421.