Amsterdam University of Applied Sciences

Powermonitoring Technical Documentation

Tom Buskens, Niels Arts, Cas van der Laan

June 21, 2019

About this Document

This document contains all of the technical documentation about the technologies used/created by the team. The document is split up in chapters, each named after their respective technology. This document will contain guides as to how the software packages can be set up, these guides can also be found on the projects GitLab page.

Contents

1 Ansible

1.1 Overview

Ansible is used as an automatic way of deploying machines, configuring machines and cloud provisioning in a scalable way.
In the Power monitoring project, Ansible is used to automatically deploy freshly installed Raspberry PI's with the necessary software packages. This iteration of this project Docker has been used to make reproducibility a lot easier. This way, only a few dependencies have to be installed using Ansible, making the install script easier to understand.

Ansible works via SSH, connecting to each individual slave from the host machine, and executing commands on them, similar to executing each command manually on each machine. Ansible does this by sending out small commands called "Ansible Modules" to the target machines. Ansible knows which machines to deploy using a simple inventory file. This file has to be included as a CLI option when executing the Ansible script.

Ansible has standard modules built in, like ping commands, but it also has a feature to make your own Ansible modules called Playbooks. This feature can be used to set up custom deploy scripts.

1.2 File structure

.gitignore
The gitignore file is used to leave out files that are not necessary to push to the git. Git will ignore this file when commiting changes.
README.md
The README file is used as a quick, but full, guide to set up and use Ansible. When a step in the project changes, this file has to be updated to match the current implementation.
install.yaml
This file is used by Ansible and contains all Ansible commands that need to be executed on the target machines.
inventory.example
This file is used by Ansible as a list of hosts. This file is included with the gitignore and only servers as an example. The end user has to update this file to match the IP's of the machines that the Ansible scripts have to be run on.
/scripts/ansible.sh
This script is uploaded and excecuted to each of the target machines. This script includes installs such as Docker and Python.

1.3 Installation/usage

1.3.1 Prerequisites

For Ansible to work, you'll need a host PC and some slaves to execute the Ansible scripts to. The host and slaves need to be accessible by SSH. This can be done by either connecting the systems to the same network using a switch

and ethernet cables or a wifi network. The slaves also need to be added to your `~/.ssh/know_hosts` file. Use this guide to find out how to do that. This can also be easily be achieved by first connecting to each slave before executing the Ansible script.

1.3.2 Install necessary software packages

To use Ansible, you first have to install Ansible by executing `sudo apt install ansible`. Check if Ansible is installed with `ansible -version`, the version installed has to be greater than 2.5.

1.3.3 Execute the Playbook

Clone this repository and navigate to the repository's folder. To the `inventory` file, add all the ip's of the pi's that need to be installed. Do this in this format: `rpi04 ansible_ssh_host=1.1.1.1`, as shown in the 'inventory.example' file. When this is done, execute the playbook using the command below.

```
ansible-playbook -i inventory -l cluster install.yaml
-i = inventory tag
-l = target group in the inventory file
```

Note: This may take a while when done on a slower system like a Raspberry Pi since it has fewer resources to work with for compiling the Python Libraries. Don't cancel any tasks, even if they take longer than half an hour.

2 INA219

This part of the project hasn't been changed a lot, just some improvements. The previous team made some documentation regarding their custom made carrier board for the INA219 sensor module. To give them credit for that, their documentation will be in the attachments down below, explaining everything you need to know about the sensor and its software.

The previous team used a proprietary port(8000) for Prometheus. To remove any complications with the metricbeats link to Prometheus, we changed it to the standard port(9090). Also, the program has been transfered to a Docker container. This makes it easier for the next user to set up. This is due to the automated required installations and the hassle free use of docker containers on different systems.

3 Architecture

3.1 Ansible

Ansible is a application that can automate your deployment and everything you can imagine in IT automation. It uses SSH to connect with device and runs playbooks. These playbooks have a human readable and machine-readable part. This is really useful to improve your integrability of your application (FAIR) The human readable part tells you what every step is doing in the process of your playbook. The machine-readable part Is a machine interpretable script for example a bash script. This is executed on every device. For more info about Ansible see https://www.ansible.com/.

We used Ansible for your cluster deployment. We created a playbook that runs on every pi defined in the hosts file. This playbook installs all dependency's including Docker and Docker-Compose. After this the our own created docker image is deployed via docker-compose.

Our Ansible Git repository: https://github.com/SoftwareForScience/power-monitoring/tree/master/2019-team/code/ansible

3.2 PXE boot (not finished)

Raspberry pi's have the ability to boot from the Network this is called PXE boot. This could be another solution to deploy our software and applications. We dived into this solation and created a Virtual machine with Raspbian X86. This OS have the ability to functionality as a master to boot all pi's connected to the network. Via this solution we wanted to deploy our docker containers on the raspberries via the Network boot. But this solution is not finished.
Link to our PXE boot can be found here: https://github.com/SoftwareForScience/power-monitoring/tree/master/2019-team/code/pxe

3.3 Docker

During This project Power Monitoring we used Docker. The reason for this is that the reproducibility of your project is really easy. You don't have to set up a specified environment with lots of different dependencies. This is already done for you inside this docker container. Beside the greater reproducibility with docker it also gives you higher scalability. This because docker is really easy to scale because you can deploy really fast on another cluster. We do this with Ansible but docker also provide Docker Swarms.

3.3.1 Docker images

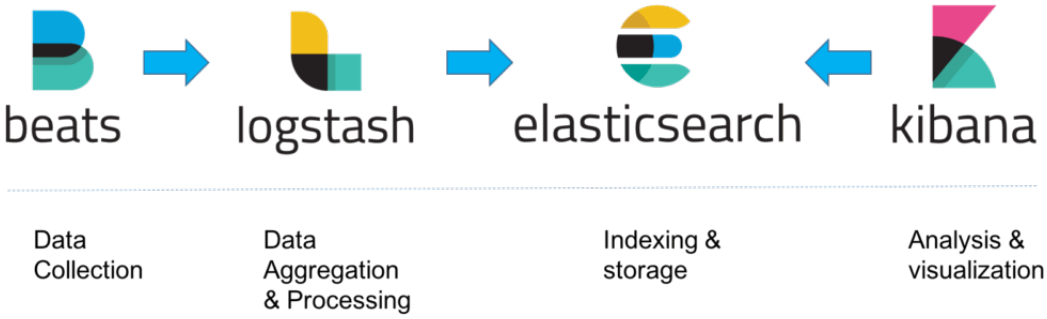
During this project we used a few docker images. Some of these images already exist and provided by the developers themselves, other images needed to be created by scratch. The Images we used by default are the images form the Elastic ELK Stack. These are the docker images Elasticsearch, Kibana, Logstash, OpenVPN. Images we created by Scratch are the Matricbeat, Prometheus with INA219 and the MD5 Hash load genartor.

The images Elasticsearch, Kibana and Logstash are images developed by Elastic. You can deploy these docker images by default on your own VPS. Because it works with docker it is really easy do get started without installing all dependencies separately. You can find the setup guide for Elastic with docker here https://github.com/elastic/stack-docker. You can Find the Images here https://www.docker.elastic.co/ The same for the OpenVPN Image is really easy to setup. You won't need it. But if you want to access your project across different networks we would recommend you to use it.
The Metricbeat image is an image build for the ARM 32 V7 architecture. This was needed because the default image from Metricbeat wasn't supported for this architecture. This image can be found here https://github.com/SoftwareForScience/power-monitoring/tree/master/2019-team/code/metricbeat-arm64-prome

3.3.2 Docker Compose

Docker Compose is used to pull and setup all your docker images in one setup file. This is done in a docker-compose.yml file. All variables for all defferent images can be defined here. With the known "Docker-Compose up -d" command. Your docker-compose.yml file can be find here https://github.com/SoftwareForScience/power-monitoring/blob/master/2019-team/code/elk-stack-docker/docker-compose.yml

3.4 Elastic



3.4.1 Elasticsearch

Elasticsearch from Elastic is an application that collects and indexed the incoming data form Logstash. It translates the data to a no SQL database which is really fast to do analysis with a lot of data. This is a really good advantage because in the radio astronomy there is a lot of data to collect and to analyse.

3.4.2 Kibana

Kibana is the visualization and analysis tool from Elastic that works with Elasticsearch. Kibana is a web app that has the look and feeling of a admin panel. In this panel you can create dashboards with different visualizations. Some of these are already pre setup for you and come with the packages that can be used with Elasticsearch.

3.4.3 Machine Learning

Elastic provide us a lot of features inside Kibana that uses Machine learning. Ed asked me to describe a few of these features. These main features are Automatically modelling complex data, detecting anomalies events inside your data, finding the properties that influenced the anomalies events and predicting the future (forecast).//
Automatically Modelling of your data. Elastic machine learning provides you a tool that is able to analyse your data and learn and model normal behaviour in your data. Detect Anomalies, because Elastic machine learning can learn normal behaviour it makes sense it can also detect anomalies behaviours in your data. Elastic goes a little bit further then detecting it. It also provides you a tool to Finding the Root Cause of your anomaly's behaviour. This means Elastic can give you more information about what properties influenced the anomaly behaviour. And Elastic gives you the ability to forecast/predicting the future of your data.

3.4.4 Logstash

Logstash is a pipeline data collector from Elastic. Logstash is a dynamic and scalable application that can collect from all different kind of sources on a multi distributed pipeline for example Elastics own Beats framework and transforms the data to it needs. Logstash can send these data to all different kind of stashes. Of course the best stash that works with Logstash is Elastic own stash Elasticsearch.

3.4.5 Metricbeat

One of these packages that works with Elasticsearch is Metricbeat and is a package from the application framework from Elastic called Beats. Metricbeat sits on your machine and monitors different processes within every heartbeat of your system. Default processes that are monitored by Metricbeat are our CPU, Memory and Network usage and more. These data are being send to Elasticsearch or Logstash as a metric that is generated every heartbeat of the system. Besides the default monitoring you can add modules to it. Some of these modules already exist for example Prometheus. Other modules can be developed by yourself.

3.5 Prometheus

Prometheus is an application aimed and designed to be used for monitoring time-based data metrics. It collects data metrics form metric-gathering clients provided by Prometheus. This application is used by the previous team.

We refactored this project into a Docker image. This makes it easier to reproduce it, because it's way easier to get this application up and running. You can find this image over here. https://github.com/SoftwareForScience/power-monitoring/tree/master/2019-team/code/prometheus-powermeter-build

This image connects to the Metricbeat module running on the same raspberry pi.

3.6 MD5 hash load generator

Part of this image is the application that create a data metrics for the INA219 current sensor that is used to measure the voltage and current of the raspberry pi's used in this project. This is one of the most important part of this project. For documentation about the PCB Design of the INA219 made by the previous team see: https://github.com/SoftwareForScience/power-monitoring/blob/master/Documentation/2018/InaDocu_update.pdf

In the application provided by the previous team the used a Phyton Client API library. This application provides the Prometheus server with the time-based data metrices.

3.7 Network Architecture

3.7.1 VPS

A Virtual Private Server is needed in this setup. This is because you need one central point to control all different raspberries from the clusters. You could see this setup as a Master Slave Architecture. This Architecture is also used with the elk stack. The master runs the ELK stack. The slaves with the Metricbeat running connects with the master.

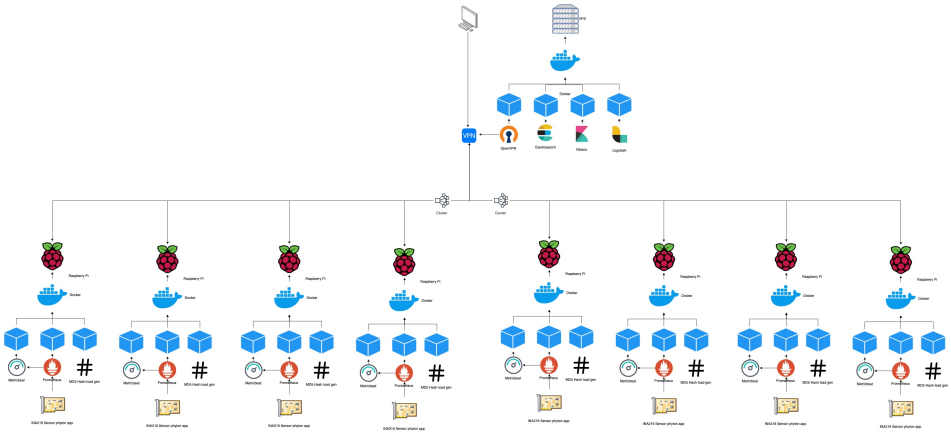
The VPS is also needed because of the resources needed for the ELK stack which a Raspberry pi cannot provide us. Beside that a VPS is always online so this is really handy when your using a VPN to connect all devices to the same network to let them communicate with each other.

3.7.2 Raspberry Pi Cluster

The Raspberry pi's are used in this project for test purpose but in fact these raspberry pi's could be any device. For example, a high and desktop. This can be useful when you want to do analyses on a very complex algorithm where you need more computer resources then a raspberry. This is possible because of the use of docker.

The raspberry pi's include a, PCB with an INA219 current sensor on top of it, a network switch to connect all the raspberry pi's

3.7.3 Overall Network Architecture



4 Documentation

4.1 Documentation

The teams documentation is set up in two ways. The first way is a paper containing an in depth view of the technical side of the documentation. The other way is guides written in Markdown for each of the individual parts of the project. Both ways of documentation can be used to set up the project yourself since they both feature a sufficient ammount of guiding.

4.1.1 Git documentation

The documentation on git is split up respectively to its project, this means that the Ansible repository contains the Ansible documentation and so on. In this way, we want to ensure that end users are able to find the documentation regarding the project and give a quick guide as to how to set up the repository.

The guides on git are set up to only contain the necessary steps. This is done to make it easy to read, thus improving the readability aspect of documentation. The decision to keep it simple was also made to just set up the repository, nothing more, nothing less, but they are also designed to guide people without any knowledge of the technology it self. This means that the guides on git are a complete guide to set up the project.

4.1.2 Technical report

The technical report was set up for people who want to get to know more about the end product. Each of the chapters in the technical report gives an in depth view of one of the components of the project. These in depth views contain an architectural overview if possible, and also contain the guide to set the component up.

Some of the components used are copied from one of the previous teams. In that case, the documentation they used will be refered to or attached to the document. The modifications made, if any, are described in the chapter about that component, while the documentation from the other team will be in the attachments in the end of the document.

4.1.3 Scientific Programming

For the Scientific programming course the team had to give a presentation about a research paper about a specific subject. Our topics turned out to be Architecture, Documentation and Testing. After the presentations were done, each student had to write a paper concerning their chosen topic.

Right now you're either reading the paper about Documentation or the technical documentation. The individual papers made by the team will also be included in the technical documentation.

4.2 Quality aspects

For the quality aspects of both the Git documentation and the technical report, I will refer to quality aspects used by the paper "Knowledge-based approaches in software documentation: A systematic literature review"[1]. The available quality aspects are listed below. Since there are a lot of them, only the relevant aspects will be discussed.

- understandability
- clarity
- unambiguity
- conciseness
- retrieveability
- traceability
- modifieability
- concistency
- completeness
- reusability
- credibility

4.2.1 Git documentation

Since the git documentation contains setup guides, the important aspects for this component are understandability, modifieability, and reusability. Understandability is one of the important aspects since the guides are written for a wide variety of people, either with or without technical knowledge. During the writing of the guides, each step that had to be taken to set up a component, had to be documented. With this execution, the aim was to improve understandability, both for people who have and don't have experience with the component.

Reusability was another important component. Because of the scientific background of the project, the aim was to set up the project in a way that other people could easily set up the components that we made. This was done by using Docker in some cases, but in cases where this wasn't applicable, we had to write longer guides since docker only required a few commands.

Modifieability is deemed important since not everyone sets up their project the same as someone else. This is why all of the guides in git explain how a command is used, and what the parameters mean. This way people should be able to reuse the components we make with their own modifications.

4.2.2 Technical report

In big lines, the technical report follows the same quality aspects of the git documentation. This is because it has the same purpose; explain the components. This is why understandability is again an important aspect. A well set up document is always nicer to read than a document that is all over the place. Thats why the technical report uses a general layout, is divided in chapters per component and uses images and code snippets to explain things instead of

plain text.

Both Reusability and Modifieability are important because of the same reason as the git documentation since the technical reports feature the same guides used in the git repositories. This is done so that the end user doesnt have to switch from documents that often and has everything in one centralized place, This also contributes to the overall understandability of the project.
Scientific Programming papers Since the Scientific programming papers are written individually by each team member, no conclusion can be made about the quality aspects used in making the documents. We can however set up guidelines as to what quality aspects are usefull.

The following statements are based on assumptions. Since these papers are trying to convey information from a source to the reader, understandability is again an important aspect. Since this information comes from a source, the information has to be traceable and known to be trustworthy information. The factor of completeness also matters in a research, if the writer does not supply enough evidence when making a statement or when making conclusions, the reader could get lost into the facts.

4.3 Wrap up

When writing documentation, put yourself in the readers perspective. This enables you to understand the perspective of a reader. Also check if you understand what you have written yourself; if you don't get it, neither will someone else.

4.4 Sources

[1] W. Ding, P. Liang, A. Tang, H. van Vliet (august, 2013) "Knowledge-based approaches in software documentation: A systematic literature review" [PDF] accessed on june 21 2019

5 Attachments

5.1 Ina Docu - Team semester 1, 2018/2019

Power Measurement
Energy consumption acquisition and processing

B. VAN WALSTIJN, R. STAUTTENER, R. BOLDING
Amsterdam University of Applied Sciences
March 5, 2019

-Overview

The power consumption of a whole Raspberry Pi cluster was measured, processed and sent to a visualisation interface. Every individual Raspberry Pi has an external hardware interface which measures physical quantities and communicates to the Raspberry Pi. The acquisited data is processed and forwarded to a monitoring agent. The monitoring agent collects all data, processes these and hosts a server. All data from this server is requested for by visualisation front-end software for the end user to monitor. In this paper we will discuss the implementations used and thought for.

-Hardware interface

The hardware interface has been made for the Raspberry Pi. The function of the hardware interface is to measure the voltage and current consumed by the Raspberry Pi. The hardware interface is made so that it can be placed on top of the Raspberry Pi, connecting via the GPIO pins. For designing the hardware interface we have used EAGLE, an application intended for designing electronic schematics and printed circuit boards (PCB's). The hardware consists of the following components:

- Power connector
- Current measurement circuitry
- Filtering

- GPIO connection pins

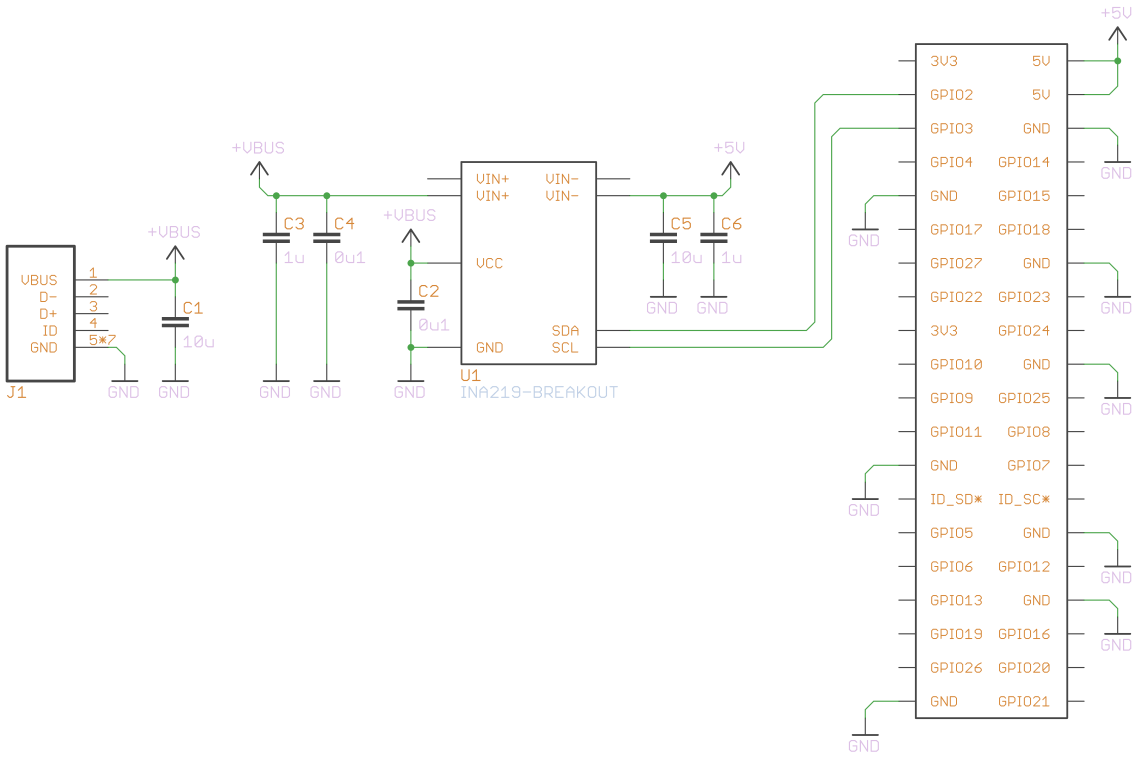


Figure 1: Hardware schematics

Let's start off with the power connector. We chose a micro USB connector because this resembles the Pi's. The placement of the connector is as close to the original USB connector so this gives the most compatibility. As current measurement IC we chose the INA219 chip by Texas Instruments. This chip has a very high measurement accuracy (max. 0.5%) and quick sampling time (0.5ms). It also features an I2C interface which is very suitable for the Raspberry Pi, since it has no internal ADCs. We have also implemented some simple filtering with the capacitors on the input and output of the current sensoring IC. This makes for a smooth measurement without too much ripple in the output data. And last but not least there are 2x20 pins GPIO pin headers for the connection with the Raspberry Pi.

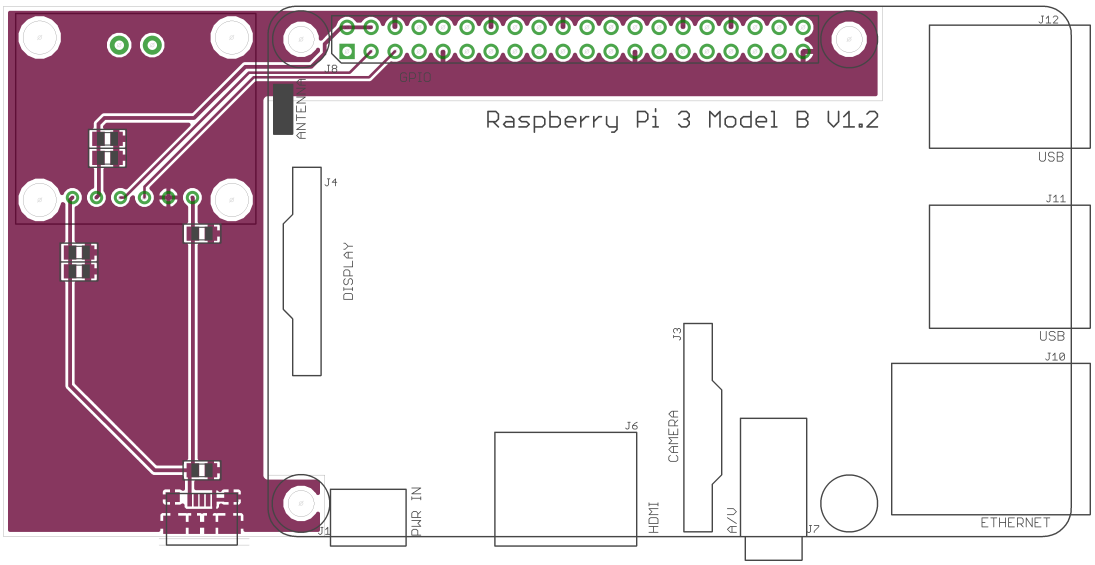


Figure 2: Hardware board

-Software

The software for getting the required data consists of the following components:

- I2C library
- Prometheus Client
- Psutil
- Filtering

The software uses the I2C library from Adafruit. This means that no further code is neccesary for using this communication channel. Having imported the 'INA219' library, we are able to create a variable using the INA219 constructor. This way, we have an instance which can be used to communicate directly.
Aside from the power measurement data, we have also implemented some CPU an RAM diagnostics. By importing Psutil we can achieve these and send them through.
Once the data is acquired, the software can start to do some processing. We know that the measurement interval is not infinitesmal, so there'll have to be some sort of smoothing and calculating. If not so, the sampling speed

could turn out lower than changes in the physical quantity, which leads to the well-known Nyquist theorem that is not met. So, we implemented a filter.
We have created three different types of filtering. One saves the gathered data in an array, moves them one to the left, appends the new value at the right and calculates the average (running average method). Optionally, the new measured value can be replaced by the calculated average value, which gives a FIR-filter implementation method, leading to a faster converging algorithm by reducing overshoot. The third algorithm is a simpler implementation of a running average and calculates the weight of the previous average value, adds this up to the new value and calculates the average. All these methods run in an infinite loop.
Currently, the third method is used. This method, however, could be further examined and implemented with inclusion of the time interval constant to determine the weight value. This way an even more exact output can be generated which is more reliable.
As soon as the data is processed, it can be sent to the next part. This is where Prometheus comes in. Because Prometheus does not run the software itself, and so does not re-run the software multiple times, the connection with the Prometheus Client can be run within the infinite loop. And so it does. Every 'interval' time, all data is sent to the Prometheus Client via its function. The HTTP-server that has been set up is now used as a server to keep a bunch of data present and presented. The data will be converted and then sent to the server. The server is running on port 8000 and can be accessed by the Prometheus Server (discussed in chapter Prometheus).

The code is as follows:

metric_INA219.py:

```
#!/usr/bin/env python

from __future__ import division
from prometheus_client import start_http_server, Summary, Gauge

from ina219 import INA219, DeviceRangeError
from subprocess import PIPE, Popen
import psutil
import time
```

```
class Filtervalue:
  def __init__(self, initialvalue):
    self.averagevalue = initialvalue

  # def __init__(self, initialvalue, sumcount):
    # self.sumcount = sumcount
    # self.lastvalues = []

    # for x in range(0, self.sumcount+1):  #0-100
      # self.lastvalues[x] = self.averagevalue


def updatevalue(self, newvalue, avgweight):  # three different
types of filter
  # sumofvalues = 0

  # self.lastvalues[self.sumcount] = self.newvalue #100

  # for x in range(0, self.sumcount):  #0-99
    # self.lastvalues[x] = self.lastvalues[x+1]
    # sumofvalues += self.lastvalues[x]

  # self.averagevalue = sumofvalues / self.sumcount  # filter
  no. 2

  # self.lastvalues[sumcount] = ((sumofvalues * (1-avgweight)) +
   (self.newvalue * (avgweight))) / sumcount  #extra filtering

  self.averagevalue = (self.averagevalue * (1-avgweight)) + (
  newvalue * avgweight)


collect_time = 1


inamodule = INA219(0.1, 2.0)
inamoduleg.configure(inamodule.RANGE_16V, inamodule.
GAIN_8_320MV, inamodule.ADC_12BIT, inamodule.ADC_12BIT)


REQUEST_TIME = Summary('request_processing_seconds',
```

```
'Time spent processing request')

prometheus_current = Gauge('energy_current', 'Filtered current
of the system')
prometheus_voltage = Gauge('energy_voltage', 'Filtered voltage
of the system')
prometheus_power = Gauge('energy_power', 'Filtered power of the
 system')

prometheus_cpu_percent = Gauge('cpu_percent', 'Filtered CPU
percentage')
prometheus_memory_mb = Gauge('memory_mb', 'Filtered RAM
consumption')


#initialize variables with Filtervalue constructor
filtered_current = Filtervalue(inamodule.current())
filtered_voltage = Filtervalue(inamodule.voltage() * 1000)
filtered_power = 0.0

filtered_cpu_percent = Filtervalue(psutil.cpu_percent())
filtered_memory_mb = Filtervalue(0.0)


if __name__ == '__main__':
  # Start server for Prometheus to scrape from
  start_http_server(8000)

while True:

  # Filter every /interval/ time
  start_time = time.time()
  while time.time() < start_time + collect_time:
    filtered_current.updatevalue(inamodule.current() , 0.1)
    filtered_voltage.updatevalue(inamodule.voltage() * 1000,
    0.1)

  filtered_cpu_percent.updatevalue(psutil.cpu_percent(), 0.1)
  ram = psutil.virtual_memory()
  filtered_memory_mb.updatevalue(ram.used / 1048576, 0.1)  # b
  to mb
```

```
filtered_power = filtered_current.averagevalue *
filtered_voltage.averagevalue / 1000

# post metrics
prometheus_current.set(filtered_current.averagevalue)
prometheus_voltage.set(filtered_voltage.averagevalue)
prometheus_power.set(filtered_power)

prometheus_cpu_percent.set(filtered_cpu_percent.averagevalue)
prometheus_memory_mb.set(filtered_memory_mb.averagevalue)
```

-Points of improvement

So our software works. However, there are still a few things that could be worked on. I will note them here:

- Power integration in t-domain to achieve energy
  What our Power monitoring does not do is calculation of the total energy consumption. As you might well know, energy is defined by the power multiplied by time. However, for changing values, energy is always calculated by integrating power over time. How can this be achieved? There are multiple answers to that question. One is to implement this is hardware. Design a PCB that filters the measured current and voltage, multiplies them and integrates them. This is difficult to do the right way, as a lot of designing skills come into view. One of the other options however, is an implementation in software. In this case it's necessary to have very fast data, very fast processing and very precise timing. I will explain more about timing in the next bullet point.

- Timestamps (snapshots) for gathered data.
  As I said in the previous bullet point, timing is very important. It's mandatory to have a very accurate measurement at a very specific, precise time. This is also called snapshots. When it is possible to create snapshots, a software implemented version of time integration can be done and precies results can be expected.

- Current measurement accuracy (not precision). It's very important to have a good current measurement. As we have seen with the Sparkfun breakout boards, probably because of the lack of designing skills of

the developers' side, these boards gave fluctuating outputs compared to one another. Their precision is high, but accuracy is low. In order to achieve representative measurement data, you will want to make a very solid current measurement. This however, is again a hardware thing. Consider predesigned alternatives with unpredictable outputs, or consult people from electronic design studies who know what they're talking about.

- Python software.
  So you'll notice that the Python software is not the fastest in the world. Communication with low level electronics is done by some GPIO libraries and their data is processed in Python. Because timing is so important, the software tries to do an infinite amount of measurements and calculations just to get the data right (the exact speed depends on the sampling speed in the INA219). The software now uses up a huge amount of CPU time just to do the filter calculations. It would be wise to make these less CPU-dependent or even better, as light as possible.