Final Report

1. Introduction

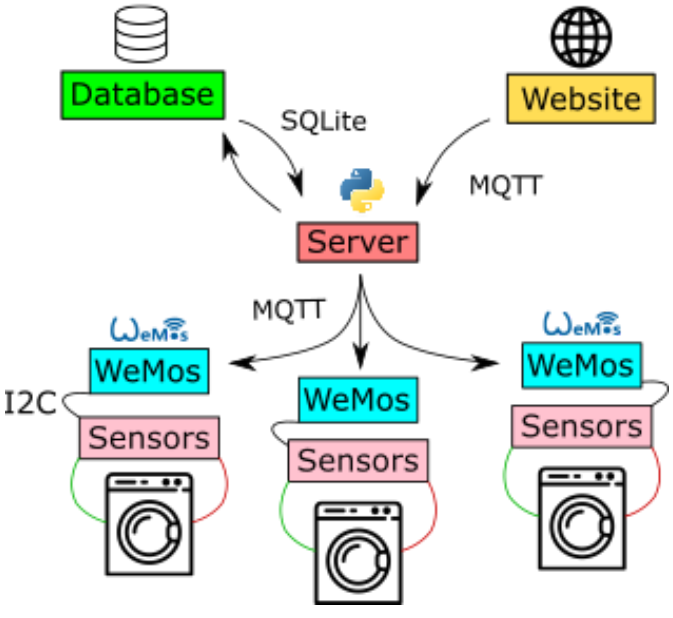
Laundromats in university colleges and halls are generally full very often and thus force users to return regularly to check for availability. This is why we have designed Laundromates: a system to monitor the usage of the laundromats remotely to check for availability, indicate usage data and track your washing machine cycle. Laundromates allows all students to access this information easily on a website, from wherever they are. Devices equipped with various sensors are set up on washing machines in the laundromat, and upon connection, data will immediately be sent to the website. Users can then find the current cycle information of the washing machines including state (I/O) and time remaining and user trends from past data - both daily and hourly.

The usage of this system is not restricted only to a university campus however, as it can be beneficial for any laundromat. In any such case with Laundromates, laundry no longer has to be a time-consuming chore.

2. Details of the approach

2.1. IoT device

Our IoT system could be split into 4 major components, the hardware which consists of the wemos and sensors connected to the laundry machine, the server which processes the data before saving it on the database or sending it to the website, the database and the website.



Data is polled from the sensors every minute and sent to the server through MQTT. Then, the server uses a machine learning model to predict whether the machines are on or off. When a washing cycle ends, data is sent to the database. In parallel to this process, if the website requests data, the server will process the data from the database and send it back to the website in a handy format through a specific MQTT topic.

2.2. Sensors used

In this project, we are using three sensors, a humidity and temperature sensor, a sound sensor and an acceleration sensor. We are using these sensors together because it allows us to compensate for the drawbacks of some of them. For example, the temperature and humidity sensors take time to capture a change in temperature that could indicate that a machine is running. In contrast, the sound sensor could directly tell that the sound has been increased. However, the sound sensor could be affected by the noise produced by the other machines. And finally, the accelerometer often faces calibration issues if the acceleration reaches a certain threshold which has been set to 2g. To summarize, the multi-sensor system is rugged because, if a sensor fails we can still rely on the other to provide data.

At the beginning of the project, it was considered to use a shock sensor instead of the accelerometer, however, this sensor is not sensitive enough to capture the machine's vibrations. The data collected is typically in the range of 1g to 1.1g for the accelerometer, between 28°C and 50°C for the temperature, 18% to 80% for the humidity and 40dB to 120dB for the sound.

We decided to use a sampling frequency of once per minute. This was decided as it was a nice equilibrium between data accuracy and power usage. Furthermore, a higher sampling rate would not have benefitted us greatly as it is approximately the time it takes our model to detect a change in state with high confidence.

2.3. ML model

For our project the problem at hand is one of classification to either an 'on' or 'off' state, given four inputs from the aforementioned sensors (acceleration, humidity, temperature, sound). For this, we decided to test both a decision tree classifier model and a neural network model.

The first step was to collect data for supervised learning. Since there was no large-scale data available online, we had to collect the data ourselves. We did this by collecting data from available washing machines and indicating the current machine state. We adjusted the sampling frequency to get more data. In order to increase our sample size further, we decided to add gaussian noise to every column of the data (the standard deviation was adapted for each column).

The next step in processing the data included scaling the data with a MinMaxScaler to ensure that each data could impact the classifying algorithm.

After adjusting the parameters in both models, we ended up with a precision of around 97% for both. We ended up selecting the neural network model with two layers, 100 epochs and a learning rate of 0.2. This was mainly because the decision tree model was at a higher risk of forming a large dependence on a single sensor in the first layers. We did not want this dependence in case this sensor doesn't work as expected due to a defect or is calibrated erroneously. Another advantage of the neural network model is that, from our tests, we had a slightly better response time; the transition between states reached a confidence score of 80%[1] in a shorter time.

The file nn.py creates the neural network model and hence takes as a sample, supervised data stored in data.csv and creates a '.hd5' folder containing the model.

2.4. Database model

The amount of data polled by sensors and sent by microcontrollers is being used to make predictions and determine the states of the laundry machines. All this data has to go somewhere in order to provide a useful application. This is where the database comes in. Specifically, a relational database.

First defined by E.F. Codd in 1970 in his research paper "A Relational Model of Data for Large Shared Data Banks", a relational model organizes the information into two-dimensional tables, called relations. Columns are the attributes and rows are records that represent an actual data observation, and are identified with a unique key.

An RDBMS is the system used to manage such a database, making the underlying work of such a system transparent to the user who can focus on managing his data. Such systems often are built with an SQL framework that allows for the smooth creation of tables and insertion, querying, deletion and other useful operations on data. Examples include PostgreSQL, MySQL etc.

The initial plan of this project was to set up a MySQL server, but complications with the server's Python initializer made it more interesting to switch to SQLite, a system that proposes the same services as other RDBMS with the difference that the information is stored in a local file on the disk. Mainly, it offered the relational properties and SQL API that the other more popular systems offer. Since the server was based on the localhost, it made sense to store the information locally as the data would then be sent over MQTT using WebSockets meaning there was little to no use to having the data stored remotely on the cloud.

Thus, with the use of SQLite, we defined one table that stored data as it was polled by sensors and sent by the WeMOS microcontroller to the server. Once the data is received, and the prediction is made using the ML model by the server, insertions are made to the database anytime a laundry cycle completes. The database stores the ID of the machine that ran the cycle, its duration (in minutes), its start timestamp (UNIX-Posix), as well as the day when the cycle ran and the hour.

[1] We only considered results over 80% as then our errors in prediction would be negligible

This data is useful as it provides updated information on the current state of the machines. The website sends requests for updated data to the server and handles this request by querying a set of information, organizing them into a dictionary and sending the latter as a JSON object through MQTT.

The information queried from the database includes the count of cycles per day, the average cycle duration, the sum of the duration of all the cycles that ran during the current day (for energy consumption purposes), and the count of cycles that ran during that day.

2.5. Website

The end product of the laundry data collection and processing is a website where users can view live machine statuses. On this site, we display the availability of machines and if they're not available, each of their estimated remaining times. In addition to machine availability, we display stats like the average cycle duration, daily power consumption, a graph of typical usage per week, and a graph of typical usage by hour based on the current day of the week.

The site is a simple, single-page app built with ReactJS on Node.js.

Retrieving the data from our IoT system directly within the browser posed some challenges, specifically because the MQTT protocol used by our server is not supported in a web browser. To work around this, we used an implementation of the MQTT protocol through the web-supported WebSocket protocol. The communication relies on two MQTT topics, one in which requests for data are sent, and another in which the data is sent back. Upon receiving this data, the site dynamically generates the charts and graphics seen by the end user.

3. Implementation details

3.1. Communication protocol

3.1.1. Wemos to the server with MQTT

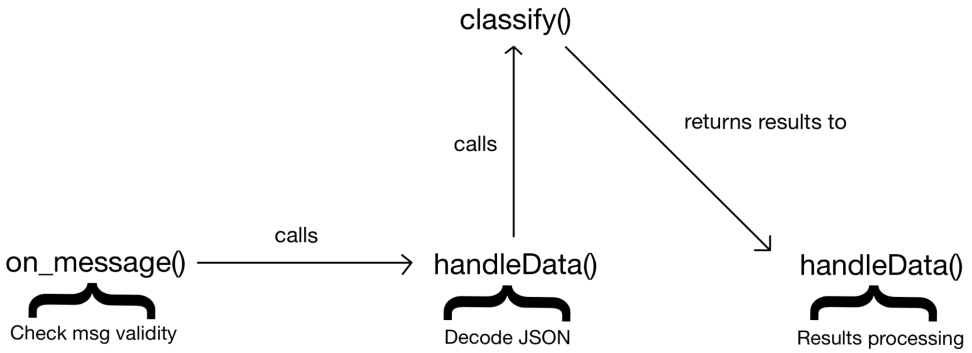
Each wemos connected to a machine sends data to the server every minute in the following JSON format:

```
snprintf (msg, MSG_BUFFER_SIZE, "{\"id\": 1, \"data\": [%1.3f, %2.3f, %2.3f, %3.3f]}", accel, temp, hum, sound);
```

This data is sent through MQTT on the topic named *IOT_nus_test*. For this project, we used a public broker setup on the website mosquitto.org. The exact IP address is 91.121.93.94 on port 1883 which is an unencrypted and unauthenticated port. The wifi connection is ensured by a phone's hotspot on which every wemos are connected.

To receive the data, the server subscribes to the same topic. As soon as a message is received, it checks if it is not coming from the website, and then the data is decoded and sent to classification. Once the classification is done the *classify()* function returns the results to the function *handleData()* which will process the information and decide whether or not a new washing cycle has been started or if a cycle has been stopped. Before processing the

server checks if the classification's results have a score higher than 80%. This condition allows for filter fall classification during the stop and start of the machine.

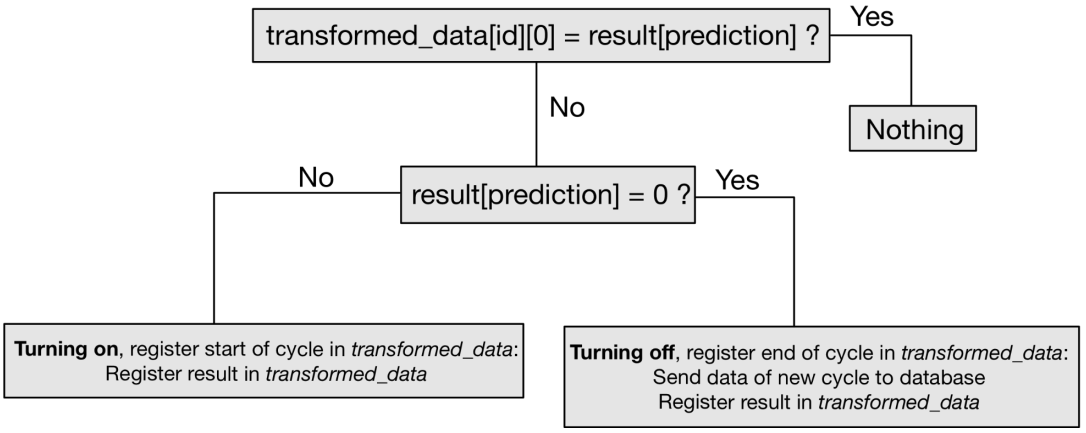


function calls in the server

To process the data, the function handleData() will compare the classification's prediction with a variable called transformed_data which is a general variable containing for each machine information about the value of a prediction and the timestamp of this prediction. This variable is initialized to:

```
transformed_data = np.array([[0,today],[0,today],[0,today]])
```

Where 0 means that the server considers that at the beginning of the program all the machines weren't running and where today represents the timestamp of the exact time where the server has been started. The figure below illustrates the functioning of the algorithm.



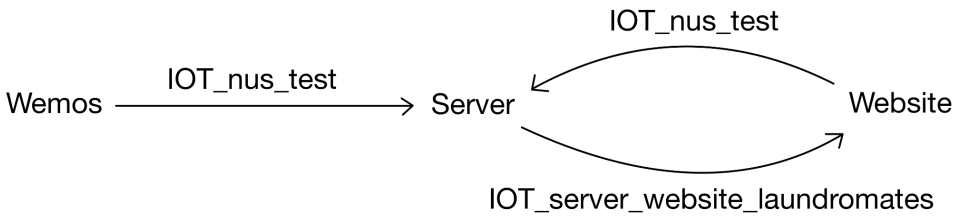
handleData() algorithm's functioning

We can notice that it is nothing more than a XOR condition between the prediction stored in *transformed_data* and the result of the current prediction. Depending on the current prediction, we can then know if a cycle has started or it just came to an end. On one hand, If a new cycle is detected, then the timestamp of the current prediction is stored in

transformed_data. On the other hand, if the end of a cycle is detected, the data concerning this cycle is stored in the database.

3.1.2. server to the website with MQTT

The website sends requests on topic *IOT_nus_test* and receives an answer on its dedicated topic named *IOT_server_website_laundromates*. The figure below summarizes the functioning of every topic.



MQTT topics' graph

3.1.3. I2C for IMU

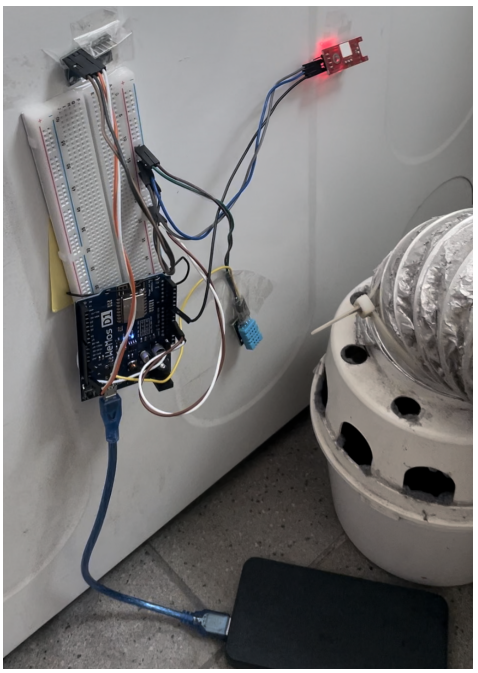
To communicate properly with the IMU, we used a dedicated library of Arduino software. Therefore we didn't have to deal with low-level I2C communication. The library is called MPU6500_WE.h and requires Wire.h as a dependency. The I2C address of the IMU sensor is 0x68. This communication protocol needs two physical lines which are the SCL and SDA pins also called D0 and D1.

3.1.3. One-wire communication with DHT11

The temperature and humidity sensor DHT11 uses a one-wire serial communication protocol. This low-level communication is carried out by a specific library designed for the sensor called DHT.h and DHT_U.h. Thus we can use high-level methods to extract easily the data we need. This communication protocol needs only one digital pin which is the pin D6 for our application.

3.1.4. Communication with sound sensor

The sound sensor can be used either in digital or analogue mode. For this project, it has been decided to use the sensor in analogue mode. That's why it is connected to pin A0 of the wemos board. The figure below shows how the sensors are wired to the board.



3.2 Database implementation

As explained above, the database implementation was over SQLite which is a RDBMS that stores the information locally in a file on disk. As the name suggests, the system provides a framework to create tables and manage data using the SQL API. Therefore, our database implementation consisted of one table named ‘predictions’ which was created using the following SQL command:

```
CREATE TABLE IF NOT EXISTS predictions(id INTEGER, prediction BOOLEAN,
duration INTEGER, timestp INTEGER, day VARCHAR, hour VARCHAR)
```

The implementation is run through Python. Python provides two popular interfaces for working with the SQLite database library: PySQLite and APSW. Each interface has different uses. We used the PySQLite library, using a connection object, and a cursor to execute statements.

Most SQL commands were executed with a try-except block of code that allowed us to catch any database errors and handle them adequately. The SQL commands themselves are rather standard (just like the CREATE statement above) and include insert and select statements that are executed at appropriate moments in the driver’s execution.

4. Experimental evaluation

4.1. Experimental accuracy and evaluation of the approach

During the final batch of tests, it has been noticed that our system takes approximately one minute to detect a changing state of a machine with a score higher than 80%. For this

reason, we decided to poll the sensor every minute. This way, the power consumption, as well as accuracy, could be optimized.

4.2. Power management

As our sampling rate is once per minute, we benefit from sleep-wake cycles in order to save power.

We decided to use light sleep in our cycles. This allowed for better power-saving capabilities than modem sleep, but without shutting off the CPU as in deep sleep. The complete shutdown of the CPU was not an option as this would not keep our IMU sensor calibrated. A recalibration was not plausible as calibration during a running machine would greatly impact the data obtained. For this, we attempted to save and load calibration data but we did not manage to find a way to accurately keep the data consistent.

We thus have the first part of the cycle taking around six seconds[2] of active mode where the esp8266 consumes around 70mA. This includes connection to a station Wi-Fi, connection to an MQTT server, reception of data and the sending of the data through MQTT.

Next, we have one minute of light sleep mode with a power consumption of around 1mA where the Wi-Fi and hence the MQTT connection are interrupted.

The sensors' consumptions are negligible during standby: the sound sensor consumes negligibly (input pin); the IMU consumes around 9.3µA; the DHT temperature and humidity sensor consumes around 0.3mA when active and 60µA on standby.

Since our device is static, it can simply be powered with a plug. It was hence not an absolute priority to have the device go into deep-sleep.

5. Summary

5.1. limitations of the solution

One of the main limitations of our system is that the data for the time remaining, indicated on the website is not immediately universal. This means that if the device is changed from one laundromat to another, each with different cycle durations, the indicated time would not be correct at first. We decided to use the past data to dictate the cycle duration. This means the system would eventually adapt to the new environment but with inertia. One way of solving this would have been to only use the past few data, but this would have come at the risk of having the cycle duration be greatly affected by a machine that stopped too early.

Another limitation that our system currently has is its dependence on the sensors working correctly. As discussed in 2.3, we chose our model to minimize the dependence on a single sensor. Nevertheless, it could be that if one of the sensors started giving false data in the wrong manner, the model would begin to make false predictions.

[2] Not including the first cycle which takes slightly longer

We could have solved this by checking in our Arduino code for realistic values before taking in the sensors' data.

5.2. possible future directions

While working on our project, we have identified varied ways that our system could be improved. These were not feasible to us due to time constraints or simply because we realised them in retrospect and would have had to restart the project, but we believe the following changes would have been interesting to adapt.

Firstly, we would have liked to add a more personal feature that allowed users to interact with the website with an account. This would have unlocked multiple features such as personal notifications giving updates on the user's laundry cycle (through for instance a telegram bot) and data indicating the frequency of use of the washing machine along with the money spent or power consumed. To take this one step further, we could have allowed for the start of a user's cycle to be detected by a camera positioned near the washing machine with facial recognition.

In terms of structure, we would have liked to centralize all sensors onto one WeMos board through ADC converters, rather than the system with one WeMos per machine that we have developed. This would have drastically reduced the cost of our system. The change would be feasible as the WeMos can supply enough current for a very large amount of sensors. However, the system would then entirely be dependent on one WeMos and everything would shut down if the board had a problem. In our project, we decided not to do this as wiring multiple machines to a single WeMos would be very inconvenient to set up temporarily as the lengths of the cables required would have to be different for every laundromat.

In terms of our model accuracy, the collection of more data would have been beneficial to have better accuracy and hence a faster response to state changes. Furthermore, the model could have been made more universal, with data from many different kinds of washing machines.

Finally, we would have liked to adopt deep sleep to save a maximum amount of power. This would have consumed a very small amount of current during standby. However, as explained in 4.2, we were not able to achieve this due to issues with the IMU calibration.