IoT-Based Smart Scale: Real-Time Weight Sensor Data Collection and Device Communication (March 2021)

Zhiyang Zhou

***Abstract*—This paper talks about the process of creating an Internet of Things (IoT) application - Smart Scale, which utilizes the use of weight sensors on the Raspberry Pi edge device and enables instant data communication between IoT devices. This application is built with Amazon Web Services (AWS), specifically AWS IoT and AWS Greengrass as the cloud service provider. Meanwhile locally we have the Raspberry Pi 3B model as the edge device with the latest Raspberry Pi OS installed that was released in 2021-01-11. One Load Cell is connected to the Raspberry Pi GPIO by using an HX711 Analogue-to-Digital Converter (ADC). The data collected on the edge device is stored locally on the Raspberry Pi device, which can be accessed by AWS Lambda functions when invoked in certain subscriptions. Such information is then sent to mobile devices by using Twilio Connector and Notification in the AWS Greengrass group. Users are able to respond to the events by sending Message Queuing Telemetry Transport (MQTT) messages to an AWS IoT topic to invoke the lambda function again. The project architecture, design strategies, implementation, challenges, and evaluation methods are discussed in this article.**

***Index Terms*—Internet of Things, Weight Measurement, Electrical Engineering, Force Sensors, Web Services, Edge Computing, Mobile Communication**

I. Introduction

SMART Scale is an IoT application that utilizes AWS and Raspberry Pi sensors to enable instant communication between devices. Real-time data collected on the sensors is stored on the local device and sent to mobile devices to notify the users.

A. Background

Smart Home has become more and more applicable to modern households. It aims to use edge devices and sensors to provide convenience and security to our life at home. Once the sensor data meets certain standards defined by the user, home automation can be activated such as curtain control, door lock change, light switch etc. There are generally a few design approaches for Smart Home setup. Since edge devices usually have a small physical size and a low-to-none computing power, they rely on additional platforms to carry out storage and calculation functionalities, whether they are on a local Operating System (OS) or a Cloud Service. Device connections can use various connection methods. The most common ones are Bluetooth and Local Area Networks (LAN). As for messaging protocols, MQTT and HTTP are among the most widely used in the current industry.

B. Motivation

Smart Scale was inspired by the concept of Smart Home to incorporate automated services into our life at home. It uses weight sensors to send users instant text notifications whenever there are weight changes on the scale. Meanwhile, the sensor data is stored on the local device. Such data and event information can be retrieved by users whenever the user requests on a device through MQTT on a particular IoT topic. Smart Scale is aimed to provide human-machine interactions in the current scope of the project, but it can be used in other applications if more features are added to the process of the weight change detection.

C. Design Overview

Smart Scale as an IoT application includes the following two major aspects in the architecture design: cloud and local. For the choice of the cloud services, AWS IoT and AWS Greengrass provide functions that are necessary for the implementation of the project. They can enable instant communication between devices, whenever a new item is placed on the scale. The weight is measured and stored on the Raspberry Pi, which is the local device that can collect and store sensor data. The information will be updated on AWS, which will then send the user a text message regarding the event via Twilio Notification. The user can interact with the application by guessing the weight and typing the number into a message on another device. This message is sent to a particular IoT topic, which has AWS Greengrass subscriptions associated to invoke the AWS Lambda function to calculate the weight different between the guess and the actual weight. The result is packaged into another message that is sent to the user via Twilio Notification.

This paper is to be submitted on March 15, 2021 as the final project of CSS 532: Internet of Things class at the University of Washington Bothell (UWB) under the instruction of Professor Yang Peng. The work was supported in part by the Computing and Software Systems Department at UWB.

Author Zhiyang Zhou is a graduate student in the Master of Science in Computer Science and Software Engineering program with the Computing and Software Systems Department of the University of Washington, Bothell, WA 98011 USA (e-mail: zyang21@uw.edu).

D. Contribution Summary

The author Zhiyang Zhou is the sole contributor to the project, including the architecture design, the software implementation, the embedded hardware setup, and the use of existing frameworks and libraries mentioned in this paper.

II. Related Work

There have been various studies done in the field of IoT working with different types of sensors. For example, in 2020 a group of researchers in South Korea carried out an experiment [1] of building a system to monitor air quality, specifically the Particulate Matter (PM_{10}), using IoT applications. This system consists of Smart-Air sensors to measure the air quality in the subway tunnels, IoT gateway and a cloud computing web server. They were able to collect data remotely and compute the differences of air pollution levels between different hours throughout a day.

Another research [2] looked into detecting car pollution in India based on AWS and IoT systems. They used IoT-based devices to collect data of Carbon Monoxide near cars where they can also track the GPS location to match with the results. The information is sent to AWS to analyze if there is a Carbon Monoxide leak for the specific vehicle. When the CO level is above the testing threshold, a message will be delivered to the user to notify about the incident.

In the book [3] published in 2020, Al-Turjman and Imran talked about the current opportunities and challenges of using IoT-based products in Smart Cities. The use of IoT was raised to a macro point of view to identify the pros and cons in a collective IoT system. It discussed the practical challenges when building, placing and connecting sensor devices around the city. One main issue was to automate the process of transferring data with unique identifiers to a central computing server to analyze the big data, while protecting the user privacy. It was also difficult to find a good balance between the cost of putting a device in use and the service quality as well as availability in return. It provided some insights in the process of designing and building Smart Scale.

The research paper [4] published by Sonsare in 2018 was particularly more relevant to Smart Scale, as it talked about the design and use of weighing systems for crates in agriculture. Essentially the weighing process starts from checking if the crate weight is above a certain threshold, which was 80% of the maximum load in their standard. Once it goes beyond that point, the microcontroller will send signals to the buzzers to remind the user about the weight information. When the weight exceeds 95% capacity, a long beep along with LED light switch will be activated as a warning message. This weight system setup used a load cell, an HX711 module and LCD display screen to show the weight number. All the data is sent to cloud services, which is then processed and retrieved back to the local devices.

We can see that there has been a significant increase in the use of IoT-based applications around the world in many types of fields. Visual, weight, air and many other kinds of sensors are used to collect data for the automation process that provides convenience to better serve human life.

III. Project Description

This section gives a brief description of the system model about Smart Scale, the problem statement, the analysis on key system modules.

A. Problem Statement

As most modern households now have scales that can measure body weight, they tend to have a built-in display that shows the weight as numbers. They provide one function as a single unit - weight measurement. For body-weight scales, the common ones have a minimum weight threshold of 3 kilograms in order to activate the scale and can usually hold up to 200 kilograms. Other scales can sometimes be seen in the kitchen to measure objects that are lighter in weight and require higher accuracies, such as getting specific grams of cooking ingredients. However, most of these scales are do not provide any more functionalities. Once the object is lifted off from the scale, the readings tend to disappear immediately or a few seconds after. There are no ways for these devices to recording historic readings for future reference.

Furthermore, these weight scales cannot be used for force detection. There are no effective communication tools for them to notify people in the house regarding events where there are sudden weight changes. For example, if intruders come close to a house and step on your doormat, or if rodents find their way in through a small hole, these weight scales wouldn't been able to provide any meaningful functions to prevent these incidents from getting worse.

B. Problem Analysis

Based on the issues listed in the previous section, we can see that often times modern homes lack the necessary tools to accurately measure weights that have a wide range of threshold. It would be very convenient to have one scale that can detect light-weight objects that are as low as a few grams. Ideally it should also be able to have a high maximum load limit for heavy-weight objects such as a human body, while having a high accuracy no matter what the weight limit is. For example, if the load limit is 200 kilograms, the scale should be accurate to at least 1 decimal point, which is 0.1 kilograms in this case.

In order to make the scale also useful in the force detection scenario, we need to build and establish its connection to the internet, or to devices that can have access to the internet. Many devices we use that don't have much computing power or storage space are often connected to a device where there is a Graphical User Interface (GUI), enough storage space, and the ability to connect to the internet. Bluetooth and Local Area Network (LAN) are two of the most common connection methods for the space within a home. In this way, data collected at the sensor is able to be transferred to a device and saved locally or online. We can also use software libraries and programming languages to process the data, and direct event logics based on customized conditions that are suited to each scenario. The most important element about event detection is to have the ability to notify the homeowners in a timely manner about these defined conditions. Common scenarios with weight sensors include the measurement of weight, the detection of weight load reaching a maximum number, or the entering of unwanted objects and people on a surface.

The solution to these issues is to essentially have an accurate weight sensor, an efficient data storage and computing method, and a secure connection to the cloud service provider on the internet. There are mainly 2 considerations for how data should be handled. The first way is to handle the data locally. All the raw input is analyzed, processed and stored on the local device. Only the result is then uploaded to the cloud for further processing. This has a lower requirement on the internet connection and speed, meanwhile it needs to make sure the local device is able to process input instructions fast enough and has enough storage space. The second way is to use the local device as an IoT Gateway, which collects all the data from the sensors and uploads everything to the cloud. This can significantly reduce the requirements on the performance of the local machine. On the other hand, additional cloud services may be needed to meet the product requirements. For example, we can use AWS Lambda functions to process the data, and store raw and processed data in AWS S3. In the prototype design, I have decided to choose the first design method to handle all sensor inputs, as it is capable of running all of the functionalities.

C. *Key System Modules*

To have all of these elements included in the same package, I propose Smart Scale. It is an IoT-based weight sensor system that can accurately measures weight, reports if the weight is reaching a certain limit defined by the user and sends notification to the user regarding certain force detection. The weight sensor consists of a Load Cell and an HX711 module that acts as an Analogue-to-Digital Converter (ADC). They are connected to a Raspberry Pi 3B to provide the weight data. Next, we have AWS as a cloud service provider to handle the interactions with the user. This design approach is to put a lot more focus on the local side compared to the cloud side. A more detailed breakdown of the mathematical comparisons between the system requirements and the system specification is listed in the next sub-section.

D. *Math and Assumptions*

Each data input comes in the size of a few bytes and it is collected every 2 to 5 seconds. The Raspberry Pi uses a 16 Gigabytes (GB) micro-SD card, and with the Raspberry PI OS installed it still has at least 4 GB left. The Raspberry Pi 3B model used in the prototype has a Quad Core 1.2GHz Broadcom BCM2837 64bit CPU [6], which is sufficient to handle infrequent input instructions. With a 6 Megabytes per second (MB/s) upload speed and 100 MB/s download speed, there is usually only 6 millisecond (ms) latency during the data transmission process.

The Load Cell selected in the prototype has a maximum load limit of 20 kg. I programmed the device so that a weight is considered valid if the weight is at least 5 grams. Additionally, a weight change is considered valid when it is at least 10% different from the previous reading. For example, if the last reading is 100 grams, the weight detection happens if the next reading is more than 110 grams, or less than 90 grams but still above 5 grams. This is to avoid the random noises that are too close to 0 grams after the device calibration, meanwhile giving it a reasonable time for a new object to sit and stabilize on the scale before a valid reading is taken.

Device calibration is done by importing the HX711 software library [5] and having an iPhone 12 Pro as the base model for the weight, which is 189 grams according to the official Apple website. The experiments are conducted in the room temperature (25 Celsius degrees).

IV. Design and Implementation

Smart Scale provides a solution to the problems stated in Section III. The prototype consists of three major components: local, cloud, and edge devices as shown in Fig. 1.

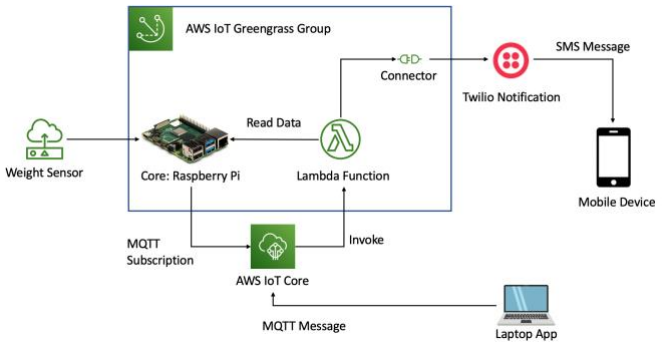

Fig. 1. System Architecture of the Smart Scale prototype. This shows the design layout and the relations between local, cloud and edge devices.

A. *System Design*

In order to measure the weight, we need to have a Load Cell, an HX711 module, and a Raspberry Pi device.

1) *Load Cell*

When an item is placed on the Load Cell, it has to be exerting force solely on one side from above. In the meantime, we need a base that can support all the weight on the other side from below. By having this structure, we have forces exerted on both sides of the Load Cell [7] from the opposite directions. This creates small changes in the resistance in the circuit, causing the variation of the current and voltage that is used as electrical signals in this setup. The design of the weight scale is shown in Fig. 2.

Fig. 2. Weight scale design with supporting platforms on 2 ends of a Load Cell from the opposite direction.

2) HX711 Module

The electrical signals are then passed to the HX711 module [8], shown in Fig. 3. It is an ADC that converts the analogue signal to digital signals, which can be better measured and standardized for the Raspberry Pi. There are specific wiring requirements for the connection of the circuit.

The red wire and the black wire from the Load Cell are connected to the E+ and E- ports on the HX711, respectively. This provides the electrical power to the Load Cell as an Excitation. The white wire and the green wire are connected to the A+ and A- ports, respectively, which is used to create Signal Amplifier. On the other side of the HX711 board, the GND (Ground) and VCC (Voltage Common Collector) ports are connected to Pin 6 (GND) and Pin 2 (5V), respectively, on the Raspberry Pi GPIO (General Purpose Input/Output) board. The DT and SCK ports are connected to Pin 29 and Pin 31, respectively, on the Raspberry Pi GPIO board. These two pins are associated with GPIO channel 5 and 6 when we access the input sources later.

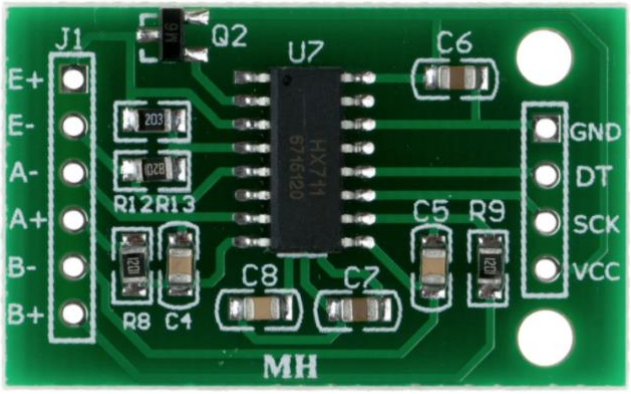

Fig. 3. Example of an HX711 Module.

3) Raspberry Pi

When the signals are sent to the Raspberry Pi as input data, we can use the HX711 software library [5] to calibrate and measure weights. To access the GPIO Pin 29 and Pin 31, we specify port numbers 5 and 6 in the script to see the weight number shown on the screen once the script is running. To calibrate the weight, place an item with a known weight to compare with the number shown. This is to set the reference unit, which is calculated by dividing the number by the actual item weight in grams. When calibration is finished and we run the script again, the number shown on the screen should be near zero, which will change to the actual weight once we place a new item on top. Fig. 5 shows the overall design of this process.

Since Raspberry Pi has the ability and the capacity to store data locally, we save the input data on the device after each valid reading. The source directory path is set to be */dest/LRAtest/test* to store the inputs. This will be accessible when we retrieve it later with the AWS Lambda function.

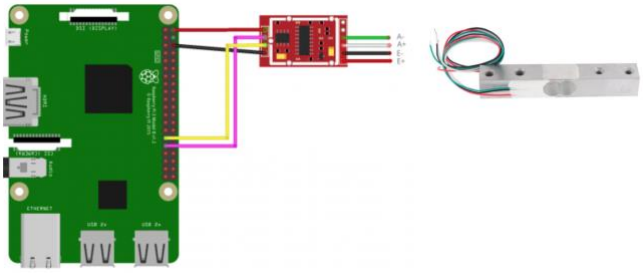

Fig. 4. Connection between Load Cell, HX711 and Raspberry Pi.

4) AWS

Smart Scale uses AWS as the cloud service provider. The role of AWS is to retrieve the data from the Raspberry Pi, process it on the cloud, and send out relevant information to other devices connected in the network. As an IoT-based product, Smart Scale relies heavily on AWS for remote data transmission and user interaction. Various services are selected in the implementation of this project, including AWS IoT, AWS Greengrass, AWS Lambda, and Twilio Notification via connectors. During this process, the local data is brought online for instant communication between devices when certain pre-defined conditions trigger the Lambda function, essentially the core of making any system “smart”.

B. Implementation Details

This section talks about the implementation details in the actual production of the Smart Scale prototype. They are broken down into the next few sub-categories below.

1) Weight Calibration

Place an iPhone 12 Pro on the scaling platform. Once the HX711 example script is run on the Raspberry Pi, the numbers printed on the screen fluctuate between -24570 and -24560. Since the Apple iPhone 12 Pro manual specifies that the phone body weight is 189 grams, we divide the numbers by 189 to calculate the reference unit, which is around -130. By setting this unit, during the next time the script is run, the initial weight readings are close to 0, indicating there is nothing on the platform.

2) Raspberry Pi Environment Configuration

To make Raspberry Pi compatible with our AWS functionality, we need to set up the local environment in the Raspberry Pi system. Then download, install and configure the Greengrass Core Python SDK software on the device. Download the root CA certificate and core device certificate and keys to authentic the Raspberry Pi as the core device when interacting with AWS. When the environment is configured, start the Greengrass daemon process on the Raspberry Pi.

Since we will use AWS Lambda function to access the local resources on the Raspberry Pi, we need to specify a system directory path and change the file permission to be accessible, readable, and writable.

3) Local Script

This is running in the full duration of when the product is

active. Essentially it checks every two seconds if the weight on the scale is valid. It first opens up the file at */dest/LRAtest/test* to check the previous weight reading, which is used to compare with the current weight reading. If the current weight is more than 10% different from the previous, it will update the local file content and send an MQTT message to an IoT topic with information including the reading. This is to invoke the Lambda function regarding the event of a valid weight change.

4) AWS Setup

In the AWS IAM (Identity and Access Management) console, create a Greengrass role with a policy allowing the basic AWS IoT "Publish" and "Subscribe" access permission. Attach "AWSGreengrassResourceAccessRolePolicy" to the same role as well. In the AWS IoT console, create a Greengrass group and set up the core device for the Raspberry Pi.

In the AWS Lambda console, create a new function and upload the Python script along with the Greengrass SDK package. The Greengrass SDK allows us to use the "iot-data" client when publishing custom MQTT messages to a specific IoT topic. Associate this Lambda function to the Greengrass group and creates a local volume resource by specifying the source directory as */dest/LRAtest/test* and granting access permissions. This way when the Lambda function is deployed, it will be able to access the local files. In the Greengrass Subscription panel, set up a subscription from IoT Cloud to Lambda, which serves as an invoke signal to call the Lambda function when certain conditions are met.

In order to send user a notification each time a valid event happens, I set up the Twilio notification service. First create a Secret resource in the Greengrass console to authenticate the tokens associated with my Twilio trial number. Next, set up a Connector for the Twilio Notifications so that we can use the service. Finally, establish a new subscription from the Lambda function to the Twilio Connector. This sends a text message from my Twilio trial number to a verified phone number and notifies about the event.

5) Lambda Function

Since the Lambda function is deployed to the core device and has access to the local file containing the previous weight information, it checks a key-value pair in the JSON-formatted messages received and determines the type of the input message. This can create two kinds of outputs.

If the input message indicates that a new item is recently placed on the Smart Scale, send the user a prompt that asks them to make a guess on the weight with a positive number. Otherwise, if the input message indicates the user already made a guess, it would retrieve the actual weight information stored on the local file at */dest/LRAtest/test* and calculate the difference. The result will also be sent to the user as a notification informing if the guess is greater than or less than the actual weight.

The Lambda function is invoked whenever a new MQTT message is received at the specified topic, due to the subscription we set earlier in the Greengrass console.

6) User Interaction

On another device such as a laptop, after it is configured for AWS IoT, we can send MQTT messages to the IoT topic to invoke the Lambda function. Note the messages sent from here should indicate that the user is making a guess, to differentiate from the weight change event. The script is run only once each time the user types a specific command in the terminal and remains idle until user gives the next one.

V. Evaluation

The majority of the evaluation is done around the weight measurement process. The accuracy of the scale, the responsiveness of the detection, and the cost of the application all contribute to the crucial components of the Smart Scale evaluation.

A. Accuracy

Even though most household items label the Net Weight, it is the weight of only all the content inside the package. Thus, the actual weight of the common household items is heavier than the labeled Net Weight. Because of the fact that objects with precisely known weights are not easily accessible, I used iPhone 12 Pro as the base model for weight calibration. According to the official Apple website, the body of an iPhone 12 Pro is 189 grams.

Name	Actual Weight (g)	Measured Weight (g)	Error Percentage
iPhone 12 Pro [10]	189	186.2	-1.5%
Amazon Echo (Gen 1) [11]	1064	1028.6	-3.3%
iPad (Wi-Fi) (Gen 8) [9]	490	469.3	-4.2%
AirPods Pro [12]	51	54.9	+7.6%
AirPods Pro Charging Case	45.6	45.8	+0.4%

Fig. 5. This table shows the difference between actual weight and the measured weight.

As we can see from Fig. 5, the measured weight fluctuates around the actual weight. Thus, it is of better practice to take the absolute values of all the Error Percentages before calculating the mean. This can better reflect the absolute accuracy between the two quantities, which yields an average Error Percentage of 3.4%. There are other factors that can contribute to the discrepancy, such as the electric noises, items with varying weights during production, the temperature of the measurement environment.

B. Responsiveness

Smart Scale is set to take a reading of the weight every 2 seconds and checks if the reading is valid. This can unnecessarily increase the cost of Twilio Notification messages and give the user wrong information about whether a new item is placed, if there are no additional restraints on the condition check. I defined a reading to be valid when it meets both of the following conditions: If the weight is at least 5

grams and if the current weight is more than 10% different from the previous reading, whether it's higher or lower.

When the same item is placed on the scale, the readings can still fluctuate because of random electric noises. Having defined a 10% threshold significantly improves the success rate of identifying the same object. Setting the minimum detection weight as 5 grams avoids reading the randomly fluctuating signals around 0 when no items are placed on the scale. Each time a valid reading is taken, the device goes to sleep for an additional 3 seconds, giving the new object a reasonable time to fully stabilize to a still state.

Reading Condition	Correct Responses	Incorrect Responses	Error Percentage
No Items	10	0	0
One New Item Placed	10	0	0
Replacing Item A with Item B	9	1	10%
Same Item Sitting Still	10	0	0
Item Lighter Than Minimum Weight Placed	10	0	0

Fig. 6. The experiment results of when different events happen on the scale.

It is notable that in Fig. 6, the device is programmed to make the correct output in a total of 50 experiments under various conditions. The only time it had an incorrect response was when a new item replaced the old item on the scale, however, with a weight difference smaller than the threshold. In theory, the device is expected to send out a new notification because a different item is added. Smart Scale failed to detect as the weight difference was small enough to be ignored according to the settings.

C. *Cost of Use*

For the prototype model of Smart Scale, I used a Twilio Trial Account that was automatically granted a $15.5 credit. Applying for a new Twilio Phone Number costs $1 as the initial fee, and each additional text message sent from the Twilio Phone Number costs $0.0035. This yields a total number of 3853 messages to send for free. Beyond that it will require additional payments to user their notification services.

VI. Future work

AWS provide other IoT services that can be used to enhance the functionalities of Smart Scale. For example, instead of calculating the weight difference for new item replacement on the scale, we can plug in a camera to the Raspberry Pi and analyze the image for detection. By analyzing the image using Amazon SageMaker, a Machine Learning service on AWS, we can detection motion or even predict item identification. With this feature, the detection can be more reliable when the item weight difference is small. Images of the item can be stored on AWS S3, which can be sent in the user notification message when a valid reading is established.

During my experiments in the design process, I found out that since AWS Greengrass SDK package is uploaded together with the Lambda function, it poses restrictions on the types of services that can be used at runtime. There are only 4 types of clients available for Greengrass deployed Lambda functions: "lambda", "iot-data", "secretsmanager", "streammanager". Initially I planned to save weight readings and images by importing the Python "boto3" library, which can be used to access AWS S3 storage place. It can allow us to create, write, read files on the cloud, which can reduce overhead on the use of local resources. However, with the structure of AWS Greengrass, I was unable to use AWS S3 in the Lambda function. For future work when we want to use other AWS services, it is possible to have to use a different architecture instead of Greengrass.

There are also other types of Load Cells that can replace the current one in the system that only has a maximum load limit of 20 kilograms. By creating the correct circuit, it can be viable to include multiple pieces of Load Cells under the scale that can measure up to 200 kilograms. The size of the Load Cells is also significantly smaller, making the product viable when there's limited space for placement.

In most modern software applications, there is almost always a Graphical User Interface (GUI) for users to interact with the system. However, in the current prototype of Smart Scale, the user has to rely on using the shell scripting to send messages to the AWS IoT cloud. Future work can be done on the creation of IoT-based mobile applications for users to directly view, edit, and send messages.

VII. Conclusion

Smart Scale is an IoT-based system that enables users to obtain real-time data from the weight sensors connected to the edge devices. It can be applicable in various use conditions to provide convenience to our life at home. The prototype is a good example that shows how we can benefit from Smart Home in creative ways. Despite facing technical challenges in the design and implementation process, Smart Scale can be further improved by completing the future work.

References

[1] Jo JH, Jo B, Kim JH, Choi I. "Implementation of IoT-Based Air Quality Monitoring System for Investigating Particulate Matter (PM10) in Subway Tunnels." International Journal of Environmental Research and Public Health. 2020; 17(15):5429. https://doi.org/10.3390/ijerph17155429

[2] A. Bhatnagar, V. Sharma and G. Raj, "IoT based Car Pollution Detection Using AWS," 2018 International Conference on Advances in Computing and Communication Engineering (ICACCE), Paris, France, 2018, pp. 306-311, doi: 10.1109/ICACCE.2018.8441730.

[3] AI-Turjman, F., & Imran, Muhammad. "IoT Technologies in Smart Cities From sensors to big data, security and trust (Control, Robotics & Sensors)". Stevenage: IET, 2020. [Online]. Available: https://alliance-primo.hosted.exlibrisgroup.com/permalink/f/kjtuig/CP71339476940001451

[4] Sonsare, Pravin. "IOT based Smart weighing system for Crate

in Agriculture". International Journal of Computer Sciences and Engineering, Jan. 2018. 6. 336-341, doi: 10.26438/ijcse/v6i1.336341.

[5] Tatobari. (2021). "HX711 for Raspbery Py". GitHub repository. [Online]. Available: https://github.com/tatobari/hx711py

[6] Raspberry Pi 3 Model B. Specification. [Online]. Available: https://www.raspberrypi.org/products/raspberry-pi-3-model-b/

[7] Sarah Al-Mutlaq. "Getting Started with Load Cells". [Online]. Available: https://learn.sparkfun.com/tutorials/getting-started-with-load-cells/all#

[8] AVIA Semiconductor. "24-Bit Analog-to-Digital Converter (ADC) for Weigh Scales". [Online]. Available: https://cdn.sparkfun.com/datasheets/Sensors/ForceFlex/hx711_english.pdf

[9] Apple. iPad Tech Specs. [Online]. Available: https://www.apple.com/ipad-10.2/specs/

[10] Apple. iPhone 12 Pro Tech Specs. [Online]. Available: https://www.apple.com/iphone-12-pro/specs/

[11] Amazon. Amazon Echo Gen 1 Tech Specs. [Online]. Available: https://www.amazon.com/dp/B00X4WHP5E/ref=twister_B01KIOU214?_encoding=UTF8&psc=1

[12] Apple. AirPods Pro Tech Specs. [Online]. Available: https://www.apple.com/airpods-pro/specs/

Zhiyang Zhou received the B.S. (2018) in physics and astronomy from the University of Washington, Seattle, WA. He received a Graduate Certificate (2020) in Computer Science from Seattle University, Seattle, WA. He is currently pursuing the M.S. degree in computer science and software engineering at the University of Washington, Bothell, WA, with an expected graduation date of June 2022.