Bachelor's thesis

Information and Communications Technology

2022

Jonathan Ecker

COMPARING IOT SYSTEM DEPLOYMENT ON RASPBERRY PI 4 MODEL B AND ODROID-C4 DEVICES

BACHELOR'S THESIS | ABSTRACT

TURKU UNIVERSITY OF APPLIED SCIENCES

Information and Communication Technologies

2022 | 27

Jonathan Ecker

COMPARING IOT SYSTEM DEPLOYMENT ON RASPBERRY PI 4 MODEL B AND ODROID-C2 DEVICES

A common problem for microcontroller devices and most technology, in general, is measuring and comparing devices or components to determine which product best suits a situation and the benefits or drawbacks each possesses. The goal is to investigate the ability for alternative microcontrollers to outperform the Raspberry Pi, which is the most common microcontroller device on the market. The common deployment of the Raspberry Pi device results in a large portion of research and education being carried out on one device, with the competing products often overlooked as a result. By attempting to compare the devices, it can be determined whether there are better alternatives for both education and personal use for monitoring and experimentation. Although the scope of this project is limited to two microcontroller devices the idea is that the system should be designed in such a way that it can be implemented on any microcontroller device and have the experiments conducted in the exact same way.

In this project, the Raspberry Pi and Odroid devices-are paired to a Bluetooth weather station which then takes selective measurement values consisting of humidity, pressure, and temperature and performs a series of mathematical operations on the data using principles of edge computing to determine how well each device performs. The use of Linux distributions which support Python is key to the universal nature of the experimentation being successful. If a microcontroller device supports a Linux operating system and can run Python scripts, then the experiments can be repeated exactly as listed and compared to the findings in this study. Regarding the findings it was initially expected for the Raspberry Pi device to be the clear favorite due to the superior hardware components available to the device. Contrary to the initial expectation, while the hardware advantage allowed the Raspberry Pi to perform better on pure mathematical operations it was not quite as dominant in other areas and even lost in performance to the Odroid device at times. For example, one of the aspects that the Odroid device performed better at was accessing and returning information directly from data storage.

The key takeaway of the project and experimentation is that there are many more consumer microcontroller devices available than just the Raspberry Pi and as demonstrated through the experiments, some will perform better in their niches. While the experimentation is not fully comprehensive, it looks at the most common solutions and deployments and crafts a rough estimate of a device's abilities. While the Raspberry Pi is still a very good option for most microcontroller experimentations, hopefully, the results and findings can compel a look into alternative microcontrollers as well.

KEYWORDS:

Raspberry Pi, Internet of Things, Microcontrollers, Edge Computing, Bluetooth

CONTENTS

LIST OF ABBREVIATIONS

ARM	A type of lightweight computer architecture used by processors with set instructions.
DB	Database, is a file system used to organize and stored collections of data and information electronically.
IoT	Internet of Things, a descriptor of a system or device which transmits information, measurements, or other data over the internet or other networks.
MQTT	A publish/subscribe network that transports messages between devices
OS	Operating system, the default base installation for any device from microcontrollers to home computers.
SQL	Structured Query Language, used for communicating with databases and database management systems.
STEM	It refers to any subjects in the categories of science, technology, engineering, and mathematics.

1 INTRODUCTION

When developing systems for automation and measurement there is a need for devices which are capable of handling complex mathematical procedures, storing data, or transmitting information. The most commonly used devices are called IoT devices which are comprised of a processor, memory, storage, and a communications interface found in every typical computing system. The distinct difference is that IoT devices are smaller in scale since the environment they are used for is often remote tasks or measurements where the information is sent to a larger computing system with more available processing power. Although with increased advancements in the technology being used, that is set to change with the advent of edge computing becoming more widely used and available. This means that much of the processing and computing can be carried out at the sensor location as well.

The Raspberry Pi 4 Model B and Odroid-C4 are two readily available, open source devices on the market that can be used for IoT development. To better understand the benefits in hardware and platforms offered by these devices, near identical IoT deployments will be built on each device and tested to determine their strengths and weaknesses. The IoT system in question will take a measurement, store the data on a database,and generate an analysis based on various sizes of selected datasets over a pre-determined timescale. The system will be measured for speed, accuracy, ease of development, and the ability of each device to be set-up from a freshly flashed state.

The objective is to examine a number of factors that go into using IoT microcontroller devices especially with their increased useage in education. For example in [1] the authors state a need for microcontroller devices (specifically the Raspberry Pi) to help bridge the gap in STEM education especially in developing due to the low cost required to purchase these devices. They put a large emphasis on the amount of benefit integrating microcontrollers into education helps a nation’s STEM growth when implemented at a high school and secondary school level.

However not all of these factors can be selected for using purely mathematical analysis, since some factors such as manufacturer support, price, and project scope could outweigh the pure computational benefits one device may offer over the other. According to the authors in [2] IoT is set to see exponential growth and integration in all of our lives and the availability of open access to data provided by these systems will help both citizens and the government better understand what is happening in the areas these monitoring systems are established at. With IoT projects, an increasing number of environmental, weather, and other monitoring systems will be able to be established at a wider array of locations creating a dispersed network of sensor with more real-time information coming in all the time.

The goal of this project is to explore alternative microcontroller devices for both educational and consumer use and test them versus the most commonly used microcontroller on the market, the Raspberry Pi. The idea is that this could open up more avenues for research and development, or may make educational opportunities better using alternatives to the Raspberry Pi. The hope is that these other devices may perform just as well at a lower price point, or offer a certain niche use where it outperforms the Raspberry Pi. Having a broader knowledge of microcontroller devices is also a net benefit for students as it allows them to explore the inner workings of the devices more in depth due to the minute difference in how they operate.

When researching other projects which investigated IoT systems, one of the most common types of projects was monitoring weather and air quality. One of the perceived reasons for this was listed in [3], citing the instability of climate and weather changes having an effect on everyone, but the current monitoring solutions are not accessible to average individuals and, therefore, there is a large amount of research and development going into creating low-cost IoT weather solutions. For that reason, it made sense to also create a simple, useful, and low-cost system in order to use as a base for comparison between different microcontroller devices.

The thesis is organized as follows, Chapter 2 describes the two IoT devices being used in the experiments as well as the Bluetooth temperature sensor being used to relay measurements to the devices. Chapter 3 discuses the types of software, measurements, and programming choices that were made to have as close a comparison as possible between the devices. Chapter 4 goes into the results and observations made throught the experimentation and gives explanation as to how the data was collected and what the differences between devices means. Chapter 5 complites the findings and evidence reached throughout the investigation and how these findings might be useful when testing other microcontroller devices, as well as how establishing how the Raspberry Pi and Odroid devices performed overall.

2 DEVICES

The selection for devices was done by initially choosing the Raspberry Pi 4 Model B since it is the most commonly used microcontroller in schools and personal development. As such it has a lot of support online for it's development and a custom operating system that is easy to install (Rasbian OS). The Odroid-C4 device was chosen since it is around the same price range and has comparable hardware performance to the Raspberry Pi allowing a direct comparison without factors like processing power or other hardware differences being too large of a factor between devices.

Although the devices chosen are similar in price range and hardware components there are still some difference between the devices hardware and software environments that will be expanded upon below.

2.1 Raspberry Pi 4 Model B

The device pictured below (Figure 1) uses the Raspbian OS which is a closed-source linux distribution that has support for many languages but promotes Python as the primary programming language and is the one that will be used for this project. The noteable hardware components for the IoT device is an ethernet port, a WiFi interface, and 40 GPIO pins. The processor is a Quad Core ARMv8 64bit processor clocking at 1.5GHz, with 8GB DDR4 memory, and a slot for a Micro-SD card [4].

Figure 1. An image of the Raspberry Pi 4 Model B device

2.2 Odroid-C4

The Armbian OS is an open-source operating system similar to Rasbian but available for all microcontrollers and will allow the programs to be written in Python allowing the code scripts that will operate on each device to be identical. The device which can be seen below (Figure 2) contains an ethernet port, 25 GPIO pins, a serial console port, and Wi-Fi receiver. The processor is a Quad Core ARMv8 64bit processor at 2.0GHz, with 4GB DDR4 memory, and a slot for a Micro-SD card for storage [5].

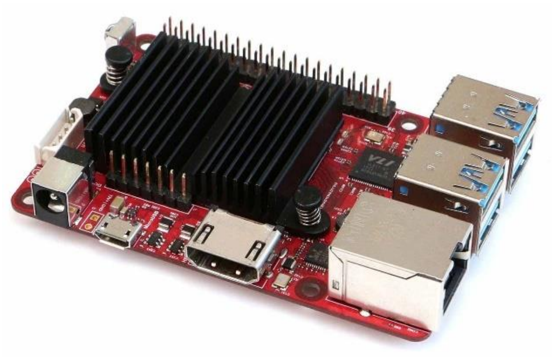

Figure 2. An image of the Odroid C4 device.

2.3 Ruuvi Bluetooth Weather Station

The Bluetooth component as seen in the image below (Figure 3) is designed to send data through beacon advertisements displaying multiple data types but for the purposes of a weather installation it only needs to select for temperature, humidity, and pressure. It is most commonly accessed through pairing with a phone or other device using Ruuvi’s own “Ruuvi Station” Application available on any app store. But for developers the device can be directly interfaced with using the “Ruuvitag_sensor” Python library available as an open-source development tool hosted on Github. This allows both the Odroid-C4 and Python devices to receive updates from the sensor to track weather readings in a more direct, barebones manner which will allow better analysis of the raw differences in data transmission.

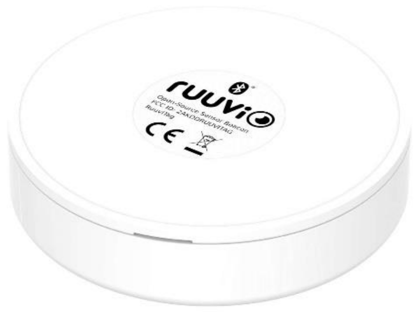

Figure 3. An image of the RuuviTag Bluetooth device.

3 IOT STRUCTURE

The proposed design shown in the UML diagram below (Figure 4) dictates how the process should work within both the Raspberry Pi and Odroid device. None of the code scripts, database format, or the dashboards need to be adjusted or changed between devices. The idea of a generalized solution for all IoT microcontroller devices makes it so that the device itself is being measure rather than optimizations or changes within the code or the devices own internal structure. Since this includes the programming language being used; Python was chosen since it handles real-time movement of data the best and is lightweight meaning it can be deployed and modified easily.

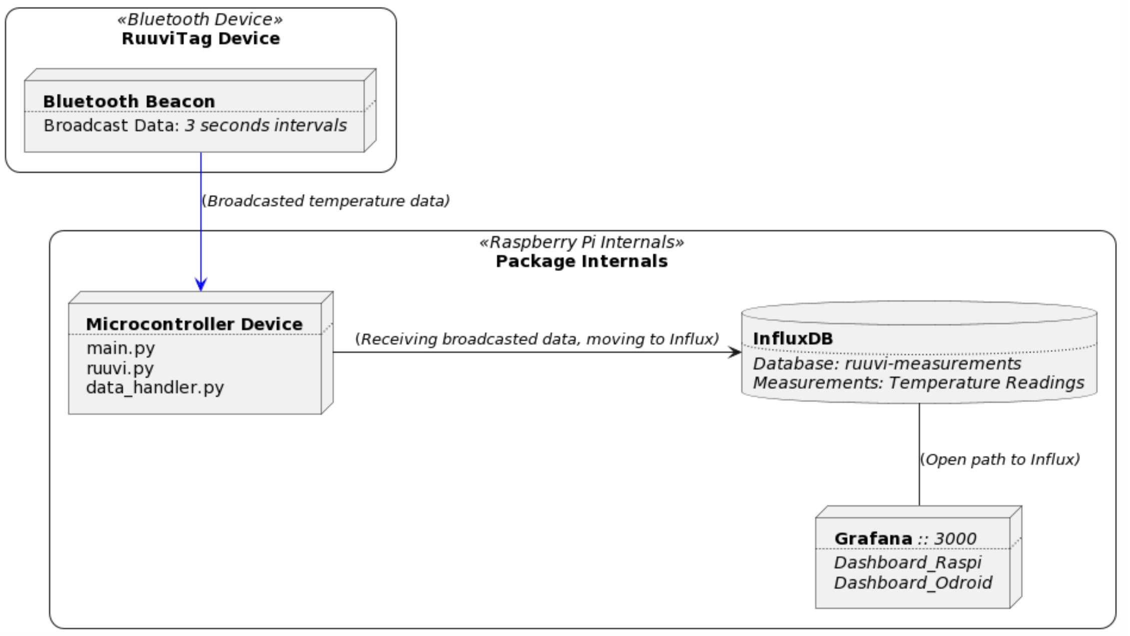

Figure 4. UML diagram showing the movement of data from the RuuviTag device through a generic microcontroller (Raspi/Odroid) to a Grafana dashboard.

The added benefit of the design is that while it currently only handles a singular Bluetooth beacon sending data to the microcontroller it can be expanded to include more beacons or other sensors broadcasting data to the same microcontroller device. By using a more simplistic approach, the design is able to be expanded later to create a more complex system without needing to restart the design from scratch.

3.1 Software choices

The main difference in deployments between the devices was the selection of OS for each device. The reason for using Rasbian on the Raspberry Pi instead of Ubuntu ARM which was used for the Odroid was that Rasbian is a natively supported OS for the Pi and is a major selling point of the device due to its ease of set-up.Therefore the choice to use different OS was made since it is one of the major reasons behind the devices success.

Each device will need to run a local database and a script that receives a Bluetooth transmission from the sensor and forwards the information into the database. For the database, initial testing was done using an SQLite3 database but was changed to InfluxDB database after issues were found due to the SQLite3 database losing entries, and having massive differences in storage time between devices. While the goal of using SQLite3 was that it was lightweight and quick to set-up it came with too many issues and problems to continue its use.

InfluxDB was instead chosen due to it’s easy integration with Grafana which allows for real time monitoring of the data, as well as a visual representation of the results. It is also a time-series database which means that every entry is automatically tagged with the current time and is not reliant on the devices to do it manually. To communicate with the Bluetooth sensor, a series of simple Python scripts will be written that send the results from the ruuvi_sensor library (available on Ruuvi’s GitHub) directly to the database using the “influxdb” Python library. All data values are sent as a JSON dictionary object which can be natively stored within InfluxDB.

3.2 Considerations for measurement choices

While there are many measurement types to consider such as monitoring CPU temperature which was done for computer vision algorithms in [6], or when looking at power consumption using simple randomized algorithm problems in [7]. These measurements were meant more for the devices health rather than the success of an IoT sensor deployment. Rather looking within the implementation for points where the systems might diverge in success it came down to the following sections:

A. Communication with the sensor

B. Sending data to the database
C. Retrieving data from the database
D. Applying operations to the data

Looking at these main points where a majority of the data was being sent, stored, and retrieved these sections would be where the largest discrepancies between devices would be found. After looking at the datasheet available for the Ruuvi weather station it was decided to store Temperature, Humidity, and Pressure. Next looking at the example in [8] there was a very thorough monitoring for the delay of data sent from the sensor. While the implementation in that case was using MQTT it felt important to use the delay in messaging for Bluetooth as well. This is especially important in this case since the Odroid device does not have innate Bluetooth and must use a dongle, whereas the Raspberry Pi has it built-in. This difference in the method of achieving Bluetooth connectivity might have an impact on the speed and accuracy of data collection.

The final aspect was looking at retrieving and processing the data, the selection was made to use a small and large dataset which was set to be a 24-hour period and then a 7-day week of measurements. The data would then use a simple algorithm and compared to the built-in SQL measurements for median, mean, and count. This gives an indication of how fast the baseline method is, how fast the simple algorithm is calculated with the device, and how the different devices compare in each category.

3.3 Programming decisions

The simplest method to test connectivity, process data, and store data is to use Python scripts executed locally on the device. Python is an interpreted language meaning it does not need to be compiled to run which makes it easier to set-up, and faster to modify. The code will be designed to receive a sensor reading over Bluetooth, modify and attach the time delay to the data, and store the response in a database for access by Grafana in order for a simple graphing dashboard to be created. Creating an interpreter on a microcontroller device that operates between a sensor, and a remote user is a very common IoT implementation.

This type of IoT solution is called edge computing and it solves the common problem of how to deal with the large influx of data. One of the primary reasons that edge computing is implemented in IoT systems is to handle and sort the large amount of data being generated by the sensors before sending it remotely to be stored. As indicated by [9] the establishment of edge computing helps the end user receive a pre-processed response with a low response time due to the real-time processing of the sensors data at the local or “edge” level of the system.

The way it is created will be to first attempt to connect to the Ruuvi device using the “Ruuvitag_sensor” library and denote whether the connection was a success or a failure. If the connection was a success the device will proceed to open a connection to the local database in order to send the most recent average temperature, time the length of the process from first receiving the Bluetooth connection to when the data is successfully stored, and lastly to format the received data and store it using an SQL command. The result will be a database consisting of sorted data readings which can be queried in a variety of ways and will later be accessible on a remote device for analysis.

4 TRIALS AND RESULTS

Using the IoT structure described in Figure 4. both microcontroller devices will be undergoing timed analysis of the scripts when entering values into the database, as well as the scripts that query data from the dataset to check their speed, accuracy, and to note any problems or irregularities which occurred. The testing is intended to produce results and highlight drawbacks between these devices.

4.1 Bluetooth Connectivity and Data Storage

The first portion of the trial was to create a link from the microcontroller devices, to the Bluetooth beacon and to stored the results on the microcontroller device in a database. The code seen in Figure 5. is what was used to create a list of entires and archive the time it took to write the entries to the database.

```
#Variable to set the timeout speed of the RuuviTag sensor.
timeout_in_sec = 2

#Function to read the beacon and get a response back.
def read_beacon():
    time_start = time.time() #Starting the timer
    data_packet = RuuviTagSensor.get_data_for_sensors(sensor_id, timeout_in_sec)
    time_stop = time.time() #Ending the timer
    time_result = time_stop - time_start  #Getting the resulting time

    #Adding measured time to the JSON data object
    data_packet[sensor_id]['retrieval_time'] = time_result
    return data_packet[sensor_id]
```

Figure 5. The code snippet which handles how fast the RuuviTag sensor is checked.

Initial testing of the devices resulted in a huge discrepancy when measuring the Odroid-C4 device when compared to the Raspberry Pi. When taking into account the time it took to receive a broadcast from the RuuviTag device the “timeout” used was lowered to the minimum acceptable value. This resulted in the timeout value being 2 seconds for the Raspberry Pi and 3 seconds for the Odroid-C4 device. The way the minimum value was determined was running the program over one hour and if the timeout value was too low the rate of exceptions quickly resulted in the device missing over 50% of it’s attempts to get a response from the beacon. As indicated in Table 1, the result from a one hour test was that the Raspberry Pi and Odroid-C4 devices are similar in the number of entries dropped or missed during a session with both devices dropping approximately 11% of broadcasts sent to each device.

Table 1. Results of one hour trial processing data into storage (Ratio indicated the % of entries that succeeded in being reached).

Device	# of entries	# of exceptions	Ratio (%)	speed
Raspberry Pi	1,345	157	89.547	2.402s
Odroid-C4	594	72	89.189	5.499s

The discrepancy between devices is highlighted with the ability of the Raspberry Pi to process and store entries at a rate 228.9% faster than the Odroid-C4 device. This discrepancy was initially attributed to using a Bluetooth dongle for network connectivity on the Odroid-C4 which was believed to be resulting in a larger time delay between entries. Even running both devices for a shorter period of time and mirroring the timeout to 4 seconds, the Raspberry Pi device still averaged a speed of 4.3 seconds, and the Odroid-C4 device an average time of 6.4 seconds.

The initial results from a one hour trial in Table 1 were conducted using SQLite 3 and the default Python SQL library. When the testing was re-done using InfluxDB and the influx_db Python library it was far more optimized and the result was decreased from a 3 second difference to a 0.3 second difference in retrieval and storage speed. This may have been the result of SQLite 3 having a worse impact that affected the Ordoid-C4 device more than the Raspberry Pi.

The first assumption was that the Bluetooth connection was worse using the dongle on the Odroid-C4 device, but changing database software caused the discrepancy to be greatly reduced without requiring hardware changes. Unfortunately due to implementation differences the ratio of exceptions or missed entries is not available but instead the average, minimum, and maximum time to retrieve entries is available. As shown in the data results from Table 2, both devices had a minimum value close to 2.8s which is likely close to the maximum rate the Ruuvi beacon can broadcast, however the Raspberry Pi had a much higher maximum delay for its response time.

Table 2. Results of one week processing data into storage

Device	# of entries	min	max	average
Raspberry Pi	154,809	2.83s	7.6ms	3.09s
Odroid-C4	166,525	2.85s	6.42s	3.25s

Using the calculations that a week contains 604,800 seconds which when accounting for a 3.09s average time, should result in 195,728 entries being generated over a period of one week, likewise the Odroid should produce 186,092 entries with a 3.25s response time. Using this approximation we can make the assertion that the Raspberry Pi was far more likely to lose entries during long term testing, than the Odroid device. This is contrary to the results found during a short 1 hour initial testing period. As noted in a paper published regarding the topic of weather monitoring [10]. The ability for an IoT system to be low-cost while maintaining accuracy and reliability is one of the biggest challenges facing IoT systems and as such is why the exception rate of entries was chosen as a metric to be monitored.

4.2 Data Accuracy and Long Term Analysis

The second test being done takes into account the availability of most SQL libraries ability to handle median, mean, and other simple models of statistical analysis using built-in library functions. In InfluxDB the primary functions MEAN() and MEDIAN() will be used in queries and the time it takes to process will be compared to a custom function solely within Python to determine whether the mean and median are accurate, and whether the speed of the library or native Python code is faster. The data will be scaled on a range of 24 hours, 3 days, and 7 days, similar to those often found when checking the forecast allowing the larger datasets to test the machines processing power at varying scales.

While leaving the Odroid-C4 and Raspberry Pi devices running it quickly became evident that there were issues immediately. The Odroid-C4 device was slower at collecting entries, but the Raspberry Pi would crash unexpectedely. During the data collection trial the length was increased from 7 days to 10 days due to periods of time where the Raspberry Pi would crash and quit out from collecting data entries.Using Grafana allows the creation a dashboard for each device showing the retrieval time for entries based off of the datetime when they were entered.

Looking at the bottom graph in Figure 6. we can see that it contains large gaps where the Raspberry Pi device was no longer processing and storing entries. Even with monitoring, the scripts would be shown to be active but no results would be generated. While the Raspberry Pi does have a better processing performance as a result of having comparatively better hardware than the Odroid device, it should be noted that this level of instability and crashing while running identical code is a huge issue in the devices long term useability. Even though minimalistic and identical code was run on both devices with the same dependencies, the Raspberry Pi device was subject to much more instability issues and crashing.

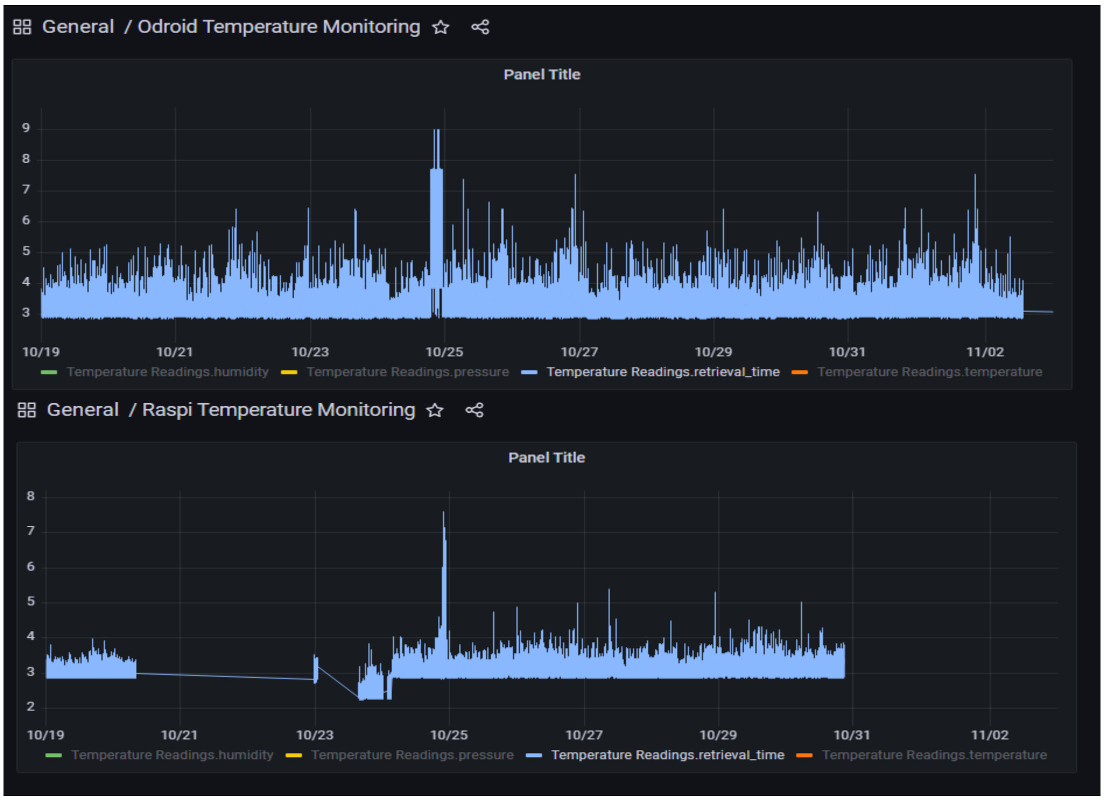

Figure 6. The comparative data retrieval issues between Raspberry Pi and Odroid Devices.

4.3 Daily Calculation Testing

The first test conducted selected for a full 24-hour time period within the dataset and checking the number of entries generated, the mean hourly values, and the median hourly values. The code seen in Figure 7. Is what was used to parse through the database for a selected length of time. Then operations are done comparing the time taken to manually calculate the median, median, and count and using the InfluxDB built-in libraries. It also checks to make sure that both the manually calculated value and the value produced by InfluxDB libraries are identical. To calculate this, the microcontroller would query the database for the desired measurement type (humidity, pressure, or temperature) and submit the results, this is then repeated for each desired measurement type.

```python
def query_mean():
    today = datetime.datetime.now()
    today = today - datetime.timedelta(days=1)
    listed_hours = []
    n = 0
    while (n < 24):
        start = today.replace(hour=n, minute=0, second=0)
        if n == 23:
            stop = stop + datetime.timedelta(days=1)
            stop = stop.replace(hour=0, minute=0, second=0)
        else:
            stop = today.replace(hour=n+1, minute=0, second=0)
        start = start.strftime("%Y-%m-%dT%H:%M:%SZ")
        stop = stop.strftime("%Y-%m-%dT%H:%M:%SZ")
        sql_query = f'SELECT MEAN({field}) FROM \"{table}\"  WHERE time
>= \'{start}\' AND time <= \'{stop}\''
        test = db.query(f'{sql_query}', database=db_name)
        values = test.get_points()
        for i in values:
            listed_hours.append(i['mean'])
        start = datetime.datetime.strptime(start, "%Y-%m-%dT%H:%M:%SZ")
        stop = datetime.datetime.strptime(stop, "%Y-%m-%dT%H:%M:%SZ")
        n += 1
    return listed_hours
```

Figure 7. Code snippet showing how the mean value for a 24-hour time period are retrieved from the database using built-in InfluxDB libraries.

The results are output to the command line and reformated manually into a more easily readable format. The outcome of this experiment was meant as a shorter term analysis of the devices within a smaller time window and the expected difference between the devices was expected to be within 500ms of each other. The reason for using a shorter time frame initially was to see if the devices showed any difference when run for a short time period compared to a longer time period. It was an important consideration due to the discrepancies found between short term and long term population of the database that was shown during the initial set-up and could potentially result in a discrepancy again.

As is displayed in Table 3, a a simple test was conducted to find how many entries were generated within the time period provided. The results were calculated both manually and using the SQL COUNT function which showed that the Raspberry Pi performed better in both metrics. The next step was to check if the results held true for all available metrics stored within the database, so the calculation was done for both devices on every data metric available to query.

Table 3. A comparison of an average 24 hour sample between devices.

Device	# of entries (hourly)	# of entries (total)	speed manual	db speed
Raspberry Pi	1,065	25,567	773ms	136ms
Odroid-C4	982	23,577	882ms	197ms

The code featured in Figure 8. shows how the values in Table 3 were calculated. Both devices requested automatic and manual queries in the functions “data_handler.query_manual()” and “data_handler.query_auto()” respectively. The length of the period and type of measurement being requested at was stored within those function as well. Then the program timed how long each operation took, checked to determine whether the results calculated automatically and manually matched. If the values were found to be identical, the results were posted to the command prompt and manually added to the table.

```
def data_test():
    time_start = time.time()
    manual = data_handler.query_manual()
    time_stop = time.time()
    time_result = time_stop - time_start
    print("Manual timing took: " + str(time_result))
    time_start = time.time()
    auto = data_handler.query_auto()
    time_stop = time.time()
    time_result = time_stop - time_start
    print("Auto timing took: " + str(time_result))
    for i in range(len(manual)):
        print("Does " + str(manual[i]) + " equal " + str(auto[i]) + "? " +
str(manual[i] == auto[i]))
```

Figure 8. The code snippet used to conduct a test for manual and automatic retrieval of database entries.

Upon repeating the operation for all data types the following data was found and collected below. Table 4 contains the information on the Raspberry Pi's performance across all data types: Counting the number of entries within the time period, finding the mean temperature, and determining what the median value was. Table 5. uses identical methods to those found in Table 4, but instead calculated on the Odroid device.

Table 4. Raspberry Pi hourly calculation speeds with and without database libraries.

	Raspberry Pi		
Measurement	Count	Mean	Median
Temperature (Manual)	787ms	754ms	817ms
Temperature (Database)	136ms	164ms	164ms
Pressure (Manual)	787ms	813ms	834ms
Pressure (Database)	148ms	170ms	164ms
Humidity (Manual)	797ms	769ms	786ms
Humidity (Database)	140ms	141ms	198ms

Table 5. Odroid C4 hourly calculation speeds with and without database libraries.

Odroid C4

Measurement	Count	Mean	Median
Temperature (Manual)	881ms	868ms	867ms
Temperature (Database)	194ms	192ms	236ms
Pressure (Manual)	898ms	881ms	909ms
Pressure (Database)	194ms	196ms	207ms
Humidity (Manual)	909ms	877ms	886ms
Humidity (Database)	193ms	193ms	249ms

Looking at the results found in Table 4 and Table 5, the result shown is the same result as the initial single 24-hour sample conducted in Table 3. The Raspberry Pi device performed better across every category when calculating results using the shorter time period. The range of how much faster the Raspberry Pi performed depends on the type of measurement being done, and was found to generally be between 20ms to 100ms faster at calculations than the Odroid device.

4.4 Weekly Calculation Testing

As shown in Figure 9, the time period is set as a number of days. In this case the number of days is set to 7 but for the 24-hour period the number of days is set to 1. It also looks specifically for temperature in this example, and the code is changed for each measurement being analyzed. The processes uses same methodology as presented in “Section 4.3: Daily Calculation Testing”, the only difference is that it instead queries a larger dataset with more entries spanning a week instead of a single day.

```
def query_manual():
    today = datetime.datetime.now()
    today = today - datetime.timedelta(days=7)
    listed_hours = []
    n = 0
    while (n < 24):
        start = today.replace(hour=n, minute=0, second=0)
        if n == 23:
            stop = stop + datetime.timedelta(days=1)
            stop = stop.replace(hour=0, minute=0, second=0)
        else:
            stop = today.replace(hour=n+1, minute=0, second=0)
        start = start.strftime("%Y-%m-%dT%H:%M:%SZ")
        stop = stop.strftime("%Y-%m-%dT%H:%M:%SZ")
        sql_query = f'SELECT {field} FROM \"{table}\"  WHERE time >= \'{start}\' 
AND time <= \'{stop}\''
        test = db.query(f'{sql_query}', database=db_name)
        values = test.get_points()
        count = 0
        total = 0
        for i in values:
            count += 1
            total += i['temperature']
        mean_temp = total/count
        listed_hours.append(mean_temp)
        start = datetime.datetime.strptime(start, "%Y-%m-%dT%H:%M:%SZ")
        stop = datetime.datetime.strptime(stop, "%Y-%m-%dT%H:%M:%SZ")
        n += 1
    return listed_hours
```

Figure 9. Code snippet showing how the manual value for a 7-day time period are retrieved from the database and calculated manually.

The reasoning behind conducting the seconda test was the evidence which showed the Raspberry Pi being less stable during longer-term time periods. This meant that while it might perform well on small datasets gathered over a shorted period of time, it might become less effective as the dataset size or time-period increases.

Similarly to how the daily testing was conducted, the results from a 7-day period were collected into tables identical to those seen above. Due to having approximately seven times the expected entries as the 24-hour dataset the expectation was that the Raspberry Pi would should be around 500ms faster than the Odroid when it came to calculations if the trend from the smaller dataset continues. Table 6 shows the results found when doing a simple request for all entries generated during a 7-day period.

Table 6. A comparison of an average 7-day sample between devices.

Device	# of entries (daily)	# of entries (total)	speed manual	db speed
Raspberry Pi	22,115	154,809	4168ms	105ms
Odroid-C4	23,789	166,525	5,360ms	119ms

As seen above (Table 6) the number of entries produced is reasonably close, although a lower number of entries is found for the Raspberry Pi device. Although both devices had a similar number of entries, the Rasperry Pi had a much faster computation speed when manually calculating the results of larger datasets even when taking into account the difference in how many entries were found. This indicates that the trend of the Raspberry Pi being the faster deviceshould continue even when using larger sets of data. However using the values generated from manual and automatic requests across all data types Table 7 and Table 8 show a different pattern than the expected results found in Table 6.

Table 7. Raspberry Pi weekly calculation speeds with and without database libraries.

	Raspberry Pi		
Measurement	Count	Mean	Median
Temperature (Manual)	4,218ms	4,147ms	4,165ms
Temperature (Database)	96ms	114ms	684ms
Pressure (Manual)	4,366ms	4,291ms	4,329ms
Pressure (Database)	97ms	129ms	679ms
Humidity (Manual)	4,231ms	4,114ms	4,238ms
Humidity (Database)	106ms	121ms	693ms

Table 8. Odroid C4 weekly calculation speeds with and without database libraries.

Odroid C4

Measurement	Count	Mean	Median
Temperature (Manual)	5,279ms	5,262ms	5,262ms
Temperature (Database)	114ms	111ms	532ms
Pressure (Manual)	5,463ms	5,429ms	5,455ms
Pressure (Database)	117ms	114ms	504ms
Humidity (Manual)	5,258ms	5,286ms	5,298ms
Humidity (Database)	119ms	118ms	436ms

Unlike the initial sample in Table 6 the evidence suggests that while the Raspberry Pi calculations done manually are still much faster than those of the Odroid, with manual calculations sometimes being finished over a full second faster. The Odroid device actually performs better on larger datasets when making use of the built-in Influx DB SQL commands. This suggest that there is either some hardware difference or perhaps a difference in how the Rasbian distribution interacts with InfluxDB than Armbian does for the Odroid device.

5 CONCLUSION

The goal of the project was to use a simple IoT deployment to determine how alternative microcontrollers would compared to the Raspberry Pi. This experiment was conducted using a Raspberry Pi and Odroid device connected to a Ruuvi Bluetooth beacon which collected weather data and stored it in an InfluxDB database. This data was then processed using manually created functions and with the baseline SQL commands available to InfluxDB, to calculate the performance of each device. The two main trials performed were measuring a smaller dataset which observed a 24-hour time period, and a larger dataset which observed a 7-day time period. Due to the better hardware found in the Raspberry Pi the expectation was that the Raspberry Pi would perform marginally better across all testing methods when compared to the Odroid device.

The experiments began with the Raspberry Pi showing that is was the superior device in initial limited testing, such as 1 hour trials, for both creating database entries and calculating values from those entries. Unlike what was expected, when observed over longer time-periods the Raspberry Pi began to display issues with keeping the programs running, timing out, and failing to reach the beacon to create entries. There was no physical change between the conditions that the Raspberry Pi and the Odroid device were operating in, and the devices also used the same code and set-up so there was nothing in the environment that contributed to it achieving poorer results over longer stretches of time. Indeed, without a second Raspberry Pi device to test, it is impossible to know whether it was this specific microcontroller with issues, or the platform as a whole.

The Raspberry Pi still performed much better at raw calculations even taking into account the observed instability and as a result would still be the recommended microcontroller for conducting some type of edge computing or when requiring heavier data processing. There was one area where the Odroid device outperformed the Raspberry Pi, which was doing computations with built-in SQL commands. The Odroid device also was not affected by instability and was much more stable during the periods it was set to monitor and record incoming data. This gives the Odroid device an advantage in IoT deployments which rely heavily on reliability and consistency. The end result is that while both devices are relatively similar for a vast majority of use cases, it is still worth noting that each device may have some advantages or disadvantages depending on what kind of IoT deployment is being created.

When reviewing the methodology for testing, all of the tests conducted were consistent and there was no issue with the quality of the data being produced or the reliability of the calculations being done. For future consideration an expansion of the IoT deployment to include the full range of measurements provided by the RuuviTag sensor, would allow for more diverse calculations to be done. The measurements could then be compared to real-time measurements conducted by local meteorological stations to determine the accuracy of the RuuviTag sensor. This type of more complex and detailed computation would be beneficial for comparing microcontroller devices because it allows for different areas of comparison, and larger computations which could indicate more areas in which devices differ.

Additionally, there is a concern with limitation of using only two microcontroller devices during experimentation. Since the scope of the project was limited to just the Raspberry Pi and Odroid, there may be other microcontroller devices that perform better in certain cases or outperform these devices entirely. This concern also includes the limitation of having only a single unit for each microcontroller. It is unknown whether the instability and InfluxDB limitations noted for the Raspberry Pi are a common problem with all Raspberry Pi devices, or whether it was specific to only the device used in these tests.

REFERENCES

1. N. S. Yamanoor and S. Yamanoor, "High quality, low cost education with the Raspberry Pi," 2017 IEEE Global Humanitarian Technology Conference (GHTC), 2017, pp. 1-5, doi: 10.1109/GHTC.2017.8239274.
2. R. Shete and S. Agrawal, "IoT based urban climate monitoring using Raspberry Pi," 2016 International Conference on Communication and Signal Processing (ICCSP), 2016, pp. 2008-2012, doi: 10.1109/ICCSP.2016.7754526.
3. D. K. Singh, H. Jerath and P. Raja, "Low Cost IoT Enabled Weather Station," 2020 International Conference on Computation, Automation and Knowledge Management (ICCAKM), 2020, pp. 31-37, doi: 10.1109/ICCAKM46823.2020.9051454.
4. ODROID, March 31st, 2020-last update, ODROID-C4 DATASHEET . Available: https://dn.odroid.com/S905X3/ODROID-C4/Docs/S905X3_Public_Datasheet_Hardkernel.pdf.
5. RASPBERRY PI, 2021-last update, RASPBERRY PI 4 MODEL B DATASHEET. Available: https://datasheets.raspberrypi.com/rpi4/raspberry-pi-4-product-brief.pdf
6. C. Manore, P. Manjunath and D. Larkin, "Performance of Single Board Computers for Vision Processing," 2021 IEEE 11th Annual Computing and Communication Workshop and Conference (CCWC), 2021, pp. 0883-0889, doi: 10.1109/CCWC51732.2021.9376035.
7. A. S. Vale-Cardoso, M. Geny Moreira, C. H. N. Martins and M. C. R. Leles, "IoT Embedded Computing Systems Performance Assessment: a Simple Method," 2019 IEEE 10th Annual Information Technology, Electronics and Mobile Communication Conference (IEMCON), 2019, pp. 0071-0075, doi: 10.1109/IEMCON.2019.8936161.
8. D. Eridani and E. D. Widianto, "Performance of Sensors Monitoring System using Raspberry Pi through MQTT Protocol," *2018 International Seminar on Research of Information Technology and Intelligent Systems (ISRITI)*, 2018, pp. 587-590, doi: 10.1109/ISRITI.2018.8864473.
9. S. Naveen and M. R. Kounte, "Key Technologies and challenges in IoT Edge Computing," 2019 Third International conference on I-SMAC (IoT in Social, Mobile, Analytics and Cloud) (I-SMAC), 2019, pp. 61-65, doi: 10.1109/I-SMAC47947.2019.9032541.
10. C. Vanmathi, R. Mangayarkarasi and J. Subalakshmi R., "Real Time Weather Monitoring using Internet of Things," 2020 International Conference on Emerging Trends in Information Technology and Engineering (ic-ETITE), 2020, pp. 1-6, doi: 10.1109/ic-ETITE47903.2020.348.