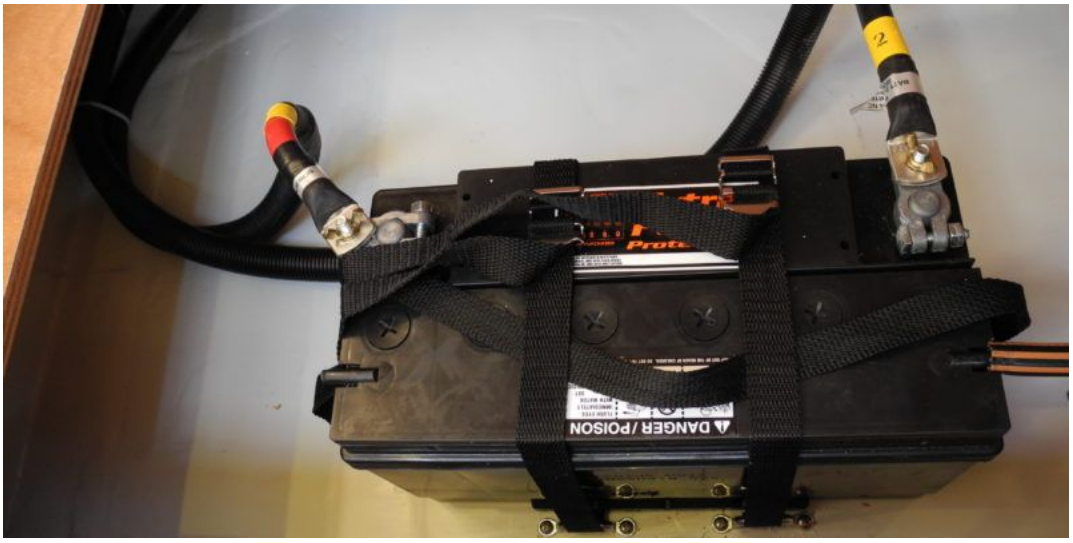
IoT Battery tester



Name	Student ID
Anders Greve Reiche	201604873
Jens True	20081266

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.

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.[1] 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. The device is now controlled and configured through a normal webbrowser. Through the webinterface 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

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

    - 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 webinterface shall allow the user to toggle a relay on the device
- The webinterface shall show a graph showing the battery voltage over time
- The webinterface shall show the accumulated Ah
- The webinterface 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

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.

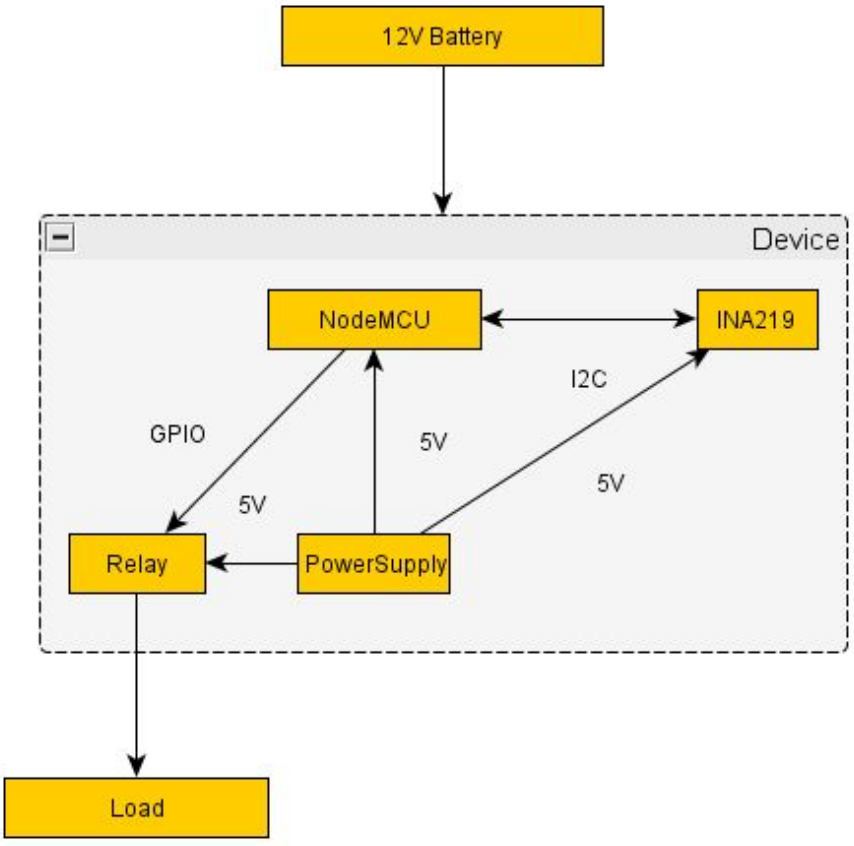


Figure x.x - Physical architecture diagram

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 Homie-esp8266 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 drawing

Software design

Embedded

Homie-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.[2]

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.[3]

Webinterface

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

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

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 webdevelopment.

- Chart.js[4]
    - Library for drawing charts in the browser
- Eclipse Paho JavaScript Client[5]
    - Javascript implementation of the MQTT protocol transported over websockets
- jQuery[6]
    - Javascript library to simplify HTML manipulation and user interaction.
- Bootstrap[7]
    - 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.[8] Following nodes and properties will be defined

Node	**Property**	**Description**
measure	voltage	

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

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.

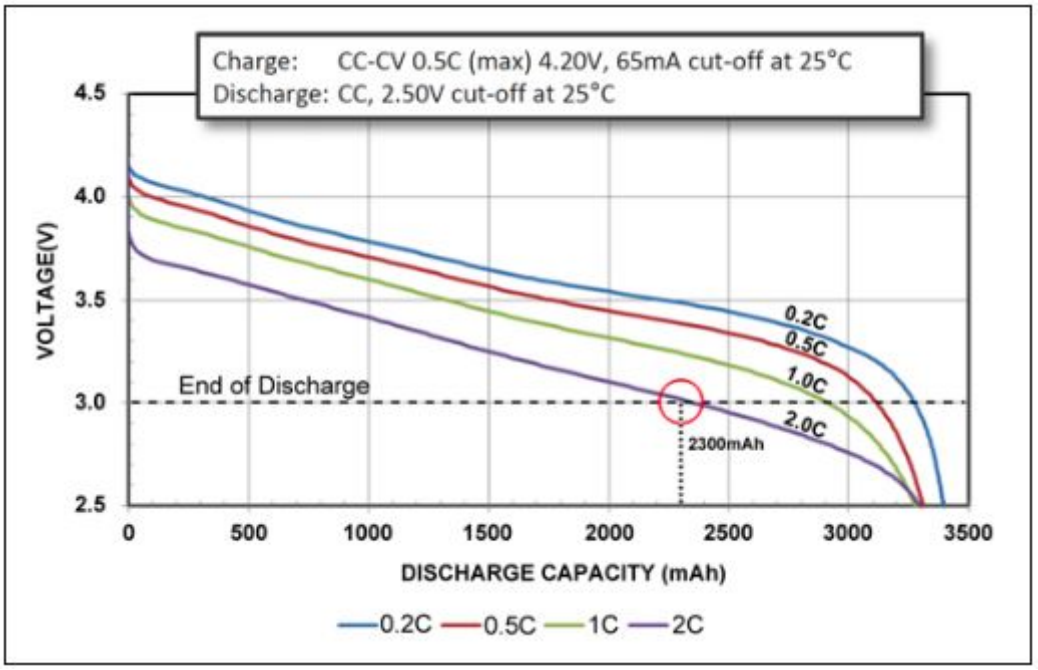


Figure x.x - Discharge characteristics of NCR18650B Energy Cell by Panasonic.

The hardware will therefore also be made in such a way that the users can easily decide the load and connect it themselves.