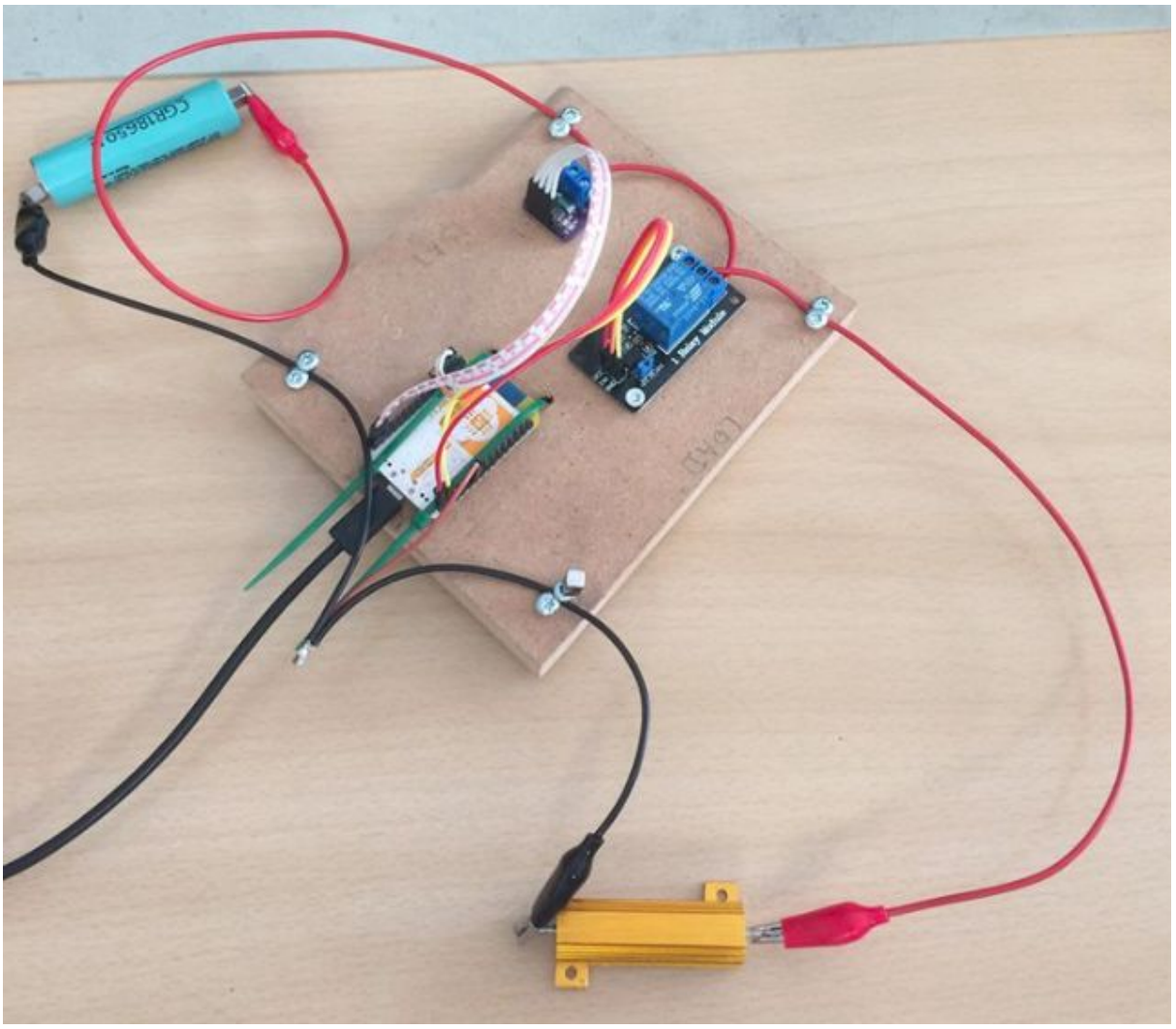
IoT Battery tester

An IOT platform for measuring battery capacity employing the ESP8266 WiFi module, an MQTT broker and a cross platform web interface.



Name	Student ID
Anders Greve Reiche	201604873
Jens True	20081266

Project span: Week 38 - Week 50
Project handin: 2018-12-14

Resume

This report covers the 5th semester IOT project autumn 2018 at the EDE education in Herning. The project consists of this report, code and resources available online on GitHub and a working assembled prototype. To understand the project requirements and limitations, it is recommended to read the original course description accessible on the Blackboard Online Learning Platform.

This project aims to create a useful testing tool for measuring battery capacity. The device will automatically send data to the internet and allow the user from any modern browser to see the logged data and export it for further analysis.

Throughout this document, references to sources will be handled directly in the footnotes on the corresponding pages.

This documentation along with supporting code is publicly hosted on the github repository associated with this project[1]. It is recommended for the reader to clone this repository as the documentation and code compliments each other.

Project description

Problem domain

Over the later years the world has seen an incredible advance in battery technology. Everything from keyring flashlights to passenger ferries now contains batteries. These batteries all varies in size, intended use and quality and it is such not easy for a normal user to estimate just how much capacity her/he has. Regular cruising sailboats carry 12V lead acid battery to run instruments, navigation lights, interior lights etc. Many of these batteries are old and see very little maintenance. The owner will connect the charger whenever he/she docks the boat. Cruises from one harbor to another are often quick and almost always done in daylight. This means the batteries see very little discharge. When the owner goes on a longer trip they will often discover that their old batteries no longer have the original capacity. This can result in dangerous situations when the battery discharges and navigation instruments and other necessary equipment stops working.

Specifications differs wildly from battery to battery.[2] Therefor testing is necessary

Project

A simple device is suggested which allows the operator to connect any given load to a battery. After configuring your WiFi settings in the device the device connects to the internet.

[1] https://github.com/furyfire/espbatterytester
[2] https://batteryuniversity.com/learn/article/discharge_characteristics_li

The device is now controlled and configured through a normal webbrowser. Through the web interface user is now able to set a few parameters before starting the measurement process. These parameters include.

- CutOff Voltage
    - Voltage at which the load will automatically be disconnected from the battery
- Maximum duration
    - A custom timer after which the load will be disconnected
- Measurement interval
    - Time between individual measurements

The idle voltage can be monitored and when the user is ready he can enable the relay and the device will record battery voltage, current and accumulated ampere hours to the internet. This data is visible through the webinterface, allowing the user to see the complete discharge graph and export the data in an open format. After a fixed duration or after the battery voltage drops below the specified voltage, the load is disconnected from the battery again, to protect against damage to the battery cells.

Storytelling

1. Disconnect the battery
2. Connect device in between the battery and the existing system
3. Load up the web interface
4. Set the cutoff voltage, measurement interval and battery type
5. Start the measurement
6. Wait for the notification telling measurement has completed.
7. Read out the Ah and total running time
8. Keep the browser window open for discharge curve

Requirements

- The device shall be an IOT device using WiFi
- The device shall log voltage and current to the internet
- The web interface shall allow the user to toggle a relay on the device
- The web interface shall show a graph showing the battery voltage over time
- The web interface shall show the accumulated Ah
- The web interface shall allow resetting and setting a cutoff voltage at which the relay is disengaged.
- The device will send a notification to the user when the relay shuts off
- The entire system shall not rely on any 3rd services

Microcontroller considerations

Boards	Price	Deep Sleep power consumption	I/O Output current	Operating power consumption	Licensing
NodeMcu ESP8266	<4 €	77uA	12mA	70mA (200mA peak)	Open source
Particle Photon	16.40 €	80uA	25mA	80mA (235mA peak)	Closed Core
Raspberry Pi Zero W	12 €	N/A	>50mA	120mA	Open source

As the battery tester will be mostly idle, the power consumption is not of much importance, as long as it stays reasonable. The Particle Photon comes with some ease of use advantages, but it is not necessary for the project. The price point together with the licensing is what makes NodeMcu ESP8266 the obvious choice for this project.

Current sensor considerations

Sensor	Price	Accuracy	Analog input range	Operating voltage	Operating current	Measures Voltage	Measures Power	Requires additional circuitry
INA219	<1 €	1%	0-26V	3-5.5V	1mA	yes	yes	no
INA209	3.5 €	0.1%	0-36V	2.7-5.5V	0.33mA	yes	yes	no
INA210	1.4 €	1%	0-26V	2.7-26V	0.1mA	no	no	yes
ACS712	3.70 €	1.5-4%	------	4.5-5.5V	6-11mA	no	no	yes

Based on the above parameters, it seems most reasonable to chose the INA219 sensor for this project, given that we don’t need measurement precision beyond 1%, and we do not want to have to build external circuits around the sensor if it can be avoided.

Hardware interface analysis

The specific development platform used in this project is a NodeMcu ESP8266 which operates on an input voltage range of 4.5 - 10V
It is a 3.3V microcontroller, this means that we cannot receive sensor data above 3.6V or the chip might malfunction. However, we can still drive 5V logic with the 3.3V output, as TTL describes a logical 5V high as anything more than 2V. The powerful 12mA outputs of the ESP8266 microcontroller, makes it possible to drive small electric control circuits without the need of an amplifier.
The ESP8266 has 17 GPIO pins, of which 11 are directly usable without major reconfiguration all of which are pulled down internally.
The ESP8266 has a single analog input that ranges from 0-1V, however the NodeMcu board provides an onboard resistive voltage divider that shifts the range from 0-3.3V

Other interfaces available on the ESP8266 are: Serial, I²C and SPI.

The batteries to be measured is marine batteries. These will range from 0V to 24V depending on their size and condition, the hardware designed to measure these must therefore work in the full range of the batteries.

The sensor used to monitor the battery current is an INA219 current/power monitor which has an analog input range from 0-26V.
The sensor uses an I²C interface to report current, voltage and power.
The sensor uses a 3-5.5V power source as input, and draws only 1mA which is suitable for a NodeMcu ESP8266 pin without mich interfacing needed.

The actuator used in this project is a simple 5V controlled relay, capable of switching a 24V battery power rail.

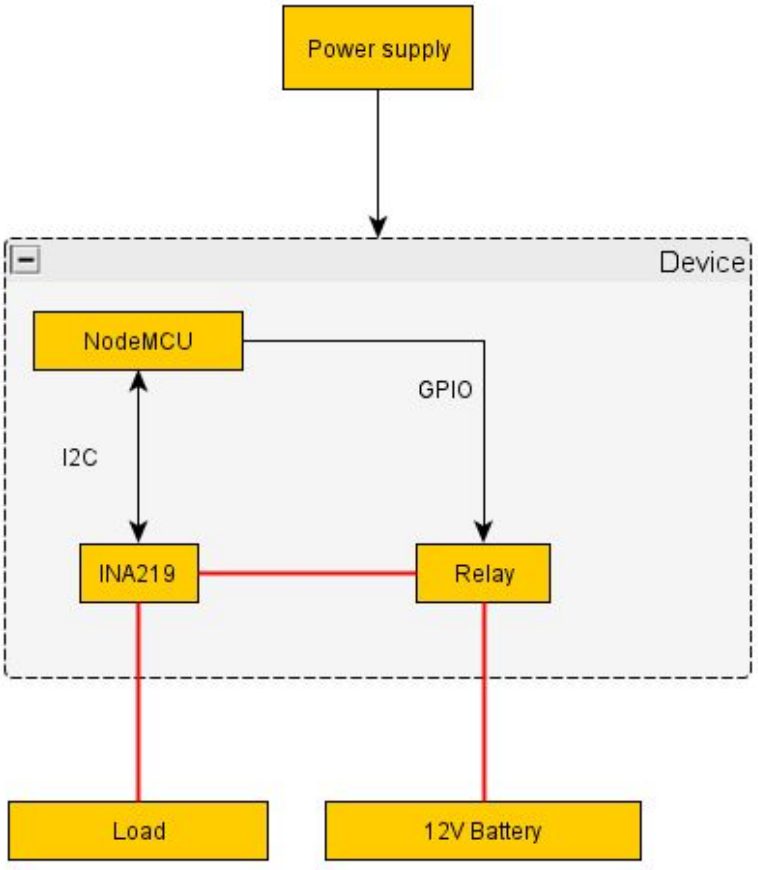


Figure x.x - Physical architecture diagram. Red connections showing high current lines from battery to load.

Software interface analysis

The software will consist of three main parts.

- Embedded

    - Should perform the periodic voltage and current measurement and control the relay accordingly.
    - The device firmware will be based on the Homie IOT framework for the ESP8266 platform which natively uses MQTT as the main method of server communication
- Webinterface
    - A modern javascript enabled webinterface that allows the user to control the device and inspect the measured values in tabular and graphical format.
    - MQTT can be tunneled over WebSockets to allow for direct communication from the client browser to the MQTT server.
- MQTT Server
    - The MQTT broker exposed to the internet accepting connections from both devices on the standard MQTT port and accepting connections from a client browser on port 8883 for MQTT connections tunneled over WebSockets.

This separation ensures logic will only be placed in two places, embedded and the webinterface. The MQTT server will not contain any logic and is only responsible for storing and forwarding communication between the system actors.

Figure x.x.x - Interface analysis

User interface

During the design face of the user interface a mockup sketch has been made showing the proposed interface. The user interface should be simple to use, with a focus towards performing the basic device operation and exporting data to whatever other form of processing the user wants. The interface should work on a standard modern browser and at minimum fulfill the project requirements.

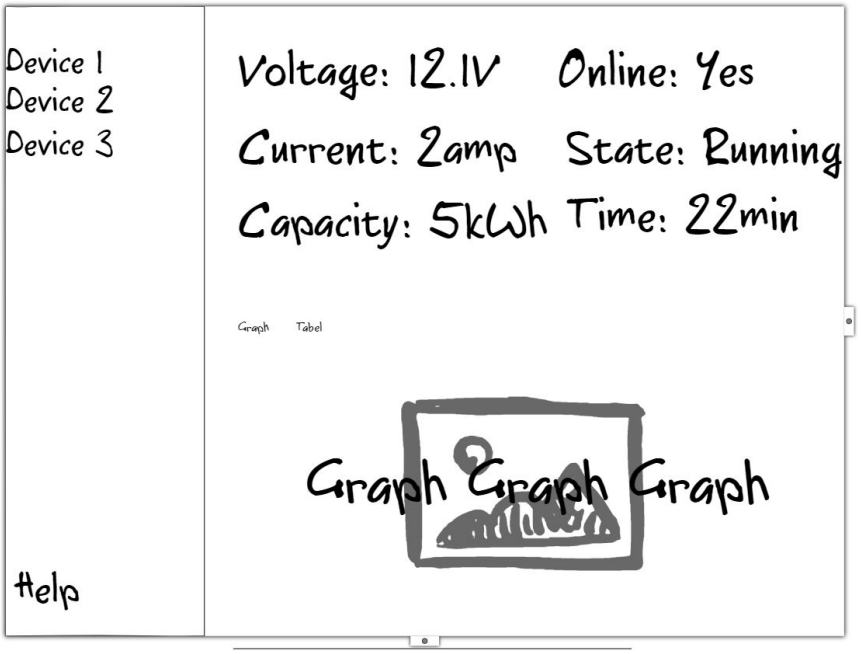


Figure x.x - User interface sketch.

Software design

Embedded

Homie IOT framework for ESP8266
To ensure quick development of the embedded system an IoT framework compatible with the ESP8266 microcontroller will be used.
“The Homie Convention” is a lightweight IoT protocol / convention using the MQTT protocol. “Homie-esp8266” implements this protocol on the ESP8266 microcontroller. This project handles WiFi configuration, firmware updates, configuration etc as well as providing a development platform for implementing custom code. For this project the homie-esp8266 package will be utilized.[3]

Platformio
For development the platformio toolchain for the ESP8266 will be used. Platformio is a build environment supporting more than 550 different development boards. It allows automatic installation of toolchains and framework through the use of simple project configuration files. This ensures a development environment can easily be recreated for other developers or for the purpose of continuous integration.[4]

Web Interface

The web interface will be built using standard web development tools. All communications from the device to the interface will be performed over the MQTT protocol. Since browsers

[3] https://github.com/marvinroger/homie-esp8266
[4] https://platformio.org/

does not natively support MQTT the transport stream will be wrapped over WebSockets. This allow the client browser to directly connect to the MQTT server for receiving and sending commands. Javascript will be used to argument the users interaction with the site. For plotting a Javascript charting library will be used. This allows for graphs to be rendered on the client side.
Following javascript libraries will be used. All selected based on previous experience with web development.

- Chart.js[5]
    - Library for drawing charts in the browser
- Eclipse Paho JavaScript Client[6]
    - Javascript implementation of the MQTT protocol transported over websockets
- jQuery[7]
    - Javascript library to simplify HTML manipulation and user interaction.
- Bootstrap[8]
    - Frontend HTML/CSS/JS library for graphic web interfaces.

MQTT

All communication to and from the device is handled via the MQTT protocol. This is an obvious choice as homie already implements a full MQTT client. This means that all that is needed for the full communication infrastructure to be in place, is an MQTT broker. Here there are two choices, an open broker such as iot.eclipse.org / hivemq or a self hosted solution The open platforms are easy to use but share some common downsides.

1. No control of the uptime of the service
2. Other users can interfere with the communication
3. Project dependent on 3rd party

The other option is a self hosted broker which enables us to control when the service is up and who is using it. We can even password protect the client authentication process it if we'd like to.
Mosquitto is one such broker that can be installed on a home server

OTA (over the air) updates are also handled via the MQTT protocol through the homie framework, this allows deployment of patches to the device moments after discovering a bug in the field.

Topics

The naming conventions for the topics will follow the homie iot convention.[9] Following nodes and properties will be defined

[5] https://www.chartjs.org/
[6] https://www.eclipse.org/paho/clients/js/
[7] https://jquery.com/
[8] http://getbootstrap.com/
[9] https://github.com/homieiot/convention

Node	Property	Unit	Description
measure	voltage	Volt	Current voltage measurement from the INA219 sensor
measure	current	Ampere	Current measurement from the INA219 sensor
measure	charge	Watt hours (Wh)	Accumulated watt hours consumed
measure	state	Enum: running, pause, stopped	State of the current measurement, settable
measure	cuttoff	Voltage	Cutoff voltage at which the relay will be disengaged. Settable.

```
homie/18fe34a28bcc/measure/state  stop
homie/18fe34a28bcc/measure/measurement  {"voltage":"0.89","current":"-0","charge":"0.000"}
```

Figure x.x - Sample mqtt messages

Hardware design

Battery measurement

The batteries will be measured with the IoT device in series with a load. The device will sit in the loop for however long it takes to discharge the battery to 50% of the label voltage. The devices takes a measurement with a fixed interval and publishes the measurement on the MQTT broker. During this process the battery is continuously discharged with an ohmic load, this means that we won't have a constant discharge current. This is why we also measure the voltage, so we can calculate AH (ampere/hour). This will only be an estimate though, as we can see in figure x.x, the AH is in relation to the current.

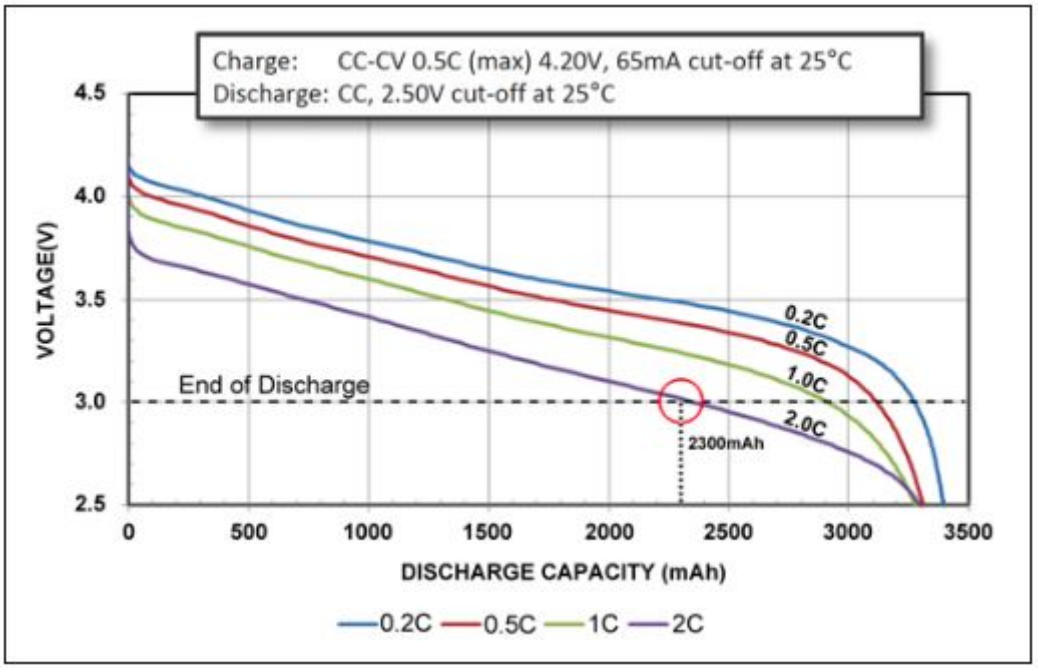


Figure x.x - Discharge characteristics of NCR18650B Energy Cell by Panasonic.[10]

The hardware will therefore also be made in such a way, that the users can easily decide his or her load and connect it themselves. This allows for very flexible use over a wide range of battery sizes and loads.

Assembly

The project will be made by hardwiring components and mounting them inside a small box or on a surface with two wires coming out of the front for connecting the load, and two wires coming out of the back for connecting the battery to be tested. This is not the focus of the project, but merely a necessary step in order to test the software / hardware integration and will be kept as simple and low effort as possible to avoid taking time away from the focus areas.

Implementation

Hardware

The hardware is assembled and mounted on a small piece of wood with pen markers and color coding to indicate battery and load connections. The power for the logic boards is supplied from a 5V USB phone charger. This is a common household items in this day and age, and would make little sense to integrate the power supply directly in the prototype. This also allows the prototype to take advantage of the recent development in mobile power banks that can easily reach 10-15.000 mAh, allowing the prototype to be deployed in the field alongside a mobile wifi hotspot.

[10] https://batteryuniversity.com/learn/article/discharge_characteristics_li

The final prototype consists of the following hardware components:

- ESP8266/NodeMCU module
- Alligator clips
- INA219 module
- USB Power supply
- Banana plug terminals
- Relay

MQTT server

The MQTT server will be running on a locally hosted server through a docker container. It will expose a standard MQTT server on the default 1883 port and MQTT over websockets on port 8883. The Mosquitto MQTT server will be used since it serves as the defacto open source implementation of MQTT on Linux. Furthermore it supports connections over WebSockets natively which is a requirement with the current design proposal.
The configuration file for mosquitto along with the docker file is located in the git repository under the “mosquitto” folder. On public internet facing server the mosquitto server can be launched by executing “docker-compose up”. This will expose port 1883 and port 8883. Configuration of a domain name and static external IP will not be covered by this documentation. During development the MQTT server is hosted on the domain jcktrue.dk running Debian 9 with the latest version of Docker installed.

```
1   # Config file for mosquitto
2   #
3   # See mosquitto.conf(5) for more information.
4   log_dest none
5   max_queued_messages 200
6   message_size_limit 0
7   allow_zero_length_clientid true
8   allow_duplicate_messages false
9
10  listener 1883
11  persistence false
12  allow_anonymous true
13
14  listener 8883
15  protocol websockets
```

Figure x.x.x - MQTT configuration file

```
version: "3"

services:
  mosquitto:
    image: eclipse-mosquitto
    restart: always
    container_name: mosquitto
    volumes:
      - "./mosquitto.conf:/mosquitto/config/mosquitto.conf:rw"
    ports:
      - "1883:1883"
      - "8883:8883"
```

Figure x.x.x - Docker compose file configuring Mosquitto

Embedded software

The embedded software is available in git under the folder “embedded”. The build environment is platformio and a working installation is therefore required. For more information check out the platformio website https://platformio.org/.
The command “pio run -t upload” will download the relevant support tools, 3rd party requirements, build the firmware and upload it to a connected device over USB. Furthermore the configuration file specifying the WiFi SSID, Password and MQTT broker details is stored in the /data/homie/edit.config.json. This configuration should be modified according to the users environment and saved under config.json. The settings can then be uploaded to the device using the command “pio run -t uploadfs”.

File	Description
src/main.cpp	Main application logic
src/config.h	General project configuration header file
data/homie/config.json	Homie configuration file specifying WiFi details and MQTT broker config.
platformio.ini	Platformio configuration file

All main application logic is located in a single file. This file initializes the Homie IOT framework and supporting libraries needed for reading the INA219 sensor on the I2C bus.

loopHandler()

The loopHandler function is called from the main application loop when the device has successfully connected to the MQTT server. When connection to the server is lost the device automatically stops calling the loopHandler(). This is implemented through the Homie IOT framework.

```
void loopHandler() {
    if (millis() - lastMeasureSent >= MEASURE_INTERVAL * 1000UL || lastMeasureSent == 0) {
        double voltage = ina219.getBusVoltage_V();
        double current = ina219.getCurrent_mA();
        if (current_state == STATE_RUN) {
            accumulated += voltage * current*MEASURE_INTERVAL;
        }

        DynamicJsonBuffer jb;
        JsonObject& obj = jb.createObject();
        obj["voltage"] = String(voltage, 2);
        obj["current"] = String(current, 0);
        obj["charge"] = String(accumulated / 3600, 3);

        String output;
        obj.printTo(output);
        Homie.getLogger() << "New Measurement: " << output << endl;

        measureNode.setProperty("measurement").send(output);

        if (voltage < cutoff_voltage) {
            Homie.getLogger() << "Cutoff Reached!" << endl;
            current_state = STATE_STOP;
            measureNode.setProperty("state").send("stop");
            digitalWrite(D4, HIGH);
        }
        lastMeasureSent = millis();
    }
}
```

Figure x.x - loopHandler() logic

Every 10 seconds a measurement is taken from the INA219 sensor. If the current state is running the accumulated energy measurement is updated. The data is then packaged into a JSON object which is then sent to the MQTT server under the property “measurement”. Finally the voltage level checked and the relay turned off if this is under the cutoff limit.

setStateHandler()

setStateHandler is a callback function called whenever data is sent to the topic /measure/state/set. This allows the user to change the device state between the 4 main states. When the state changed the change is propagated back to the user on the /measure/state topic. This allows the interface to always reflect the actual state set on the device.

State	Description	Relay state
STATE_STOP	Device is waiting to start a new measurement.	OFF

	Accumulated counter is 0.	
STATE_RUN	Device is running, counting accumulated energy.	ON
STATE_PAUSE	Incrementing the accumulated energy counter is temporarily paused until measurement is either resumed or stopped.	OFF
STATE_DONE	The voltage measurement has reached the cutoff limit and counting has stopped. A new measurement can now be started.	OFF

```
bool setStateHandler(const HomieRange& range, const String& value) {
  if (value != "stop" && value != "run" && value != "pause") return false;

  Homie.getLogger() << "Got new state: " << value << endl;
  if(value == "run") {
      if(current_state == STATE_STOP)
      {
          accumulated = 0;
      }
      current_state = STATE_RUN;
      measureNode.setProperty("state").send("run");
      digitalWrite(D4, LOW);
  }

  if(value == "pause") {
      current_state = STATE_PAUSE;
      measureNode.setProperty("state").send("pause");
      digitalWrite(D4, HIGH);
  }

   if(value == "stop") {
      current_state = STATE_STOP;
      measureNode.setProperty("state").send("stop");
      digitalWrite(D4, HIGH);
  }

  return true;
}
```

Figure x.x - setStateHandler() logic

setCutoffHandler

Similar to setStateHandler, setCutoffHandler handles the cutoff value at which the relay will be disengaged to project the battery from over discharging. The value will be evaluated during the next measurement cycle (each 10 seconds). The value is also sent back to the MQTT server.

```
bool setCutoffHandler(const HomieRange& range, const String& value) {
  if (value.toFloat() == 0) return false;

  Homie.getLogger() << "Got new cutoff voltage: " << value << endl;
  cutoff_voltage = value.toFloat();
  return true;
}
```

Figure x.x - setCutoffHandler() logic

Web Interface

The webinterface code is available in git under the folder “webinterface/public_html”.
The web interface is built from templates using the Bootstrap[11] framework. The framework sets up the index.html page and allows for pre built javascript features like the buttons and the folding table / chart sections to be implemented using a neat standardized method described further in the Bootstrap documentation[12]. The other way to do it would be to find the individual sub parts of the web page and hack them together or design them from the bottom up by hand.
To establish a connection to the MQTT server the javascript library Paho is used[13]. Paho provides a pure Javascript implementation of the MQTT protocol. Due to browser limitations only WebSocket connections can be established from within a standard configured browser environment. Therefor the MQTT datastream is tunneled over a WebSocket connection, Paho provides helper functions to facilitate this. All JavaScript logic is contained to the app.js file.

```
// Create a client instance
var client = new Paho.MQTT.Client(mqtt_url, Math.random().toString(36).substring(20));

// set callback handlers
client.onConnectionLost = onConnectionLost;
client.onMessageArrived = onMessageArrived;
// connect the client
client.connect({onSuccess:onConnect});
```

Figure x.x.x - Snippet from app.js showing Paho instantiation

Configuration parameters are defined in config.js. Here the mqtt server details are saved in the global variable mqtt_url. The webinterface in its current form only supports a single device. Therefore the deviceid of the used device is hardcoded into the “device_id” variable.

```
var mqtt_url = "ws://jcktrue.dk:8883/";
var prefix = "homie/"
var device_id = "18fe34a28bcc"
```

Figure x.x.x - config.js variables

[11] http://getbootstrap.com/
[12] http://getbootstrap.com/docs/4.1/getting-started/introduction/
[13] https://www.eclipse.org/paho/clients/js/

onMessageArrived()

Each incoming MQTT message will result in the onMessageArrived callback function being called. This function decodes the message and updates the UI accordingly. Following MQTT topics are handled in the function. All other messages are ignored. (Please note that the device_id prefix has been left out for readability in the following table.)

Topic	Value	Description
/$online	true/false	Indicates if the device is currently online and connected to the MQTT server.
/measure/state	Enum {run, pause, stop}	Indicates the current state of the device
/measure/cutoff	Float voltage	The current cutoff voltage set on the device.
/measure/measurement	JSON object containing voltage, current and charge fields	Latest measurement from the sensor packaged in a single JSON object.

The webinterface implemented as a proof of concept.

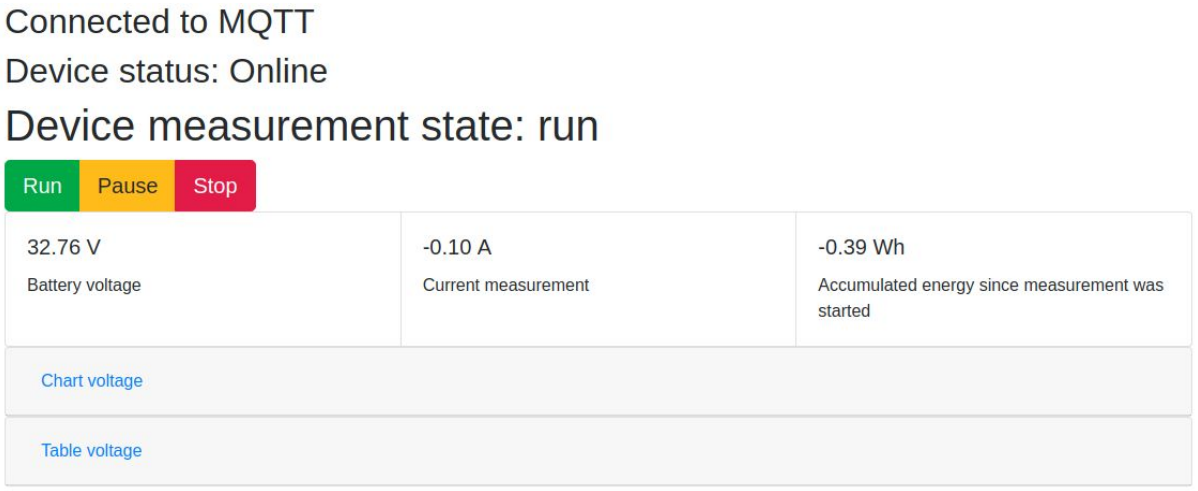


Figure x.x - Webinterface loaded

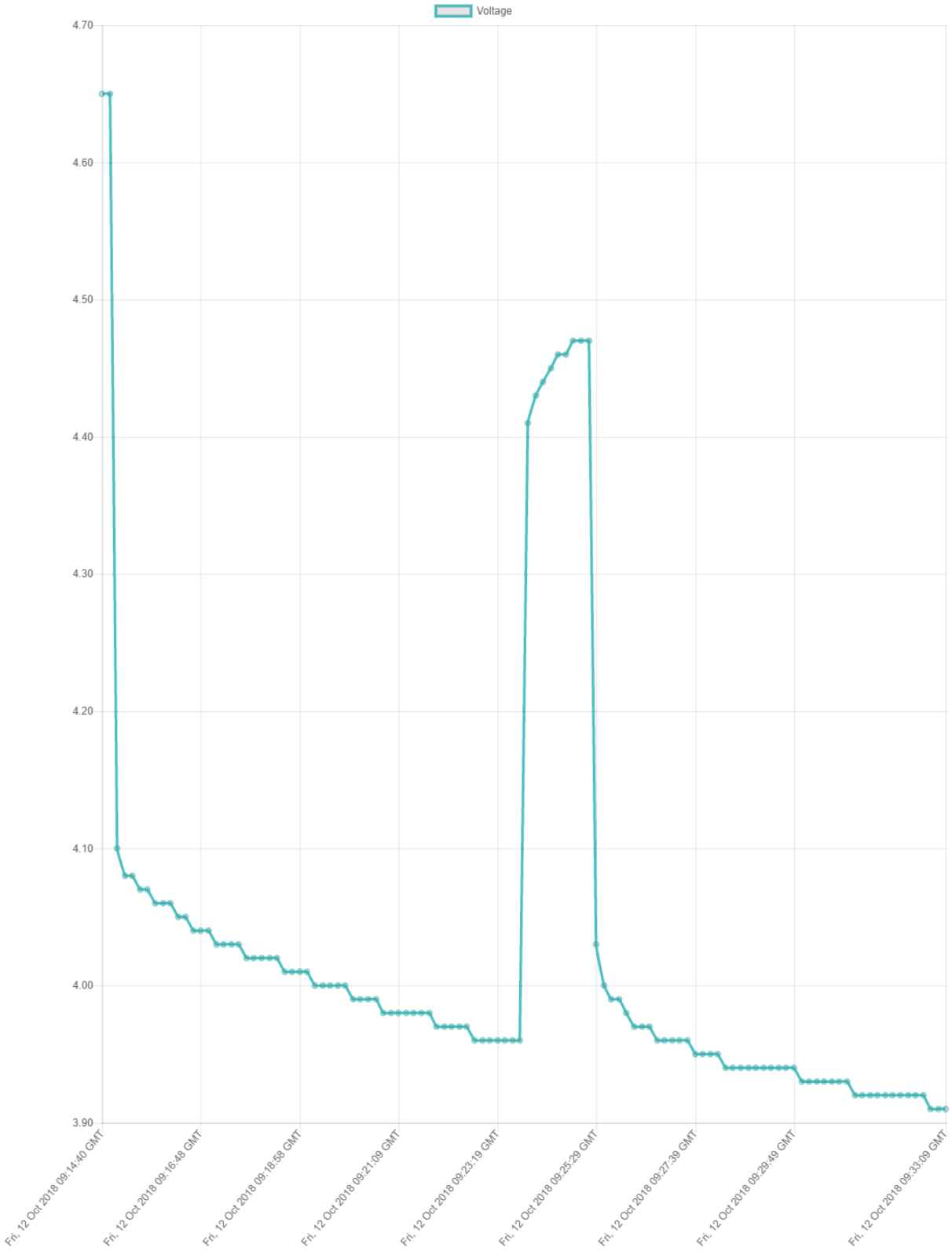


Figure x.x - Sample discharge curve on the web interface. The spike shows a pause in the measurement allowing the battery voltage to recover before reapplying the load.

Verification

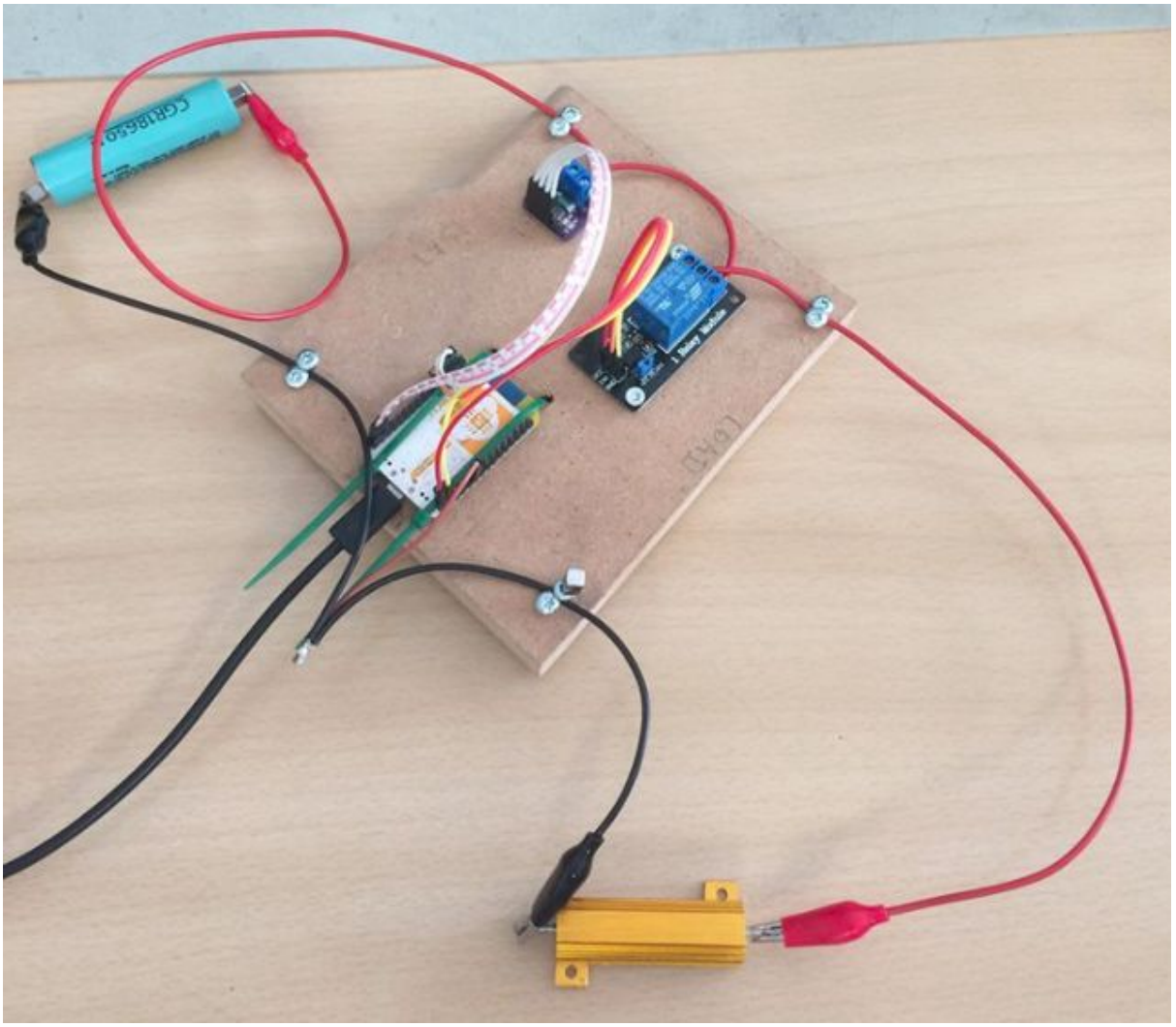


Figure x.x - Test setup

Figure x.x - Cutoff set to discharge the battery almost completely

486	Fri, 16 Nov 2018 12:13:16 GMT	3.22 V	392 mA	2123.375 Wh
487	Fri, 16 Nov 2018 12:13:26 GMT	3.21 V	391 mA	2126.866 Wh
488	Fri, 16 Nov 2018 12:13:36 GMT	3.21 V	391 mA	2130.350 Wh
489	Fri, 16 Nov 2018 12:13:46 GMT	3.21 V	391 mA	2133.833 Wh
490	Fri, 16 Nov 2018 12:13:56 GMT	3.20 V	390 mA	2137.307 Wh
491	Fri, 16 Nov 2018 12:14:06 GMT	3.20 V	390 mA	2140.775 Wh
492	Fri, 16 Nov 2018 12:14:16 GMT	3.20 V	390 mA	2144.234 Wh

Figure x.x - Measurement reaching cutoff

The web interface in conjunction with the embedded software and hardware meets the specified project requirements.

- The device shall be an IOT device using WiFi ✔️
    - The device connects to the WiFi net, specified in the config.json file on startup

- The device shall log voltage and current to the internet✔️
    - This is realized in the table and the chart on the web interface

- The web interface shall allow the user to toggle a relay on the device✔️
    - By using the “Run”, “Pause” and “Stop” buttons on the web interface, the user can directly manipulate the state of the relay.

- The web interface shall show a graph showing the battery voltage over time✔️
    - This is realized in the chart on the web interface

- The web interface shall show the accumulated Ah✔️
    - This is realized, however the unit is shown in mAh

- The web interface shall allow resetting and setting a cutoff voltage at which the relay is disengaged.✔️
    - This is realized with a slider on the web interface that goes from 0-32V in 100mV intervals

- The device will send a notification to the user when the relay shuts off❌
    - The device changes the state on the web interface to stop, indirectly notifying the user that the relay is off

- The entire system shall not rely on any 3rd services✔️
    - Everything used in the project is open source and can be downloaded and self hosted

The requirement about notifying the user is accepted as is, since the web interface is directly accessible by the end user. It gets a red cross as the requirement is not implemented exactly as originally stated in the requirements breakdown.

Improvements

- TLS encryption of the data communication stream.The hardware supports secure connections but it has not been implemented to ease the development process.

- A way to persist the measurements on the web server would be beneficial. Either in a .csv file which can be served to download, or in a database system. Currently data is only kept in the browser window for the duration of the session.

- A battery database with presets for the individual battery cutoffs so the user would not need much knowledge about batteries in order to use the device correctly. This way the user interface can also tell the user about the battery capacity decay % instead of just the mAh measurement.

- No power saving steps have been taken. With the high update rate on measurements and the desire to have an quick updating user interface, sleep routines have not been implemented. Furthermore the relay draws approximately 50mA a relative high amount compared to 80mA average current draw from the ESP8266 module. Powering down the ESP8266 module between measurement and reconnecting to WiFi would have limited effect on saving power.

- A cellular option would allow for measurements out of reach for normal WiFi networks. Using the cellular network could also simplify the user setup procedure. The device would simply work from the second the device is connected first time.

- A Web portal could be implemented allowing the user to manage multiple devices and batteries. Further data analysis could be performed over time where the relative performance of several battery banks could be visualized. This could be useful for modern “off the grid” houses relying solely on green energy or managing an electric car fleet.

Conclusion

The prototype has proven to be capable of delivering results that meet the initial expectations for the project. The idea of discharging a battery while measuring the time it takes has been successfully implemented, and using the INA219 sensor, the prototype is able to gather very precise readings during the discharge. This lays the foundation for a good mAh measurement, as a bad current and voltage measurement would accumulate over time and grow worse to the point where the mAh measurement would be worthless.

The project is 100% open source. This means that every component used in the project is completely downloadable and self hostable and can be deployed anywhere without dependencies on 3rd party companies having access to the data or information about the operation of the service.