Dyson School of Design Engineering | MEng Design Engineering

Sensing and IoT Coursework
Max Hunt

Module exam

Module code and Name	**DE4-SIOT Sensing & IoT**
Student CID	
Assessment date	**4pm 14th Jan 2021**

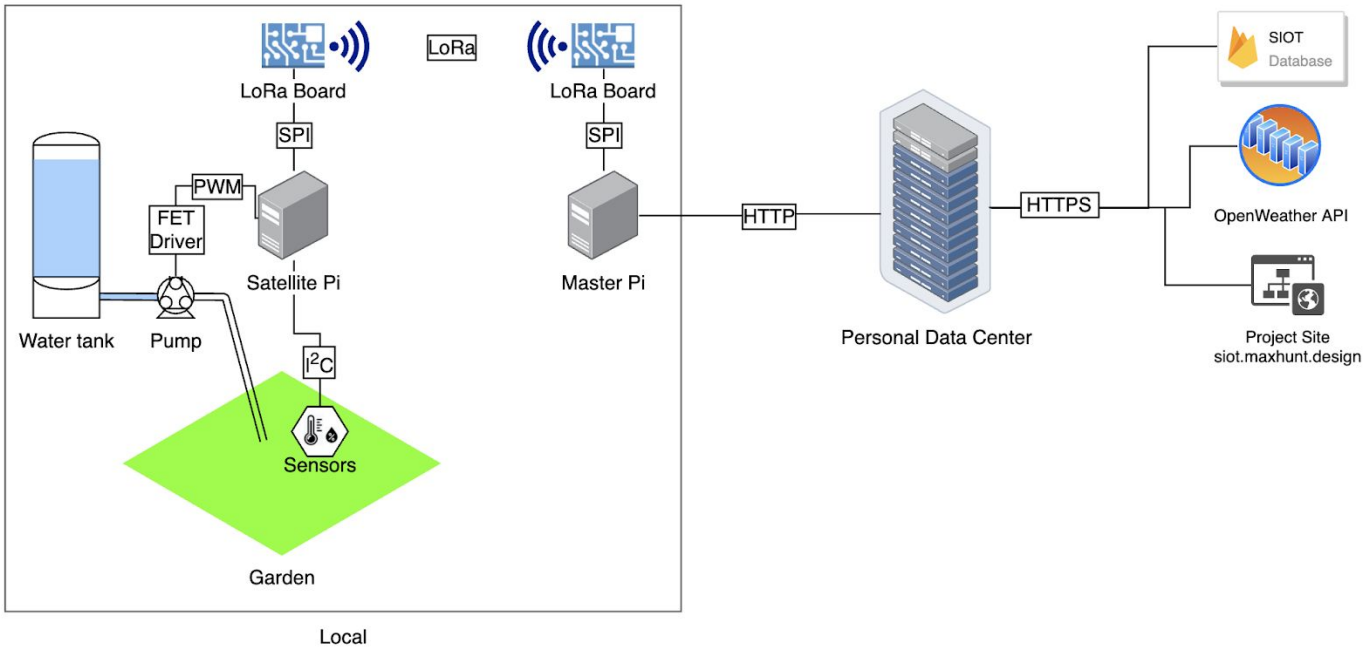
Presentation URL (publicly accessible link):
YouTube video: https://youtu.be/dQw4w9WgXcQ?t=43

Code & Data (publicly accessible link):
Code repository: github.com/maxDeCod3r/DE4-SIoT

Database: Securely hosted Firebase project, exported dataset can be found in repo, most recent dataset available on request.

WebApp: siot.maxhunt.design

Graphical Abstract



Coursework 1: Sensing

This project aims to use IoT technology to automate data collection of weather data and local soil data. The collected data will later be used to identify opportunities to optimize plant watering patterns.

Intro and objectives

It is estimated that up to 50% of water used for plants goes to waste [1], this is caused by farmers overwatering crops to ensure they do not dry out. In the case of agriculture, overwatering is much safer than underwatering which could lead to loss of crops. Since the plant evapotranspiration index changes every day depending on the weather, plant water requirements also change. Much of this water could be saved if the evapotranspiration index was considered prior to watering. To validate this approach, an IoT system was developed that would take measurements and automatically dispense the minimum required volume of water.

System Targets

- Set up a local system for collecting soil measurement data
- Periodically collect data from a local sensor and OpenWeather
- Safely store the collected data
- Analyze the data and identify correlations
- Use ML to make useful predictions in watering patterns
- Implement a streamlined automatic watering system
- Visualize the data with an online WebApp

Planning

For optimal results, over 7 days of data needed to be collected for the data processing stage. The project was broken down into manageable tasks and a timeline was created (Figure 1). The project was managed in the Notion [2] platform. Notion was chosen as it provides an all in one platform for information storage, planning tools, and a Trello style board interface for easy task management.

The initial part of the project (Sensing) consists of storage architecture, purchasing hardware, assembly, collection code writing, and data processing. The second, IoT part consists of extra hardware purchasing and assembly, automation, and website development. The third stage involves report writing, resource and documentation formatting, and video creation.

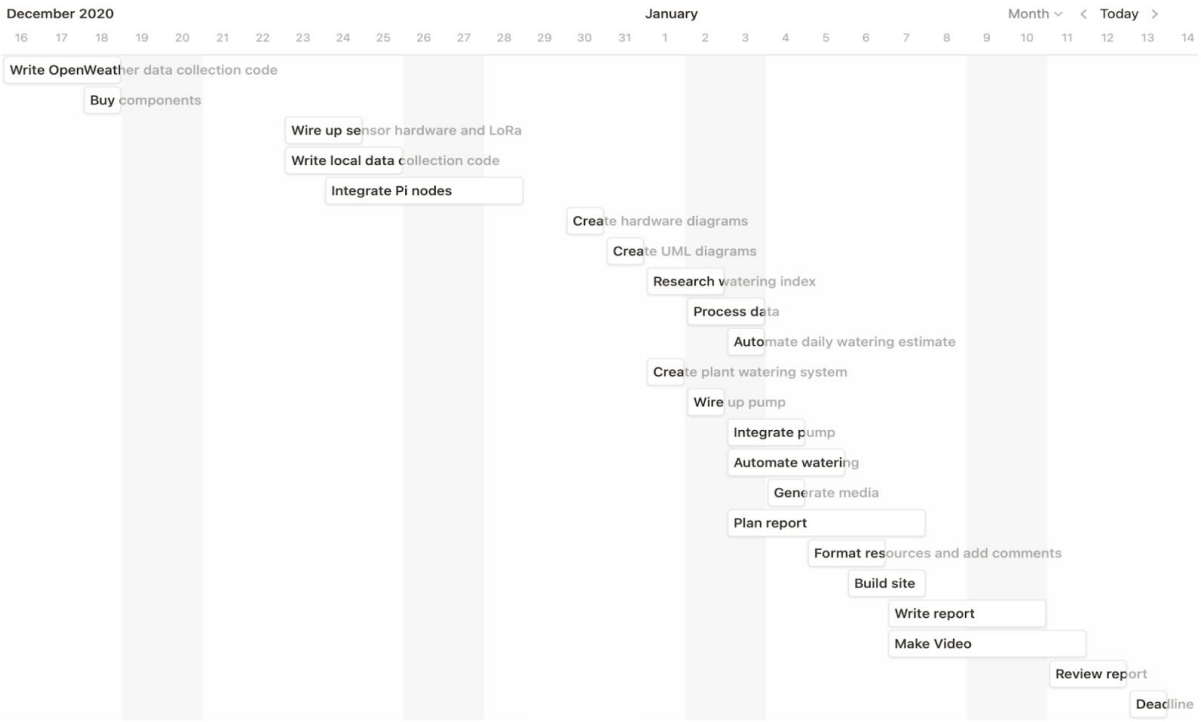


Figure 1: Project plan

Data privacy and integrity

The collected data consists of weather and temperature values that are not sensitive in nature. As a result, no necessary precautions had to be taken during data upload and processing outside of standard responsible data handling practices. Where possible, HTTPS protocols were used during data transmission. As standard, all devices were password protected and all credentials were kept away from version control.

Data sources and sensing setup

The data collection setup process started with the identification of factors that could potentially impact plant water requirements. Sources for those factors were then identified and a system was set up to poll the relevant data at fixed time intervals and upload it to a database.

Data sources

Research was conducted into different factors affecting plant evapotranspiration. From [3], [4], [5], and [6], the main factors identified were:

- Air temperature (increases transpiration),
- Air humidity (decreases transpiration)
- Cloud cover (decreases evapotranspiration)
- Wind speed (increases evapotranspiration)
- Precipitation (increases the water available to roots)
- Soil temperature (increases evaporation)
- Soil humidity (increases evaporation and indicates the water available to roots)

The OpenWeather API [7] offered reliable and free access to the first five variables and was selected for this project.

Since the project will be dispensing water in my garden, soil temperatures and humidities were to be collected locally, using commercially available sensors and IoT hardware. During testing, it became evident that the humidity probe did not function as advertised and was either defective or did not have a high enough resolution to measure soil humidity deltas. As a result, the soil humidity dataset was excluded from further analysis.

Setup Overview

Different options were carefully considered during the hardware and software selection stages, with aims to maximize reliability while minimizing cost and time to integrate. The project hardware diagram can be seen in figure 3 and a picture of the Sensor Pi in figure 2.

Note that the project site and water pump are not discussed in this part of the report.

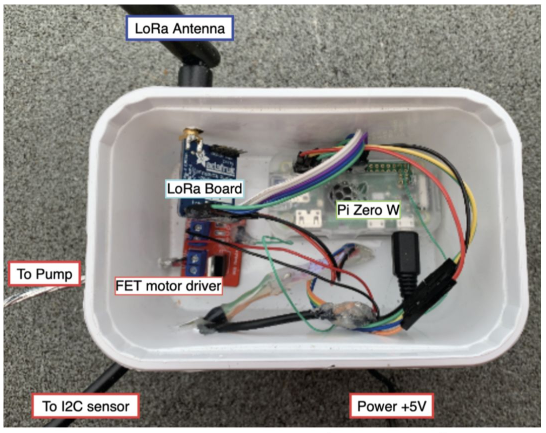


Figure 2: Sensor node internals

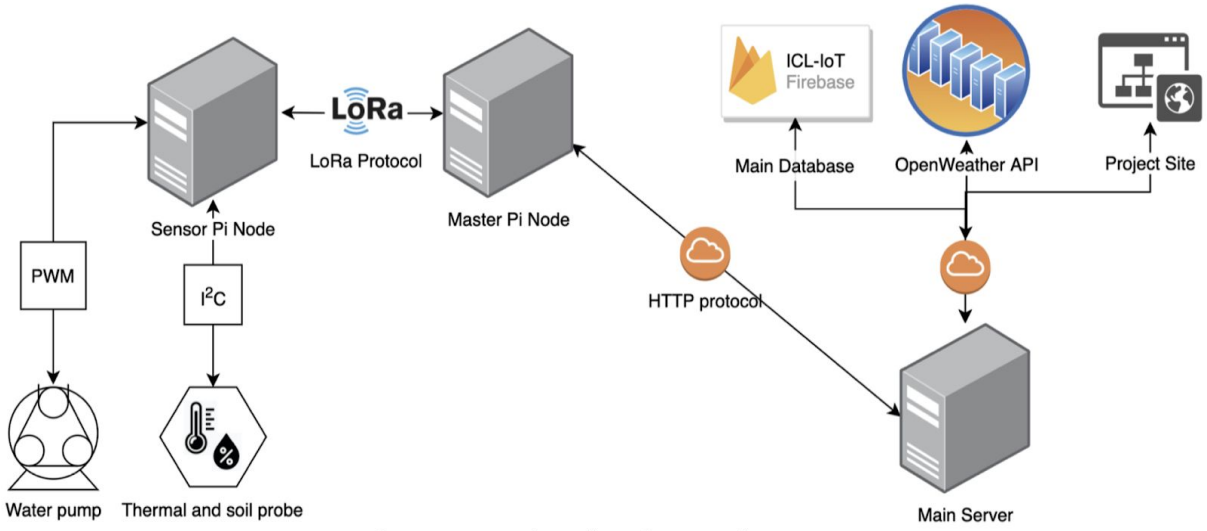


Figure 3: Project hardware diagram

Storage: Google Firebase Firestore

Firebase is a free to use fully integrated backend for small to mid-scale tech companies. It offers database services, site hosting, functions, and more. The provided NoSQL database (Firestore) is a secure and robust solution for this project's data storage.

Main Server

The Main Server/ Personal Data Center was a Raspberry Pi model 4. It was chosen as I already had it set up as a server, with all the relevant software packages for this project including python3, Docker, Nginx, and ddclient. The Pi was connected to the internet via dynamic DNS and google domains, making it remotely accessible and ideal for free site hosting. As this server operated from my bedroom, potential reliability issues were considered and addressed prior to the start of the data collection.

Pi Nodes and sensors

The local garden data collection was set up using Raspberry Pi Zero W boards [8]. They were chosen due to their low price, powerful processor, and Linux architecture, simplifying setup and providing support for docker.

Due to the location of the plants relative to the wifi access point, the sensor Pi was too far away for a reliable internet connection. A second Pi was used as a relay. The master Pi was connected to WiFi and used an RFM69HCW [9] LoRa radio to communicate with the distant sensor/ satellite Pi, as seen in figure 3. The LoRa protocol was chosen as it offered long-range communications while being affordable in the scope of this project. The sensor Pi was connected to two I^2C sensors: an MCP9808 [10] temperature probe and a STEMMA [11] soil humidity probe.

Problems and unexpected complications

During the wiring and testing of the sensors, the MCP9808 sensor died unexpectedly and no longer showed up on the i2c scan. This issue was resolved by using the temperature sensor onboard the STEMMA probe.

During long-range testing, it was identified that the RMF69 basic wire antenna [12] was not sufficient. Pycom [13] antennas were ordered from RS and attached to the radio boards using SMA Edge Mount Connectors. The modifications sufficiently improved the radio range.

Data collection and storage process

The data collection occurred every hour. The hourly polling rate is directly driven by the OpenWeather precipitation data, available in hourly or daily formats. A detailed UML diagram can be seen in figure 4. Note that the second loop is relevant to data actuation and is not discussed in this part of the report.

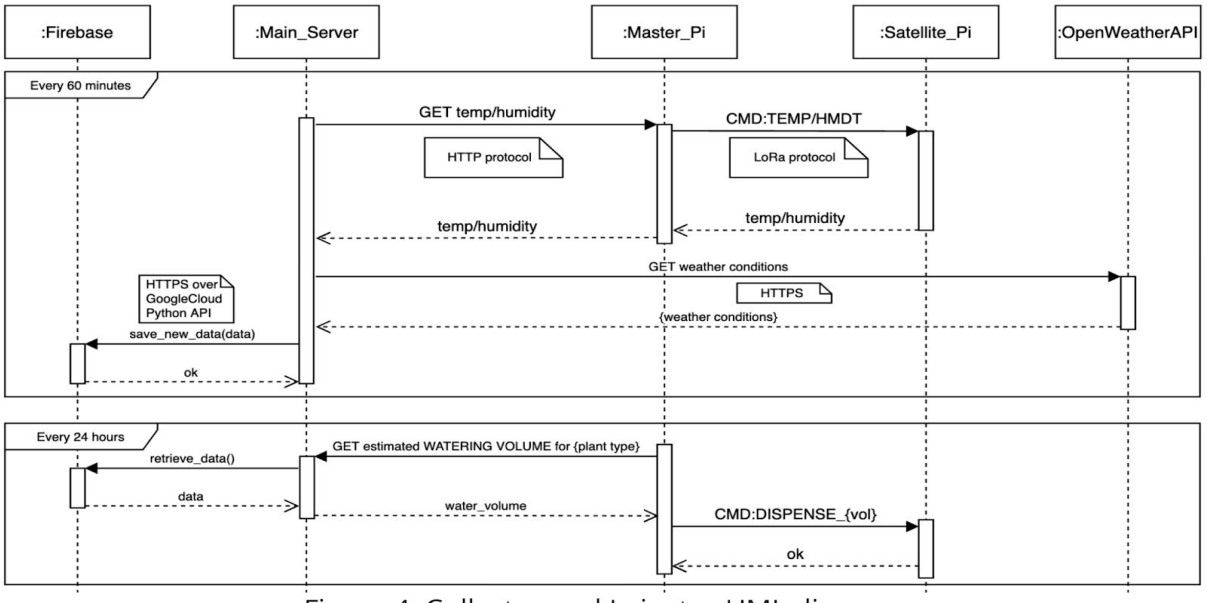


Figure 4: Collector and Irrigator UML diagram

The Main Server runs a docker container with a python script. Every 60 minutes, the script polls the local soil data from the Pi node. The Master Pi is running a dockerized Flask server. On an HTTP GET request, the Master Pi sends a command to the Satellite Pi via LoRa, querying temperature or humidity. The relevant data is then returned to the Master Pi, decoded, and sent to the Main Server. The Main Server then polls data from the OpenWeather API via a similar GET request, providing an API key and location parameters. Upon receipt, the data is validated, parsed, and extra information is added including a timestamp, DateTime object, and testing flag. Finally, the data is uploaded to the Firestore database via a secure connection through the Google Cloud API.

To avoid dependency issues and improve reliability, all python code was executed in docker containers on the Pi's. Providing the `--restart=always` flag ensured that scripts would restart in the event of an unexpected error or power loss.

Since the project used free tier OpenWeather and Firebase accounts, data rate limitations had to be considered. OpenWeather allows for 1k calls per day, and Firestore allows for 50k reads and 20k writes per day. Since the collector polled and uploaded data 24 times a day, the usage rates did not approach free tier limitations. To reduce potential issues with loading Firestore data, only the last 240 data points (10 days worth) were loaded, allowing the page to be theoretically loaded over 200 times per day assuming no caching is used.

Basic characteristics of the end-to-end systems setup and data

The complete end to end data system can be seen in figure 5. Data is generated from the local sensors and the OpenWeather API and ends in the database and WebApp.

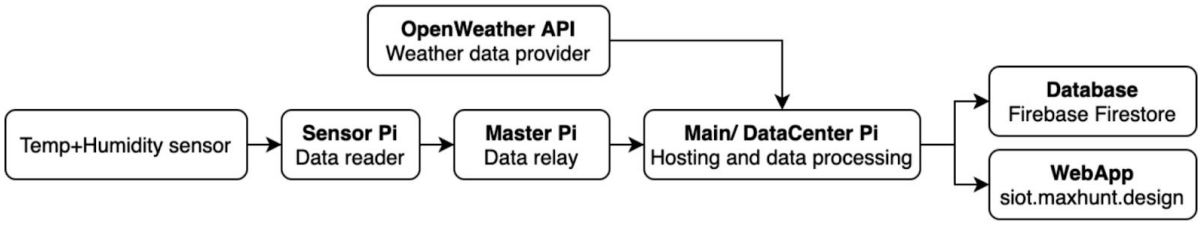


Figure 5: End to end data flow

A quick website was set up for easy data visualization. Figure 6 shows a clear cyclic pattern as well as a close correlation between soil and air temperatures.

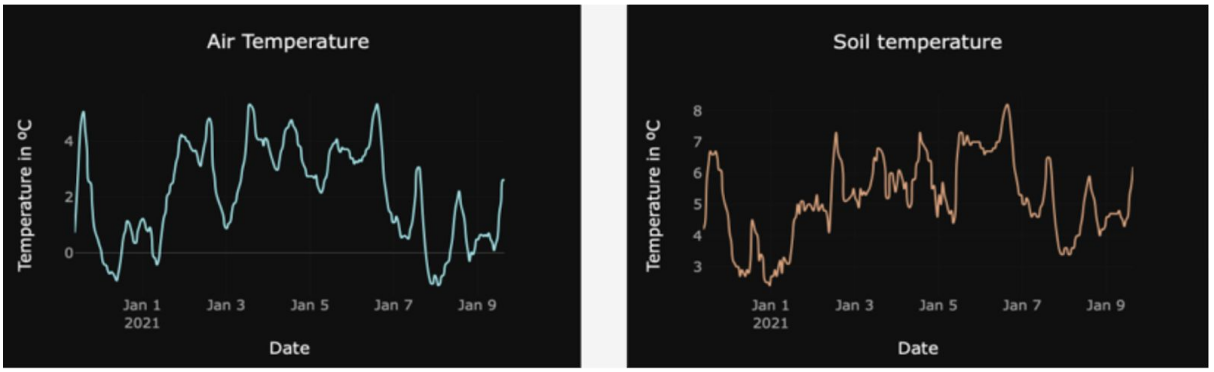


Figure 6: Raw air and soil temperature plots

Coursework 2: IoT

Once a sufficient quantity of data was collected, it was processed and an ML model was built to predict plant watering requirements based on weather data. A pump was purchased and added to the system for automatic water delivery, and a WebApp was built for data visualization.

Data Interaction/visualisation/actuation platform

Interactions and visualisation

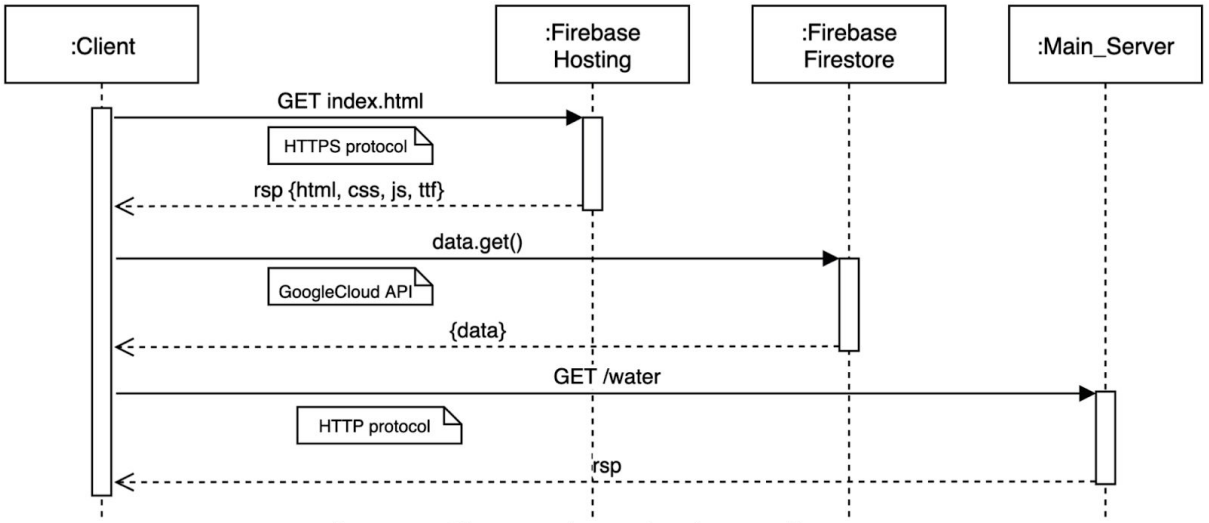


Figure 7: Client WebApp load UML diagram

The data interaction and visualization is structured around a WebApp displaying raw sensor data as well as the latest plant watering estimate. The app data loading process is outlined in the UML diagram in figure 7. The client requests the HTML page via a simple GET request. The downloaded files trigger a series of JavaScript functions that establish a connection with the firebase backend and plot the raw sensor data from the Firestore database using the plotly.js library [14]. Next, another script executes a simple API call to the Data Center Pi to obtain the latest plant watering estimate. On a GET call, the main server script automatically pulls the latest 24 hours of sensor data, cleans, normalizes it, and calculates average values. Next, the single feature row is passed into the already loaded ML model and a prediction is made. The prediction is then returned to the client via response to the GET request. This process takes approximately 2 seconds to execute but does not impact the page load time as the request is made asynchronously.

The WebApp was originally hosted on the local Data Center Pi, where the page load speed was found to be suboptimal. To improve latency and reliability, the WebApp hosting was moved to Firebase Hosting, running on the Google Cloud Platform.

Actuation

To automate the plant watering system, a peristaltic pump [15] was purchased along with a simple MOSFET motor driver [16]. Extra code was then written to enable the satellite pi to control the pump. A peristaltic pump was chosen due to its simplicity, cheapness, and repairability. In the event of a freeze, the pump's rubber hose will allow ice to expand preventing any damage to the housing.

Additionally, another dockerized python script was deployed on the Master Pi. The script runs a loop every 24 hours, querying the predicted watering volume and sending a Lora command to dispense a specific amount of water. The UML process can be found in the 24-hour loop in figure 4. A graphical explanation of all code running in the IoT system can be found in Appendix 1.

Data analytics, inferences, and insights

Analytics

The collected data was loaded into a Google Colab notebook for analysis. The data was cleaned and formatted using the pandas library. As a first attempt, an equation was created (figure 8) that would take into account the most important factors affecting evapotranspiration to calculate an estimated watering index, relative to the average amount of water to be dispensed.

$$\Delta water = w_1 air\_temp + w_2 wind\_speed + w_3 soil\_temp - w_4 air\_hmdt - w_5 cloud - w_6 rain$$

Figure 8: Initial watering equation

The equation used 6 weights ($w_{1\to6}$) to balance the variables for a reliable watering output. Selecting those weights became an optimization problem similar to linear regression. Upon further inspection, it was determined that a Deep neural network would be a better option as it could handle nonlinearities, and could easily be retrained for a different plant or soil type.

Prior to training, the data was cleaned and converted into 24-hour blocks. All variables were averaged except for precipitation, which was summed. The data was then normalized by converting values into their Z scores, ie. subtracting the dataset mean and dividing by the standard deviation.

A simple deep neural net was created in TensorFlow using Keras. The input was a feature column with the 6 aforementioned features. The net had two hidden layers of width 14 and 7 respectively, the first layer width was set to $2n+2$ where n is the dimensionality of the feature layer. The relu activation function was used and the model was trained with the Adam optimizer and mean squared error loss function. To avoid overfitting, L2 regularisation rate was set to 0.05.

The model was trained for 100 epochs with a learning rate of 0.005 and batch size 40. The final model had an MSE of 0.02, which was considered good enough. The model was evaluated by plotting the predicted values (red) against the ground truth (green) in figure 9. Note that no values are above 0.2, this is due to the data collection taking place during winter when plants need less water compared to the warmer seasons.

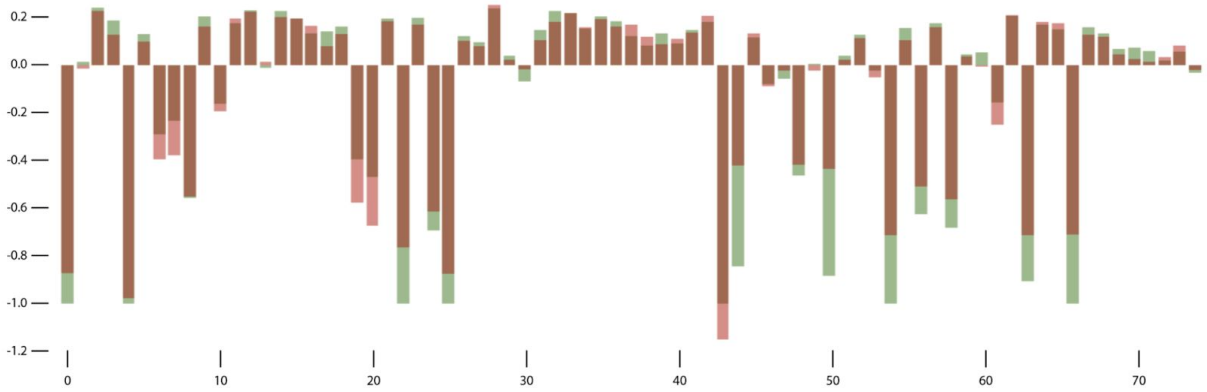


Figure 9: Plot of predicted (red) vs ground truth (green) watering biases

The model outputs values between +1.0 and -1.0, symbolizing the watering bias on any given day. A bias of 0.0 indicates plants should be watered their average amount, a predicted bias of -1.0 means that plants do not need to be watered, and a bias of +1.0 suggests that plants need to be watered 200% their average water volume. The automated watering predictor performs the percentage scaling and uses predefined values for the total garden area and average water volume per m^2 to calculate the total volume of water to be dispensed.

Inference and Insights

To determine any correlation between the measured data, a cross-correlation scatter plot was created (figure 10). The data shows a strong correlation between soil and air temperatures, $\mathbf{R}^2 = 0.9$. Some correlative properties were identified between cloud cover and other variables ($\mathbf{R}^2 = 0.4$), though this is likely caused by the resolution and rounding of the cloud cover sensor.

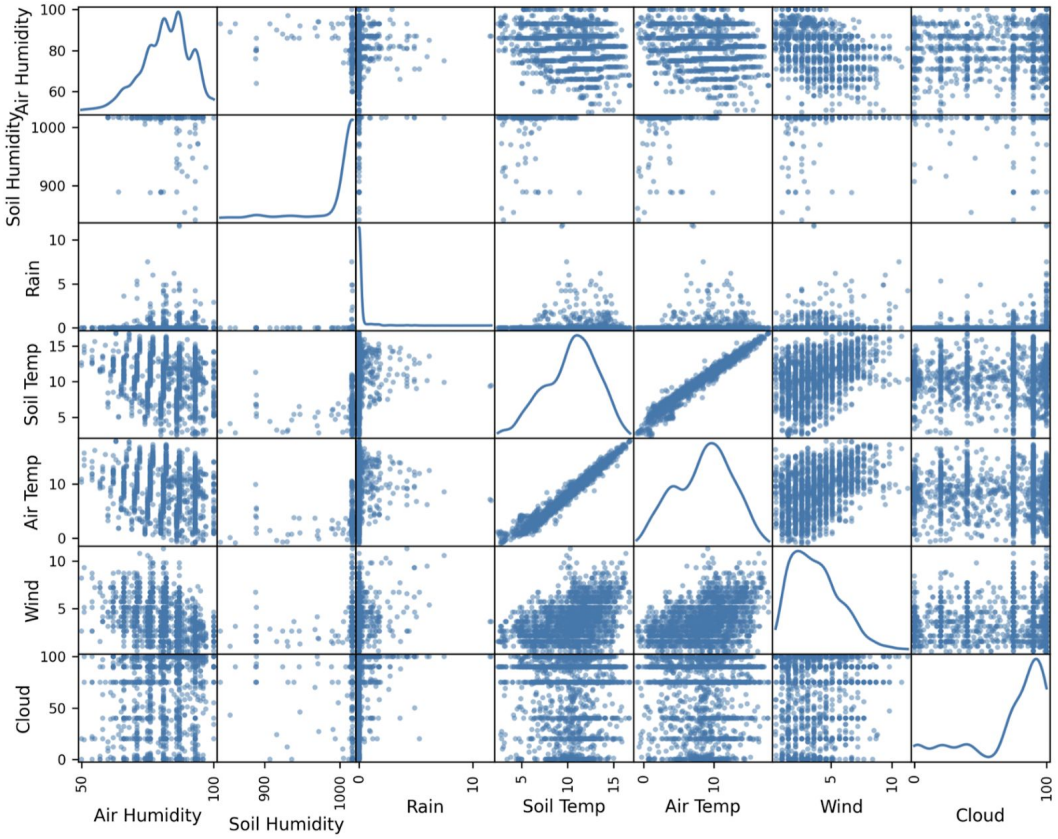


Figure 10: Correlation matrix scatter plot

Discussions on the important aspects of the project

Over the duration of the project, a number of key targets were achieved, and many new software, hardware, and statistical skills were developed including:

- Object-oriented Python programming
- Use of Docker for application containerization
- HTTP and LoRa communication protocols
- Use of the Firebase backend platform for data collection and WebApp hosting
- Coding in HTML, CSS, and JS for WebApp creation
- Data cleaning, processing, and correlation analysis using pandas
- Deep neural net training with Keras and Tensorflow

Avenues for future work and potential impact

While the system performed well with the limited dataset and sensor data, many improvements can still be made to develop a more reliable autonomous water-saving irrigation system.

Primarily, better quality soil humidity sensors must be used to gain reliable measurements of local soil humidity. Reliable soil humidity data will enable a feedback system, validating the model's predictions and improving the model with time.

Furthermore, extra sensor nodes can be placed in crop fields to gain further insight into soil water distribution.

Additionally, cameras can be placed above certain plants in order to gauge plant health, this can be done in two ways: leaf image classification and spectral reflectance measurement. Leaf image classification will employ a Convolutional Neural Network (CNN) to identify whether plant leaves appear healthy, dry, or infected [17] [18]. Spectral reflectance will use IR+RGB cameras to measure leaf reflected wavelengths and identify the state of the leaves [19] [20]. Healthy leaves reflect IR and green light while absorbing red and blue wavelengths, deviations in the leaf spectral reflectance could indicate issues with plant health.

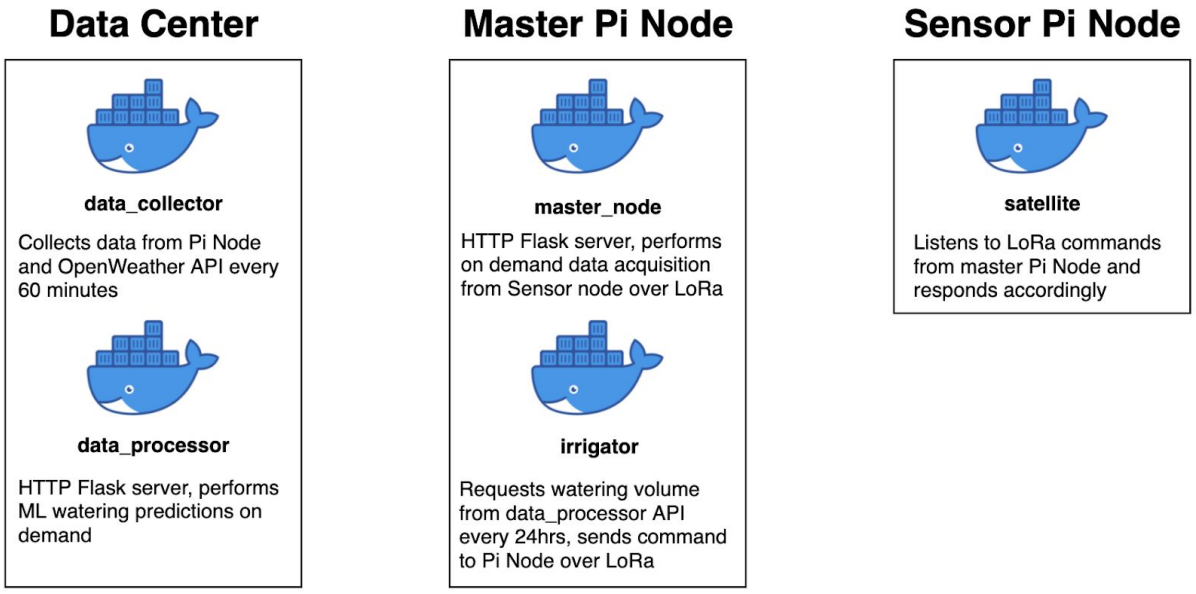
Finally, this IoT solution could be employed in an enterprise setting, on large scale crop fields. A central control system would poll data from multiple soil nodes and local meteorological stations to make accurate predictions in watering patterns as well as forecast plant watering based on weather forecasts, this would ensure that farmers are never met with an unexpected water shortage during a drought. The vision system could also be used to monitor plant growth and recommend fertilizer use in a similar fashion to water.

References

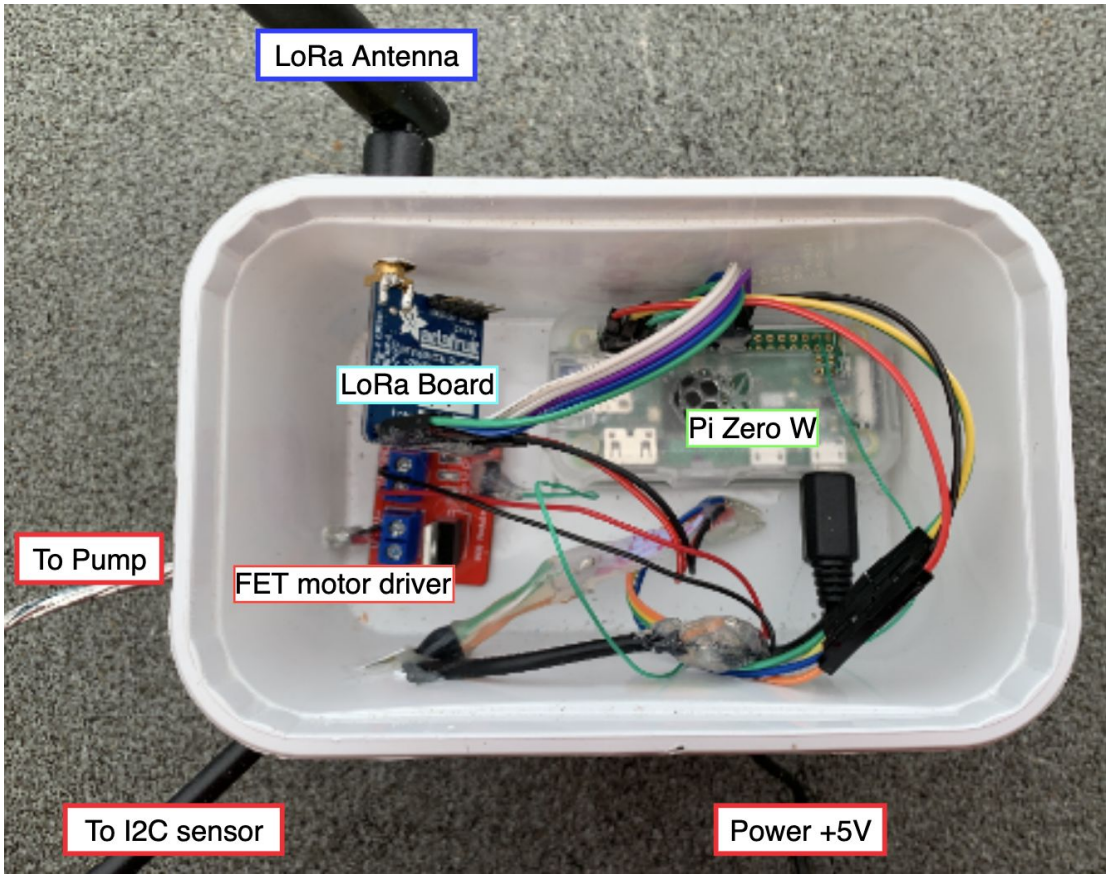
1. EPA. (2018). When it's Hot. [online] Available at: https://www.epa.gov/watersense/when-its-hot [Accessed 11 Jan. 2020].
2. Notion. (2021). All-in-one workspace. [online] Available at: https://notion.so [Accessed 11 Jan. 2020]
3. USGS. (2019). Evapotranspiration and the Water Cycle. Available at: https://www.usgs.gov/special-topic/water-science-school/science/evapotranspiration-and-water-cycle?qt-science_center_objects=0#qt-science_center_objects [Accessed 11 Jan. 2020]
4. Villalobos F.J., Testi L., Fereres E. (2016) Calculation of Evapotranspiration and Crop Water Requirements. In: Villalobos F., Fereres E. (eds) Principles of Agronomy for Sustainable Agriculture. Springer, Cham. https://doi.org/10.1007/978-3-319-46116-8_10
5. FAO. (2018). Evapotranspiration process. [online] Available at: http://www.fao.org/3/x0490e/x0490e04.htm [Accessed 11 Jan. 2020]
6. Bewaterwise. (2017). Watering index. [online] Available at: https://www.bewaterwise.com/assets/watering-index.pdf [Accessed 11 Jan. 2020]
7. OpenWeather. (2020). Current weather data. [online] Available at: https://openweathermap.org/current [Accessed 11 Jan. 2020]
8. ThePiHut. (2020). Raspberry Pi Zero W. [online] Available at: https://thepihut.com/collections/raspberry-pi/products/raspberry-pi-zero-w [Accessed 11 Jan. 2020]
9. ThePiHut. (2020). Adafruit RFM69HCW Transceiver Radio Breakout - 868 or 915 MHz. [online] Available at: https://thepihut.com/products/adafruit-rfm69hcw-transceiver-radio-breakout-868-or-915-mhz?variant=27740302289 [Accessed 11 Jan. 2020]
10. ThePiHut. (2020). Adafruit MCP9808 High Accuracy I2C Temperature Sensor Breakout Board. [online] Available at: https://thepihut.com/products/adafruit-mcp9808-high-accuracy-i2c-temperature-sensor-breakout-board?variant=27739616977 [Accessed 11 Jan. 2020]
11. ThePiHut. (2020). Adafruit STEMMA Soil Sensor - I2C Capacitive Moisture Sensor. [online] Available at: https://thepihut.com/products/adafruit-stemma-soil-sensor-i2c-capacitive-moisture-sensor-ada4026?variant=18634592485438 [Accessed 11 Jan. 2020]
12. Adafruit. (2016). Adafruit RFM69HCW and RFM9X LoRa Packet Radio Breakouts. [online] Available at: https://learn.adafruit.com/adafruit-rfm69hcw-and-rfm96-rfm95-rfm98-lora-packet-padio-breakouts/assembly [Accessed 11 Jan. 2020]
13. RS. (2021). Pycom Universal Antenna Kit for use with FiPy, LoPy, SiPy. [online] Available at: https://uk.rs-online.com/web/p/development-tool-accessories/1259535/ [Accessed 11 Jan. 2020]
14. Plotly.js. (2021). JavaScript Figure Reference: Single-Page. [online] Available at: https://plotly.com/javascript/reference/ [Accessed 11 Jan. 2020]
15. Amazon. (2021). DC 6V Peristaltic Pump, Professional Lab Aquarium Micro Peristaltic Pump for Biochemical Analysis Pharmacy(#3). [online] Available at: https://www.amazon.co.uk/dp/B08424Q1F6/ref=cm_sw_em_r_mt_dp_Ht1-FbYPH82AX?_encoding=UTF8&psc=1 [Accessed 11 Jan. 2020]
16. Amazon. (2021). HALJIA IRF520 MOS FET MOSFET Driver Module Compatible with Arduino Raspberry Pi ARM MCU). [online] Available at: https://www.amazon.co.uk/gp/product/B06XB9B16Z/ref=ppx_yo_dt_b_asin_title_o02_s00?ie=UTF8&psc=1 [Accessed 11 Jan. 2020]
17. Sharada P. M., David P. H., Marcel S. (2016) Using Deep Learning for Image-Based Plant Disease Detection. In: Frontiers in Plant Science. https://doi.org/10.3389/fpls.2016.01419
18. Srdjan S, Marko A, Andras A, Dubravko C, Darko S, "Deep Neural Networks Based Recognition of Plant Diseases by Leaf Image Classification", Computational Intelligence and Neuroscience, vol. 2016, Article ID 3289801, 11 pages, 2016. https://doi.org/10.1155/2016/3289801
19. Huymboldt. (2017). Reflectance. [online] Available at: http://gsp.humboldt.edu/OLM/Courses/GSP_216_Online/lesson2-1/reflectance.html [Accessed 10 Jan. 2020]
20. Digital @ Dai. (2017). Remote Sensing Part 3: Identify Healthy Vegetation From Space. [online] Available at: https://dai-global-digital.com/lush-green-remote-sensing.html [Accessed 11 Jan. 2020]

Appendix

1 All docker containers running in IoT project



2 High res image of satellite Pi Node



ML model architecture

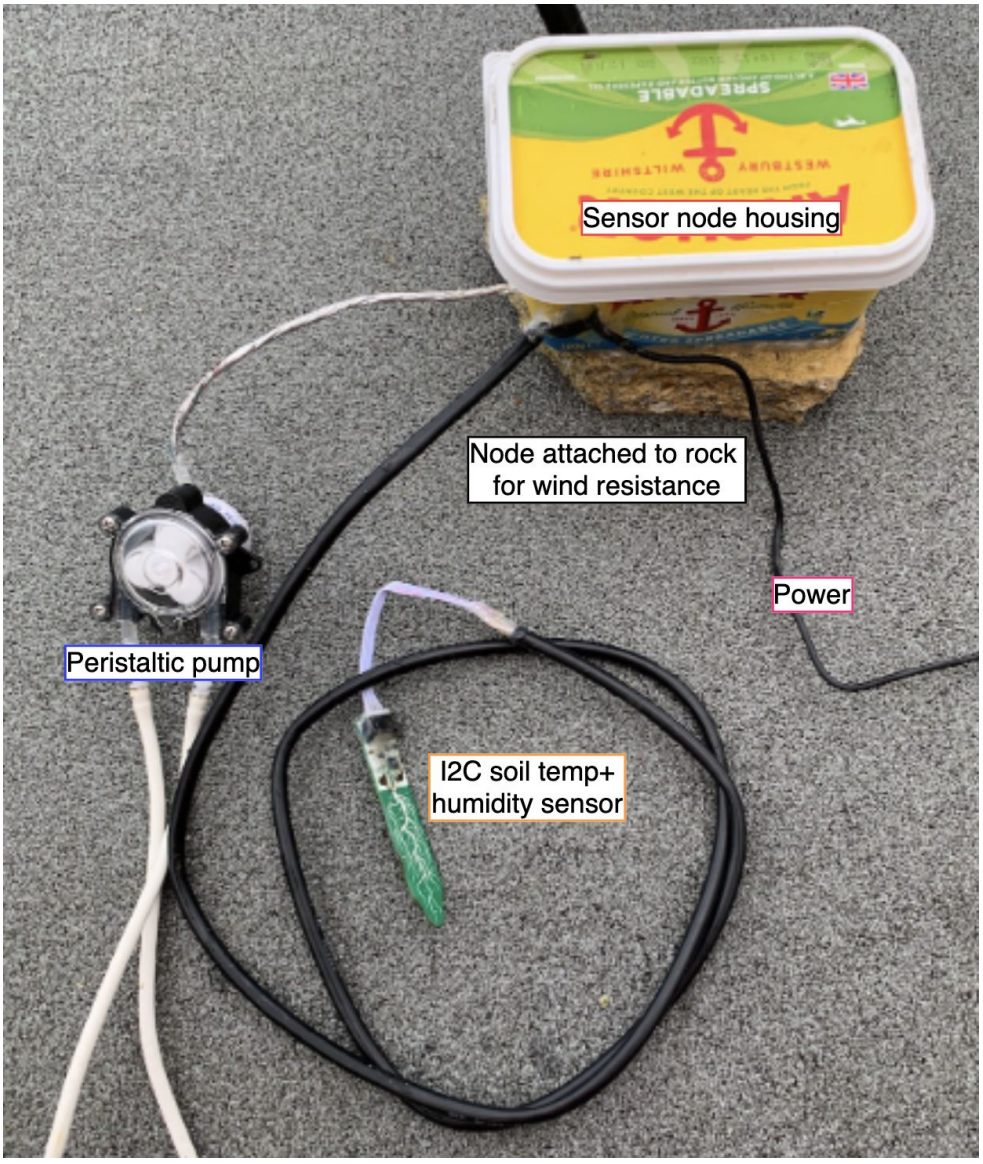
DE4-SIoT - model.py

```
model = tf.keras.models.Sequential()
model.add(feature_layer)
model.add(tf.keras.layers.Dense(units=14, activation='relu',
                                kernel_regularizer=tf.keras.regularizers.l2(l=0.05)))
model.add(tf.keras.layers.Dense(units=7, activation='relu',
                                kernel_regularizer=tf.keras.regularizers.l2(l=0.05)))
model.add(tf.keras.layers.Dense(units=1))

model.compile(optimizer=tf.keras.optimizers.Adam(lr=0.005),
              loss="mean_squared_error",
              metrics=[tf.keras.metrics.MeanSquaredError()])

```

Sensor node external components



API Endpoints

DE4-SIoT - api_endpoints.py

```
__NODE_TEMPERATURE_ENDPOINT = "http://ss.maxhunt.design:3333/temp"
__NODE_HUMIDITY_ENDPOINT = "http://ss.maxhunt.design:3333/hmdt"
__WATER_PREDICTOR_ENDPOINT = "http://api.maxhunt.design/water"

```