AY 2022/2023 Semester 1
CS3237 Introduction to Internet of Things
Project Report
Title: WeMos VenD1ng

Group 1

Kom Xing Yuan - A0217957N

Phua Jun Heng - A0222415U

Poon Chuan An - A0210750W

Mayank Panjiyara - A0221571N

Abstract

According to a study by Mckinsey [1], the 2030 economic value of Internet of Things (IoT) adoption is estimated to range between $650 billion and $1,150 billion in the retail environment. We see huge potential in this field, and our project specifically targets this area in the form of a smart vending machine as a complete IoT system. This report will focus on the detailed implementation of the entire system, highlighting challenges faced and solutions. The video link is available here, and the source code is available here, which is specifically implemented to be as portable and modular as possible. We hope that our report and code base would provide valuable insights and benefit future readers doing similar work.

Introduction

Project motivation

The business of vending machines is known to be very scalable compared to conventional businesses. However, a traditional vending machine is bulky and weighs at least 180kg [2], with a minimum cost of $2000 [3]. We hope to tap into this market not just by joining as players, but also produce our own vending machine model.

Problem statement

An obvious drawback of traditional vending machines is that they are fixed in place, due to its bulky nature and high energy requirements. The classic coke vending machine has a high energy consumption of at least 3.8kW/h [4], requiring it to be plugged into a power source at all times. It is then unable to react to variable footfall demand.

Proposed solution

Our solution involves building a portable, light and comparatively low power smart vending machine that accepts coins as our main mode of payment, using computer vision to verify. This ensures that it can respond to variable footfall demand and can be placed anywhere with an internet connection. It also logs transaction data which can be used for long term analytics. The overall architecture diagram is shown below.

IoT Architecture

Figure 1: IoT Architecture

An oversimplified explanation of the entire system is as follows. The user will interact with the vending machine and insert coins. The image of the coins will be sent and processed on the cloud to determine its authenticity. When the transaction is completed, a snack would be dispensed. The user also has an option of paying via the telegram bot. In that case, the vending machine would just dispense the snack via telegram bot command through the cloud. More details will be elaborated in the corresponding sections.

The entire tech stack we used is as follows. Database: MongoDB Atlas; Cloud: Google Cloud Platform (GCP), Amazon Web Services (AWS); Communication protocols: UART, I2C, MQTT; Languages: C++, Python, Arduino; Other tools: Telegram API, Scikit learn, OpenCV.

Hardware

3 WeMos D1 and 1 Raspberry Pi 3B board is used. WeMos D1 R1 is a discontinued Wi-Fi board based on ESP8266-12E, advertised to come with 11 GPIO pins.

Due to the lack of digital pins output, we partitioned the 3 WeMos boards for modularity. The “master” WeMos is used to receive user inputs, communicate with the server and give commands to the 2 other “slave” WeMos. The 2 “slave” WeMos are used solely for acting on received commands.

The “master” WeMos is triggered from the interrupts from either one of the 4 buttons(return, jackpot, snack1, snack2 button), knock sensor or mercury tilt switch. The knock sensor serves the purpose of telling the server when to trigger the camera via coin sliding down, while the mercury tilt switch acts as a security mechanism, detecting if anyone is tilting the machine beyond a certain angle. Its only output is when the server commands it to turn on the white LED to provide camera lighting.
(note that figure shown is an Arduino Uno, while the actual board is a WeMosD1)

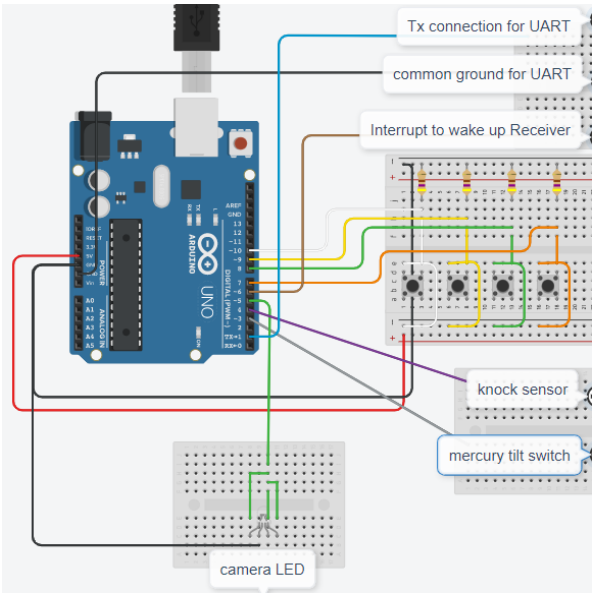


Figure 2: Wemos master pinout

The 2 “slaves” are used for output purposes. Slave 1 controls the drives the 3 servo motors, 2 DC motor via a L293D driver chip [5] and buzzer, while slave 2 controls I2C LCD to display messages for user interface.

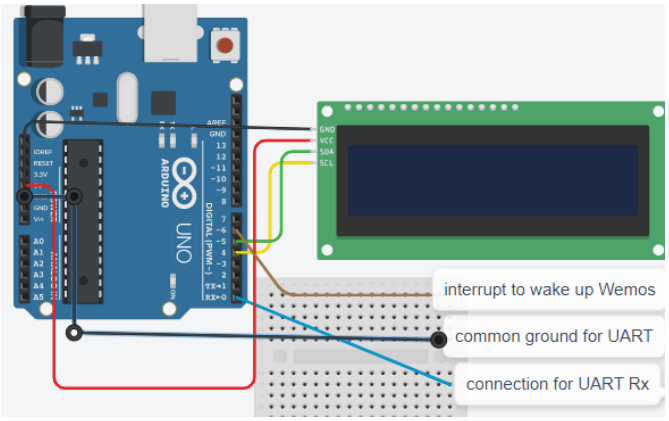


Figure 3: Wemos slave 2 pinout

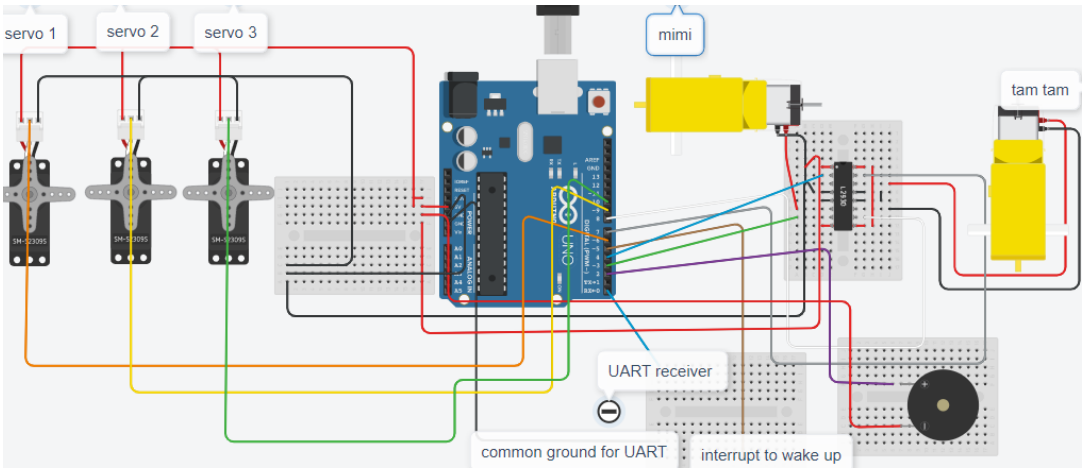


Figure 4: Wemos slave 1 pinout

The 3 servo motors allows for coins to be kept in a holding area for return midway the transaction, and each motor drives the coil for each snack.

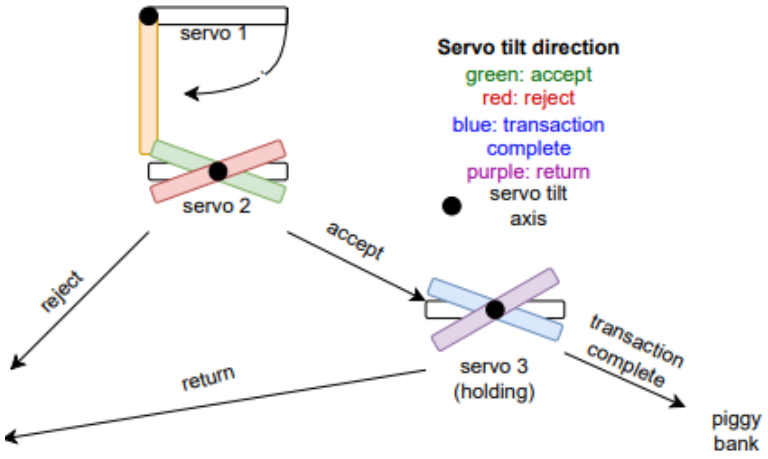


Figure 5: Side view of servo motor subsystem

The Raspberry Pi is only used because of the camera module, connected via a CSI connector. The Raspberry Pi camera v1.3 [6] is a low cost camera with a still picture resolution of 2592 x 1944, allowing us to take clear pictures. In our machine, we fixed the camera in place with a white LED to provide adequate lighting. The focal length of the camera was manually adjusted for the best results. We also coated our “photo booth” with black paper as it experimentally gave us the best picture quality.

Figure 6: Image of photo taken by camera

A flowchart of our entire hardware subsystem is shown below for completeness.

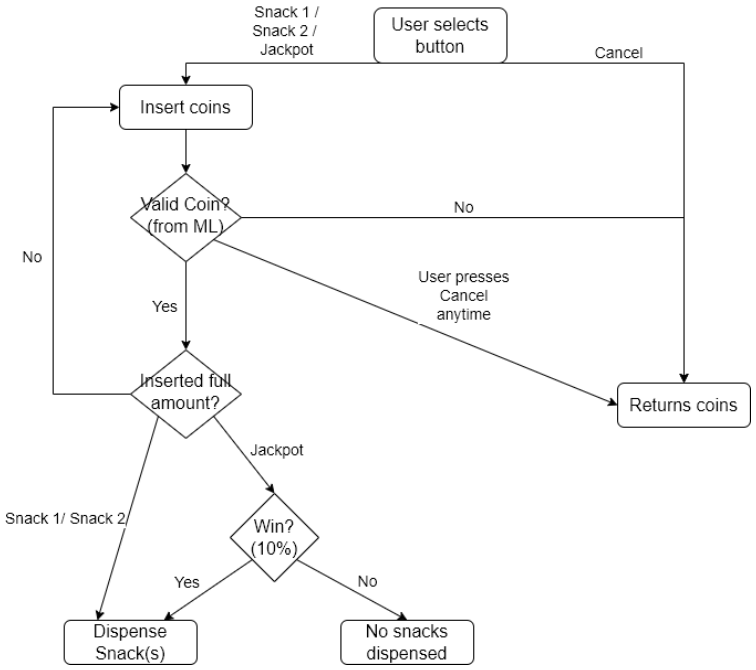


Figure 7: Flowchart for hardware

Insights

We experienced problems using WeMos pin D3 and D4 as interrupts, which is well documented. Our recommendation is that one should just skip using D3 and D4 as interrupts. On a side note, L293D driver motors are a useful tool in supplying voltages up to 36V if needed. One should also consider using tinkercad and coloured wiring whenever wiring becomes nontrivial, as it allows for team members to visualise the wiring clearly. Lastly, the mislabelling of some 37 in 1 sensors forced us to enhance our technical hardware skills via extensive usage of a multimeter, and we should always double check instead of solely relying on the datasheet.

Power

Given that we use 3 WeMos and 1 Raspberry Pi 3B, our power consumption would be higher than a typical Iot system. However, if we were to compare ourselves to a typical vending machine, our power consumption is reasonable.

We have implemented forced light sleep mode on the 2 “slave” WeMos, that is woken up by a GPIO interrupt from the “master” WeMos when needed.However, the “master” WeMos is always active.

```
//GPIO wake up
gpio_pin_wakeup_enable(GPIO_ID_PIN(LIGHT_WAKE_PIN), GPIO_PIN_INTR_LOLEVEL);
wifi_set_opmode(NULL_MODE);
//sleep mode
wifi_fpm_set_sleep_type(LIGHT_SLEEP_T);
wifi_fpm_open();
wifi_fpm_set_wakeup_cb(callback);
wifi_fpm_do_sleep(FPM_SLEEP_MAX_TIME);
delay(1000);

//insert code between here and
if(startTriggered == true){
  Serial.println("pressed");
  startTriggered = false;
}
Serial.println("Exit light sleep mode");
//here
wifi_set_sleep_type(NONE_SLEEP_T);
delay(1000);  //  Put the esp to sleep
```

Figure 8: Code snippet for WeMos light sleep mode

This brings our total power consumption of all 3 WeMos from 510mA to 171.8mA most of the time. We expect our machine to be operating at 171.8mA more than 90% of the time.

On the Raspberry Pi side, we disabled the USB controller via *sudo tee /sys/bus/usb/drivers/usb/unbind* command which reduced 200mA and HDMI via *sudo /opt/vc/bin/tvservice -o* command which reduced 20mA and ran the Pi headless. This brought down our power consumption from 550 mA at 5V to 335 mA at 5V. However, a limitation is that the Raspberry Pi is always on active mode.

The powerbank we used has a capacity of 30000mAh rated at 65W [7]. With power saving features implemented, our battery life increases from 28.3hours to 59.2hours theoretically. As a result of parasitic capacitance and energy leakage, our experiments yielded a worse battery expectancy ranging from 40 to 50 hours. This may be due to fluctuating usage of the vending machine during testing.

Moreover, we have implemented an energy harvester in the form of a solar panel, which is capable of supplying a 15W maximum power under 5V [8]. Our solar cells also have an energy efficiency of 19%. If we think of our battery as an analogy of a bucket full of water, the water level is currency dropping via consumption and will be empty eventually. By energy harvesting of ambient energy in the form of solar panels, we are also topping up the water level concurrently.

Figure 9: Solar panels

Insights

Power analysis is nontrivial and experimentation takes a long time, resulting in low test samples and a large range of battery expectancy. We realised that we take power for granted most of the time, and noted its significance in building a successful IoT system. Although energy harvesters can supplement our battery life expectancy, it is not 100% reliable due to its fluctuating power supply. Hence, power saving measures on the software side is key. A combination of both would yield the best results.

Communication

WeMos - WeMos

Within the 3 WeMos, we implemented a unidirectional form of UART, where only the “master” WeMos transmits while the 2 “slave” WeMos receives. This results in more simplistic wiring and multiplexing is done within the software of the 2 “slaves” itself. This means that while both “slaves” would receive the message, the message will only be meaningful for one of the “slaves”, as the specific case code is exclusive. Since both “slaves” are in sleep mode, there is an Output pin from the “master” to trigger a wake up via external GPIO interrupt before any message is sent. The obvious disadvantage is the higher power consumption, but we feel the benefits are greater in more simplistic wiring and software multiplexing provides greater scalability. Moreover, it introduces multitasking, as each board can perform tasks simultaneously.

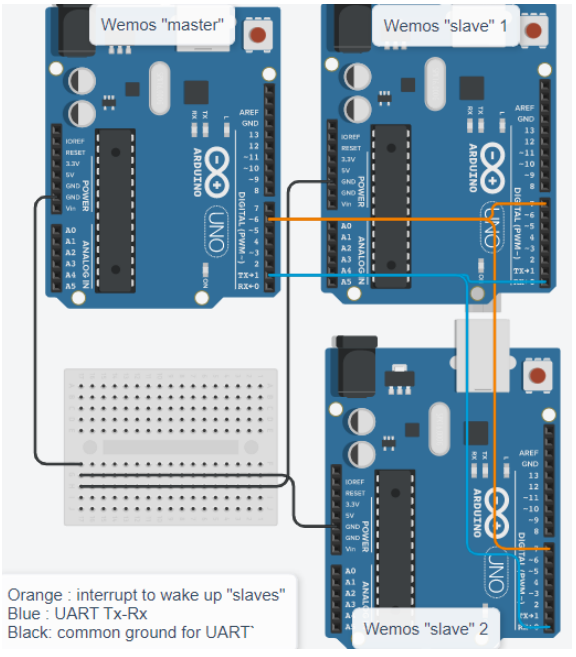


Figure 10: UART wiring

MQTT

MQTT (Message Queuing Telemetry Transport) was used for inter-communication between the WeMos, Cloud Server, Telegram Bot and Raspberry PI. MQTT uses a publish and subscribe methodology where a subscriber can listen for messages from a topic and the publisher can send messages to the topic. The broker receives all the messages and routes it to the specific nodes.

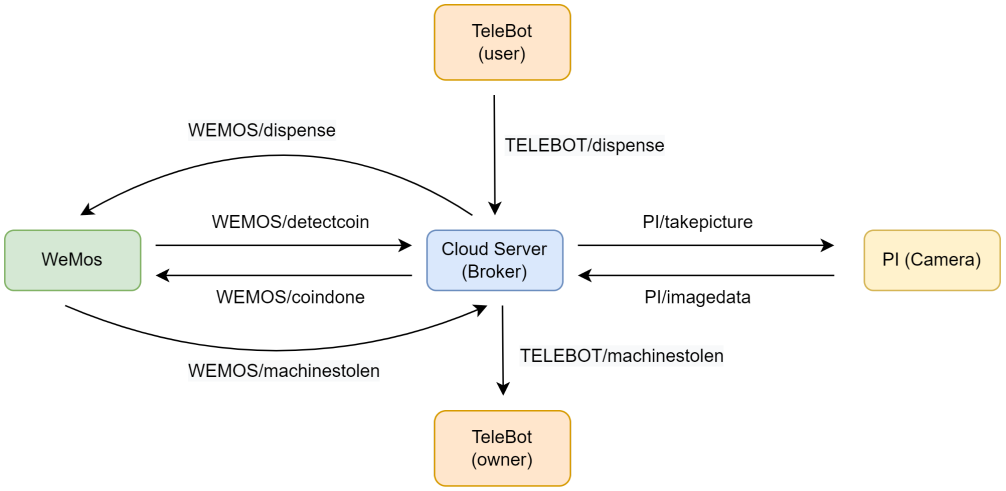


Figure 11: MQTT Flow for Device Intercommunication

Above is the flow chart for our MQTT flow, outward arrows indicate publishing to a topic and incoming arrows indicate subscribing to a topic.

WeMos - Server

As seen above, we made the WeMos subscribe to the WEMOS/dispense, WEMOS/coindone topics and publish to the WEMOS/detectcoin, WEMOS/machinestolen topics. When the knock sensor on the WeMos detects a coin, it publishes a trigger message to the cloud server which then forwards the message by publishing to PI. This trigger prompts the PI to take a picture, then publish back to the cloud server. The cloud server then classifies the coin based on our weighted voting based on our 3 different models, CNN, RF and KNN respectively and publishes back the final result to WeMos.

Additionally, when someone tilts the vending machine beyond a certain angle, the mercury tilt sensor is triggered which publishes a message to WeMos to alert the owner of the action.

Raspberry Pi - Server

We used a Raspberry Pi to take pictures of the coin. It subscribes to the PI/takepicture topic and publishes to the PI/imagedata topic. Once it receives a message, it takes a picture with our predetermined settings and sends the image back to the cloud server.

Telegram - Server

The Telegram Bot publishes to TELEBOT/dispense and subscribes to TELEBOT/machinestolen. When the user confirms a snack on the bot and the payment is

confirmed, the telegram bot publishes to the cloud server indicating what the user has selected to dispense, in this case Snack 1, Snack 2 or Jackpot. If the machine is moved from its place, the mercury tilt sensor will be triggered and the WeMos publishes a machine stolen trigger to the cloud server. The owner then gets a message from the bot indicating the machine has been moved.

Insights

For inter WeMos communication, I2C was initially chosen but dropped due to its malfunctioning library for esp8266 [9], resulting in I2C working on master mode only. Thankfully, we were able to use I2C for our LCD display. The I2C pins also were linked to D1 and D2 pins, and the input signal interferes with I2C. This led to the design rule that communication pins should be fully segregated from I/O pins. Regarding UART, it is crucial that there is a common ground demonstrated in figure 10 above. In UART, specifying the number of bytes received and sent is also required. While using the UART, we found out that we should not use *Serial.println()* concurrently as it interferes with the transmission, as the bytes will be out of order.

Machine learning

Data collection

We were tasked with collecting our own dataset of coins as we were unable to find suitable datasets online. We used new 5 cent coins as our false ground truth label, while old and new 10 and 20 cent coins are our true ground truth label. As each coin has both heads and tails, we collected a total of (5 types of coins x 2 sides x 30 samples) 300 pictures. We have an initial set of 10 classes (2 sides x 3 types of coins x 2 different generations (10 and 20 cent only)) which we eventually shrunk down to 3 after in depth analysis.

In our preprocessing, we clipped and rotated the imagery 4 degrees within our own load dataset function in an effort to reduce noise and unmeaningful pixels.

Figure 12: Pre and post processed image side by side

Machine Learning Models

We used a weighted voting of 3 supervised machine learning models, allowing for ease of editing and adjustment. This is in contrast to a majority voting model, as from an objective

point of view, not all models are equal and we are also backed by explainability and interpretability of our KNN model. As a result the weights we assign are 0.50 for KNN, 0.25 for CNN and 0.25 for RF. We then used the maximum predicted class and if there is a tie, fall back to our KNN model's prediction as it is most accurate and reliable out of the 3 models.

Convolution Neural Network (CNN)

It is our belief that whatever the human eye can distinguish, a CNN can do so as well. CNN was the first model we used in our initial set of 10 classes, which proved to be disastrous and which is why we reduced the number of classes to 3.

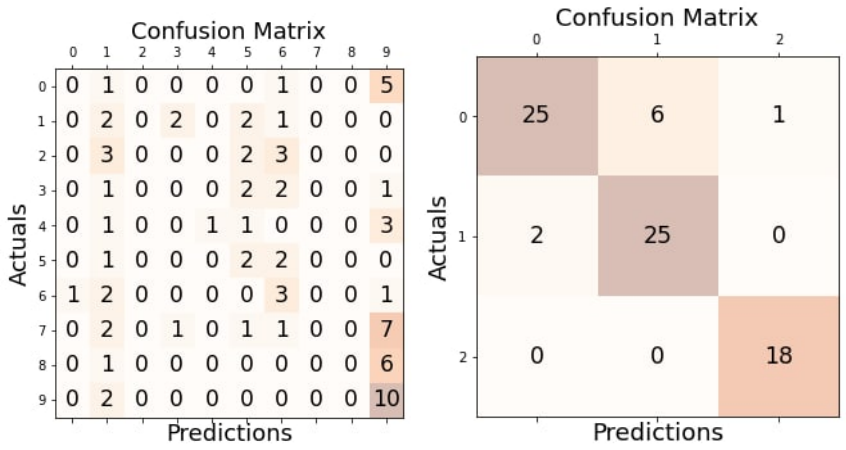


Figure 13: Test confusion matrix of 10 and 3 class CNN

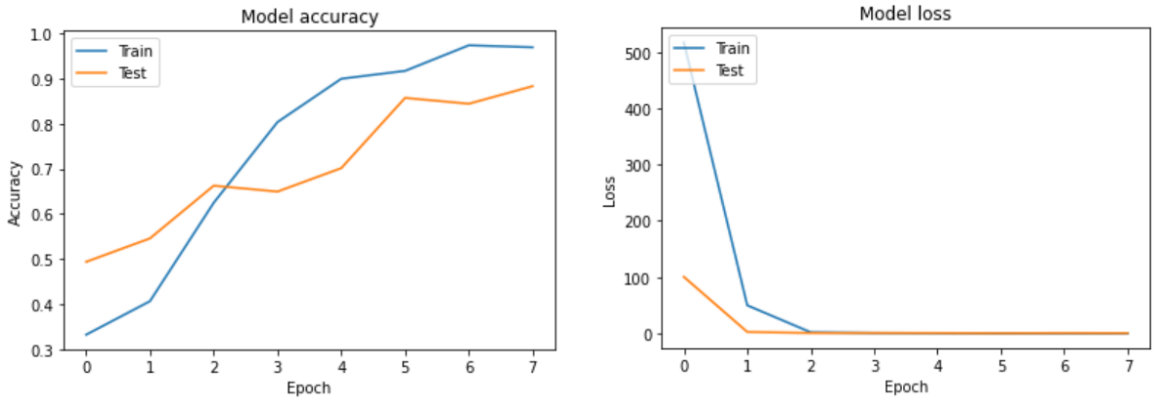


Figure 14: Training / testing accuracy and loss progression for 3 class CNN

The train and test accuracy were 49.78% and 23.3% for 10 classes, while the train and test accuracy were 96.94% and 88.31% respectively. In machine learning, any accuracy below 50% is practically worthless as you are better off flipping a fair coin. We implemented trial and error to vary the convolution, pooling and dropout layers shown below.

```
model = Sequential()

# Convolutional layer
model.add(Conv2D(32, kernel_size=(3,3), activation='relu', input_shape=(IMG_WIDTH, IMG_HEIGHT, 3)))

# Max-pooling layer, using 2x2 pool size
model.add(MaxPooling2D(pool_size=(2,2)))

model.add(Conv2D(64, kernel_size=(3,3), activation='relu'))

model.add(MaxPooling2D(pool_size=(2,2)))

# Flatten units
model.add(Flatten())

# Hidden layer with dropout layers
model.add(Dense(64, activation='relu'))
model.add(Dropout(0.1))
model.add(Dense(32, activation='relu'))
model.add(Dropout(0.1))

# output layer for output for all categories
model.add(Dense(NUM_CATEGORIES, activation='softmax'))
```

Figure 15: Implementation of CNN

Even though CNN is a widely used model for image classification, it is often described as a black box model. This is because while it can approximate any function, studying its structure will not give us any insights on the structure of the function being approximated.

Random forest (RF)

The second model we tried was the Random Forest. Random Forest is an ensemble learning technique, which decreases overfitting without decreasing performance. It does this through bootstrapping, which is random sampling with replacement. Next, multiple different decision trees are trained on each subset. When searching for the best attribute to split on each tree the set of all features are considered. It also uses aggregating, which then combines multiple predictions via averaging or majority voting. We then hypertuned our parameter via *RandomizedSearchCV.* Although ensemble learning techniques are very successful in many kaggle competitions [10], we achieved a performance of 88.3% test accuracy compared to 100% training accuracy, meaning our model has overfitted.

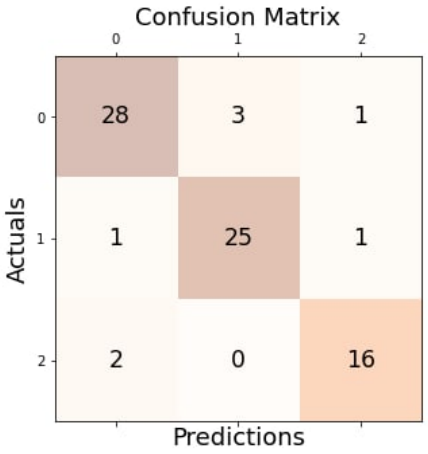


Figure 16: Test confusion matrix of RF

K-Nearest Neighbours (KNN)

On a more theoretical basis, we used KNN with hough transform specifically for circles. Hough transform is a feature extraction technique used for detected edges. We believed that using the circle variant of hough transform, *cv2.HoughCircles* as a feature would result in a high accuracy rate. This is because the shapes of our coins are also circular in nature. We implemented blurring prior to calling the hough transform to reduce noise and avoid false circle detection [11]. Using circle radius as our only feature, our KNN achieved a 96.94% train and 93.51% test accuracy.

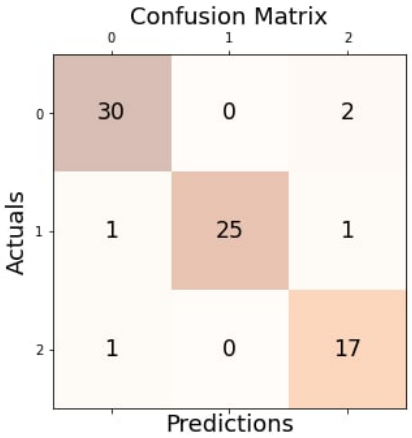


Figure 17: Test confusion matrix of KNN

Regarding circle radius, we also have to tweak the minimum and maximum radius of circles detected via *minRadius* and *maxRadius* parameters respectively. Miscalibration would result in wrong circles detected.

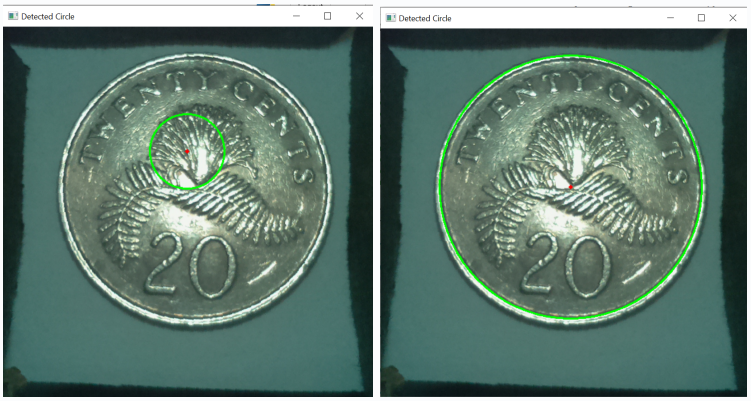


Figure 18: Left: minRadius = 10, maxRadius = 150 ; Right: minRadius = 100, maxRadius = 300

Insights

Collecting our own dataset was a valuable experience albeit time consuming. For loading data to a neural network, all images should be the same size, done through the cv2.*resize()* function. Even though we used a combination of all 3 models, our runtime of testing a single coin lies around 3 seconds, which we deem acceptable for a vending machine. We note that although KNN has the highest accuracy, we cannot totally rely on it as there may be

potentially different value coins with the same radius. Our code is thus written for easy portability for foreign countries' coins, where we can tweak the weighted average values easily to suit specific needs. Should coins of similar radius need to be distinguished, we can assign lower weights for KNN and correspondingly give higher weights to the other 2 models.

Server

Our server is hosted on a Virtual Machine on google cloud platform (GCP) as it is secure and low-cost per use. GCP offers data recovery services in case of data loss, and it is easier to collaborate and run code on a centralised server rather than on multiple devices. GCP also allows us to run our code 24 hours with low costs given that most vending machines are switched on 24 hours for convenience.

Insights

GCP is pay-per-use, and from a development standpoint, we do not need the vending machine to be on all the time, only paying for active use. Using a low-spec virtual machine on the cloud, brings us to the cost of around 2 cents per hour of usage. Additionally, GCP provides 300 dollars of credit for students and small and medium enterprises, meaning our server could potentially run for years without additional out of pocket costs.

Cloud Backend

For our backend, we decided to use MongoDB Atlas hosted on AWS Singapore for security and ease of access. MongoDB Atlas requires database owners to register IP addresses that can access the database [12]. Since our telegram bot queries the database, limiting access to administrative IP addresses ensures proper access control to prevent malicious attacks on the database.
Additionally, having a database on cloud allows easy adaptation to other vended products as well as scalability as the same software architecture can be reused with little configuration.
We've decided to use a NoSQL database as the data is stored in collections of documents. It fits our requirement as it is more flexible as compared to SQL databases. For example, the attributes for snacks and jackpot differ.

Database structure

The structure of the database is as follows: 4 collections containing documents of users and owners details, users purchase history and snacks information respectively.

User details such as their userID, name and telegram handle are stored for identification purposes. This is important as the userID uniquely identifies the user and allows the Bot to send messages to them. Account balance is also stored here as their telegram account acts as a ewallet for the system.

For owners, their details are stored separately in a different collection. We stored the unique userID of telegram users and the telegram bot will query the database for authentication of owner-level privileges.
We stored the name and cost of products to ensure that the telegram bot will have the same functionality even if products were to be changed. In this case, only the individual document in the collection would have to be updated, and additional configurations do not cascade past the database. Snacks quantity will also be updated for the owners to restock when the stock is low.
Lastly, purchase history of users are also stored every time a successful purchase has been made. This allows for long-term data analysis of the business, as well as a way for users to check their purchasing records.

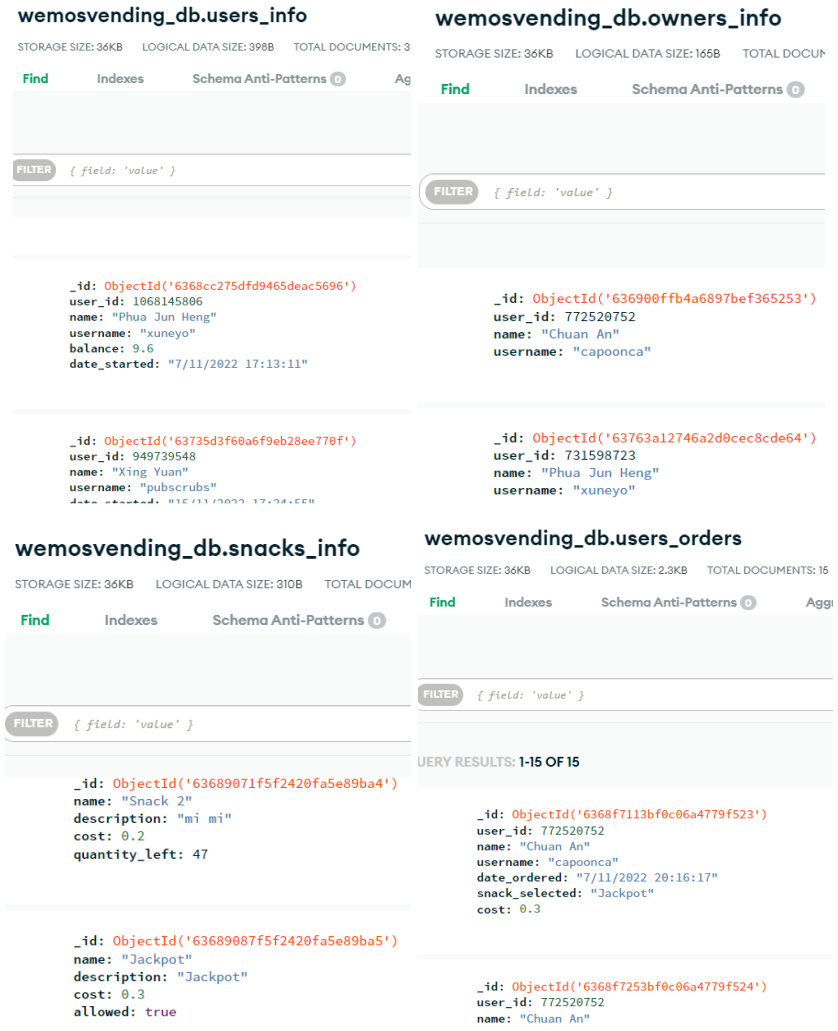


Figure 19: Database structure of mongoDB.

Insights

MongoDB Atlas gives a free cluster to students or low-scale users and multiple databases can be created. If different types of vending machines were to be created, multiple databases can be stored on the database without editing of current databases. This ensures scalability of the project by modulating the hardware portion as well as backend portion, while keeping the server and telegram bot as a staple.

User Interface Frontend

We chose to use a Telegram Bot as our frontend because it is a secure, cross-platform application for both iOS and Android users. This is convenient as users would not have to download an additional application on their phones. They would also not need to create an additional account as their Wemos VenD1ng account will be tied to their Telegram account handle. To sign up, users simply have to activate the WeMos VenD1ng bot on Telegram.

Data Storage

As mentioned in the section above, for persistent storage such as account details and purchase history, all data is stored in MongoDB Atlas. For security, we have inbuilt authentication for owners stored in the database in order to give the bot owner-only commands which additionally improves the security as well as easily extending access to new owners or users.

Our telegram bot is split into owner and user functionalities. Detailed usage is shown in the video demo linked in the first page of this report.

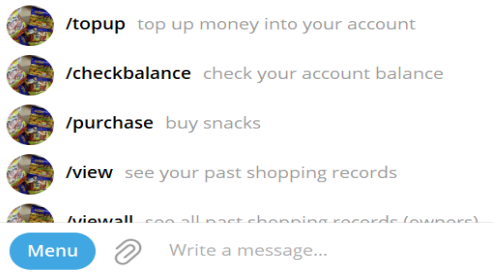
User

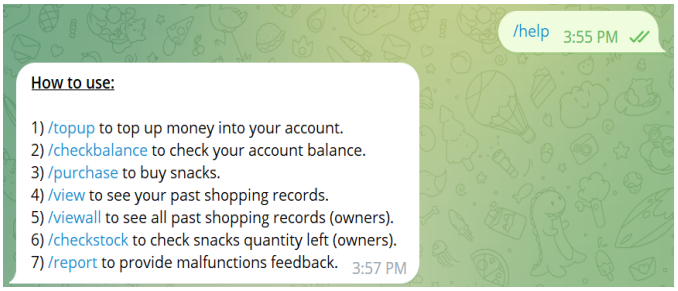
Functionalities

Telegram commands were used for users to perform the necessary actions to purchase a snack.
/topup - top up money into account
/checkbalance - check current account balance
/purchase - buy snacks
/view - see past shopping records (includes item, cost and timestamp)
/report - contact of owner to provide malfunctions feedback

User experience

A clickable menu and help section with the list of commands was also created for a more user-friendly experience. Buttons prompts were used in conversations to minimise typing required as well.





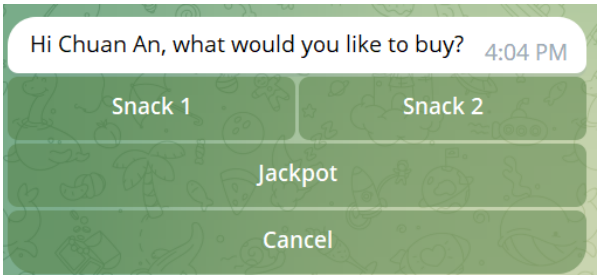


Figure 20: Telegram Bot UI

Owner

Functionalities

In addition to the users' commands, owners have additional commands which allows them to view the statistics of the purchases to better plan their business. These commands are restricted to owners only.
/viewall - see all past shopping records of all users
/checkstock - check snacks quantity left, so that the owner can be notified of when to restock.

Long Term Analytics

With persistent storage from MongoDB, we can store all kinds of relevant and useful information for future analysis. Here are some examples:

- Stock level - allows for analysis of popular items by looking at the quantity of snacks purchased. Simple regression models can be used to predict the expected business for the next few days to better plan the stocks.
- Purchase history - the cost, date and time of an item purchased by all users can be retrieved by the database, which allows for analysis such as the popular and peak business timings.

In fact, there is a lot of information that we can get from the database. We can filter the documents by users, purchased item, or time. This allows for in-depth analysis of the purchasing habits of users.
If the business expands to multiple vending machines in different locations, we are able to scale it easily by adding more collections in the database, and filter the documents accordingly. This allows for valuable insights of places with large human traffic where the machine can be deployed for more sales.

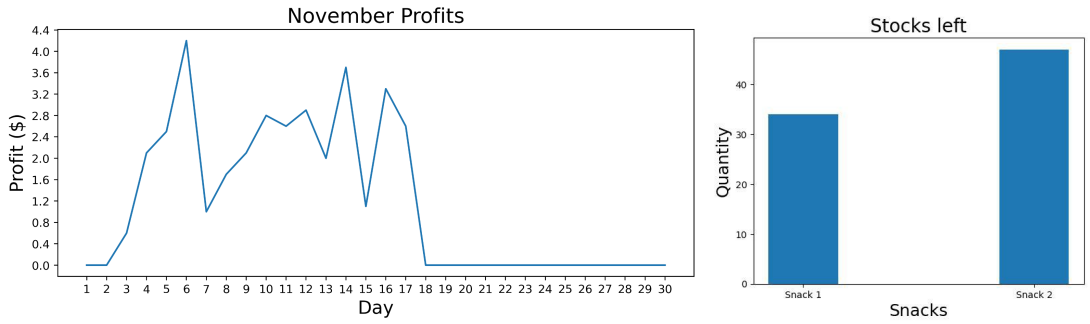


Fig 21: Examples of in-depth analysis of data collected which will benefit the owners.

Insights

We opted for a free cloud based database (MongoDB Atlas) instead of local MongoDB as it is much more convenient to deploy and collaborate as members would be able to access the database with their Google accounts. However, MongoDB Atlas is not accessible through NUS Wifi, thus an alternative hotspot has to be used if our telegram bot were to be run locally during the testing phase. This would incur additional data charges.

Considering that our application does not need many functionalities, Telegram UI is sufficient and in fact more efficient in this case. However, should we require more complex functionalities such as integrating Paylah API which requires scanning of QR codes, making an app would be a better idea.

Conclusion

In this project, we challenged ourselves by setting ambitious targets where we incorporated many aspects of an IoT system. We also took a greenfield approach and wrote the code from scratch, as opposed to adapting from open codebases. From this, we had many takeaways due to the difficulties faced. It was truly a greenfield project as we did not have anything to refer to, pushing on one problem at a time. A fun fact we found was that the first “smart device” was a Coca-Cola vending machine connected to the ARPANET back in 1982 [13].

Lastly, to misquote a famous quote: “In the grand scheme of (Internet of) Things, the average piece of junk is probably more meaningful than our criticism designating it so”. We hope our report will help future readers in various aspects through insights provided.

References

[1] M. Chui, M. Collins, and M. Patel, “IOT value set to accelerate through 2030: Where and how to capture it,” *McKinsey & Company*, 06-Apr-2022. [Online]. Available:

https://www.mckinsey.com/capabilities/mckinsey-digital/our-insights/iot-value-set-to-accelerate-through-2030-where-and-how-to-capture-it.

[2] E. Staff, “How much does a soda vending machine weigh? size & weight,” *VendingMachineInsider*, 17-Nov-2022. [Online]. Available: https://vendingmachineinsider.com/how-much-does-a-soda-vending-machine-weigh/.

[3] “How much does it cost to start a vending machine business in Singapore?,” *AsiaOne*, 26-May-2021. [Online]. Available: https://www.asiaone.com/lifestyle/how-much-does-it-cost-start-vending-machine-business-singapore.

[4] “Vending Machines,” *CokeSolutions*. [Online]. Available: https://www.cokesolutions.com/equipment/vending-machines.

[5] “L293D,” *L293D data sheet, product information and support | TI.com*. [Online]. Available: https://www.ti.com/product/L293D.

[6] “Raspberry pi camera board v1.3 (5MP, 1080p),” *Pi Supply*. [Online]. Available: https://uk.pi-supply.com/products/raspberry-pi-camera-board-v1-3-5mp-1080p.

[7] “Baseus Amblight Power Bank 65W 30000mah,” *Baseus*. [Online]. Available: https://www.baseus.com/products/amblight-power-bank-65w-30000mah.

[8] *Decathlon.sg*. [Online]. Available: https://www.decathlon.sg/p/trekking-15w-portable-usb-solar-panel-forclaz-slr900-black-forclaz-8581513.html?channable=40f1a8696400383538313531331332e2&gclid=Cj0KCQiA1NebBhDDARIsAANiDD3GoL1OPxU0VycxazNk4ysd6yyCR_L9imT0Qd8Bqa-sWTRktKZ6AHEaAj5HEALw_wcB.

[9] esp8266, “ESP8266 as a I2C slave · issue #5762 · ESP8266/Arduino,” *GitHub*. [Online]. Available: https://github.com/esp8266/Arduino/issues/5762.

[10] J. M. O. de Zarate, “Kaggle Champions: Ensemble methods in machine learning,” *Toptal Engineering Blog*, 29-Sep-2021. [Online]. Available: https://www.toptal.com/machine-learning/ensemble-methods-kaggle-machine-learn.

[11] “4.3 The Hough Method for Curve Detection, ” University of Edinburgh [Online]. Available: https://homepages.inf.ed.ac.uk/rbf/BOOKS/BANDB/LIB/bandb4_3.pdf.

[12] Kathryn , V. How to fix mongodb Atlas IP Whitelisting issues, Studio 3T. 31-August-2021 [Online]. Available: https://studio3t.com/knowledge-base/articles/mongodb-atlas-login-ip-whitelisting/.

[13] *The "Only" Coke Machine on the Internet,” CMU School of Computer Science*. [Online]. Available: https://www.cs.cmu.edu/~coke/history_long.txt.