Brightness Monitor Doc Documentation

Release 1.0.0.

s0556350 Samuel Erb & s0556166 Steffen Exler

05.07.2017

Contents:

KAPITEL 1

Einleitung

1.1 Einleitung

1.1.1 Idee

Es sollen die exakten Sonnenstunden an mehreren Standorten erfasst werden. Dazu wird die Helligkeit mit dem Photowiderstand gemessen und von einem Raspberry Pi verarbeitet. Als Ausgabe dessen wird sein: Sonne, Wolken mit verminderter Wolkendichte, Wolken mit verstärkter Wolkendichte, Dunkel. Gleichzeitig wird die Dauer dieser Zustände gemessen und in einer Datenbank gespeichert.

1.1.2 Umgesetzt wurde

Ein Programm im Server - Client Verbund, wobei der Client die Licht Intensität misst und diese in 5 Minuten Takt zum Server weiter sendet. Die Klienten messen nur innerhalb der Zeit in der die Sonne scheint, dafür übergibt der Server dem Client den genauen Standort womit der Client ausrechnen kann, ob die Sonne scheint oder nicht.

Der Server ist eine Website welche die gesendeten Daten zusammen fasst und sie Grafisch in einer Karte & einem Chart ausgibt.

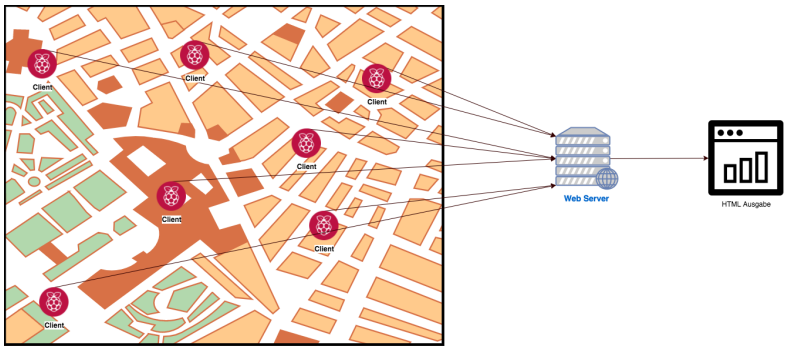


Abb. 1.1: Software Design

Raspberry Pi Client

Der Client wurde in Python 2.7 für den Raspberry Pi geschrieben.

- Quellcode : https://github.com/BrightnessMonitor/BrightnessMonitorClient

Abb. 1.2: Raspberry Pi Brightness Monitor Client mit WLAN

Website

Der Server wurde in das Python Web Framework „Django“ geschrieben und auf einer gratis Instanz von heroku.com gehostet.

- Quellcode : https://github.com/BrightnessMonitor/BrightnessMonitorWeb
- Website URL: https://infinite-crag-79176.herokuapp.com/

Abb. 1.3: Bild Quelle: https://en.wikipedia.org/wiki/Heroku

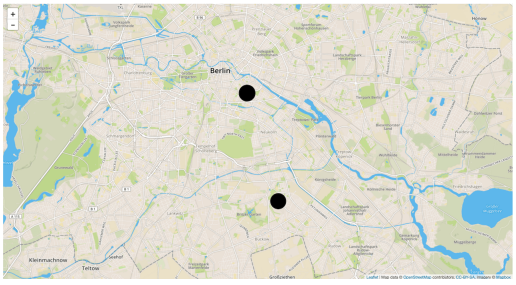


Abb. 1.4: Screenshot von https://infinite-crag-79176.herokuapp.com/

Dokumentation

Die Dokumentation wurde mit Sphinx *[5]* erstellt und gehostet auf readthedocs.io und github.com.

- Quellcode : https://github.com/BrightnessMonitor/BrightnessMonitor.github.io
- Online Dokumentation: https://brightnessmonitor.github.io/

- Read the docs: http://brightness-monitor-docs.readthedocs.io/de/latest/

Präsentation

Die Präsentation wurde in PowerPoint erstellt und ist auf github.com gehostst.

- Präsentation : Download Link

KAPITEL 2

Client

2.1 Client

2.1.1 Software

Grundkonzept

Die Hauptaufgabe der Client-Software ist es, die Messwerte, wie in *Erläuterung* beschrieben, zu sammeln und anschließend an den Server zu verschicken.

Helligkeitsmessung

Die Helligkeit wird gemessen und anschließend in einer lokalen Datenbank hinterlegt. Dieser Vorgang erfolgt alle 5 Sekunden. Da der Server den Durchschnitt eines Tages berechnet, würden Messdaten, die vor dem Sonnenaufgang bzw. nach dem Sonnenuntergang erhoben wurden, das Ergebnis verfälschen. Um diese Problematik zu lösen werden Messwerte verwendet, die vom Sonnenaufgang bis zum Sonnenuntergang erhoben wurden. Um diesen Zeitraum zu ermitteln haben wir eine Formel *[2]* verwendet, die anhand des Höhen- und Breitengrades, den Sonnenaufgang und den Sonnenuntergang eines Tages berechnen kann. Diese Berechnung erfolgt alle 5 Minuten in einem eigenen Thread um die CPU-Auslastung zu senken.

Versenden der Daten

Die Datenbank wird im 5 Minuten Intervall ausgelesen und die Daten werden an den Server übermittelt. Anschließend werden alle Einträge der Datenbank gelöscht. Dieser Vorgang erfolgt jedoch nur dann, wenn auch eine aktive Verbindung mit dem von uns verwendeten Server über das Internet vorhanden ist. Sollte dies nicht der Fall sein, überspringt die Software den Vorgang und wartet 5 Minuten bis zum nächsten Versuch.

2.1.2 Probleme

Verwendete Hardware

Mit einem Kondensator der mehr Kapazität besitzt, würden die Messergebnisse bei starker Helligkeit (in dem von uns beobachteten Zeitraum von 11:00 bis 14:00 an sonnigen Tagen) genauer werden, da eine längere Zeit von Nöten wäre um ihn aufzuladen. Über den oben beschriebenen Zeitraum hatten wir Messwerte von „0“ was „sehr hell“ entspricht. Auf Grund dessen sind unsere Statistiken teils nicht aussagekräftig.

Auch orientieren sich unsere Messwerte an keiner physikalischen Größe, sodass man diese nicht für andere Zwecke weiterverwenden könnte. Es gibt jedoch Module, die genauere Messwerte liefern bzw. die Helligkeit in Lux ausgeben können.

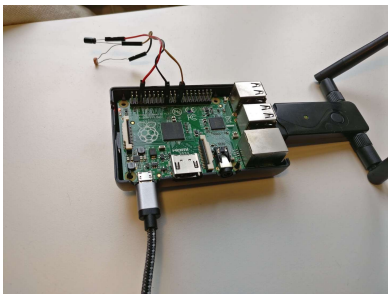


Abb. 2.1: Raspberry Pi mit Sensor

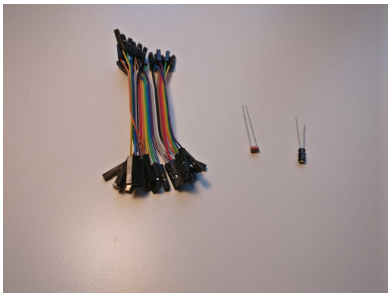


Abb. 2.2: Raspberry Pi mit Sensor

Helligkeitsverlauf über den Tag

Das nächste Problem stellt der Helligkeitsverlauf über einen Tag dar: umso rechtwinkliger die Sonne zum Standort, desto intensiver wird das Licht. Daraus würden fehlerhafte Messdaten hervorgehen, denn z.B. eine Wolke die mittags die Sonne verdeckt könnte äquivalent zu Sonnenschein nach dem Sonnenaufgang sein. Es muss also eine Justierung der Messdaten stattfinden. Wir haben eine Formel *[3]* gefunden, die die absolute Helligkeit unter Angabe des Standpunktes, des Datums sowie der Zeit, ausgibt. Die Maßeinheit ist Solarimeter (W/m2), welche man zu Lux umrechnen kann. Die Justierung der Messdaten wurde mangels Zeit nicht implementiert. Unter Verwendung genauerer Messgeräte und mit Hilfe dieser Formel ließe sich die Problematik lösen.

2.1.3 Sensoabfrage

Aufbau

Um die derzeitige Helligkeit zu erfassen wurde in unserem Versuchsaufbau ein Fotowiderstand GL5528 LDR, ein Kondensator mit der Kapazität von einem Mikrofarad und 50 Volt Maximalspannung benutzt. Die negative Seite des Kondensators ist mit der Masse des Raspberry Pi's (im folgendem schwarzes Kabel) verbunden, die positive Seite ist mit einer Seite des Fotowiderstands verbunden sowie an einen „GPIO"-Pin des Raspberry Pi's (im folgendem blaues Kabel). Die andere Seite des Fotowiderstands ist mit einem „GPIO'-Pin verbunden an dem eine 3,3 Volt Spannung anliegt (im folgendem rotes Kabel). *[4]*

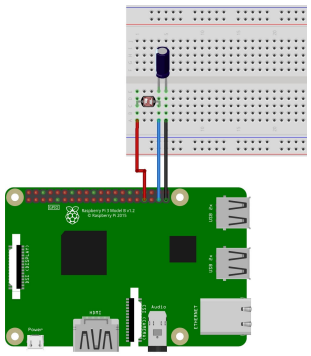


Abb. 2.3: Raspberry Pi Versuchsaufbau mir Steckplatine und LDR Fototwiderstand *[4]*

Erläuterung

Der Widerstand des Fotowiderstandes ist lichtabhängig und verändert sich bei unterschiedlicher Helligkeit. In einem hellen Zustand ist der Widerstand größer (bei 10 Lux 8 ~ 20 K). In einem dunklen Zustand ist der Widerstand sehr gering (bei 0 Lux 1 M Minimum).

Ein Kondensator im Gleichstromkreis lädt sich auf, bis seine Kapazität erreicht ist. Ist der Kondensator geladen fließt kein Strom.

Funktionsweise

Es läuft nun ein Strom durch den Fotowiderstand in den Kondensator, dieser lädt sich ungefähr bis zu ¾ seiner Kapazität auf. Während dieses Vorgangs liegt kein Strom an blau an. In diesem Zeitraum wird ein Wert hochgezählt der später als Ausgabe der Funktion gilt. Ist der Kondensator nun zu ¾ geladen, wird der Widerstand größer und der Strom beginnt über das blaue Kabel zu fließen. Dies bemerkt der Raspberry Pi sofort und bricht den Vorgang ab. Der gezählte Wert wird in unserem Fall in eine Datenbank zusammen mit der aktuellen Zeit abgelegt und wird später in den nächsten Ablaufschritt übergeben.

Quellcode 2.1: Code-Ausschnitt Messwerterfassung *[4]*

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

timing = 0
GPIOPIN = 11
GPIO.setmode(GPIO.BCM)

def RCtime (RCpin):
```

```
    reading = 0
    GPIO.setup(RCpin, GPIO.OUT)
    GPIO.output(RCpin, GPIO.LOW)
    time.sleep(0.1)

    GPIO.setup(RCpin, GPIO.IN)
    while (GPIO.input(RCpin) == GPIO.LOW):
        reading += 1
    return reading
```

KAPITEL 3

Website

3.1 Website

3.1.1 Aufgabe des Servers

Das Ziel des Server war es eine Zentrale Instanz zu erstellen, wo alle Daten gesammelt und Zentral ausgewertet werden können. Auch war es wichtig möglichst kosten effektiv zu arbeiten und Resourcen schonend zu designen.

3.1.2 Verwendete Software & Hoster

Als Software wurde das Python Framework Django eingesetzt mit dem Django REST Framework *[6]* um eine Api Schnittstelle zu erzeugen. Die auswahl des Hoster fiel auf heroku.com, da Heroku für Entwickler eine Gratis Instanz für Python Websites anbietet welche sich nach 30 Minuten inaktivität automatisch herunter fährt und eine PostgreSQL Datenbank mit 10.000 Zeilen enthält. Der Quellcode ist auf github.com gehostet und mit einem Webhook *[9]* verbunden, dadurch wird mit jedem Git push auf dem Master Branch der aktuelle Quellcode auf der Heroku Instanz gesendet.

3.1.3 Bestandteile der Server Software

Models - Datenbank

Die Datenbank besteht aus 3 großen Komponenten, dem User, Device (Raspberry Pi) und den täglichen Werten.

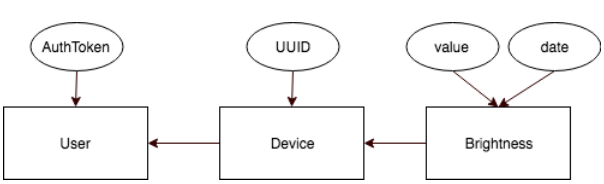


Abb. 3.1: Datenbank Model

API - Schnittstelle

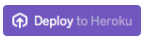
Die Schnittstelle gibt bei einer leeren anfrage Value anfrage die latitude und longitude Koordinaten des gesicherten Device aus und wenn ein Wert mit gegeben wird, wird in der Datenbank überprüft ob der User mit dem gesendeten Auth Token das Device mit der gesendeten UUID gehört, wenn ja wird der gesendete Wert mit dem gesendeten Datum in der Datenbank gesichert.

View - Ausgabe

Die Ausgabe umfasst 2 Seiten, die Landingpage gibt auf einer Open Street Map *[1]* grafisch wieder, wo sich die Geräte befinden, die Karte wird automatisch zentriert damit alle Geräte sichtbar sind. Auf der zweiten Seite unter `/stats` befinden sich Bar Charts *[10]* welche alle Geräte den täglichen durchschnitts Wert ausgibt.

Deployment

Um ein möglichst einfaches und stabiles deployment zu ermöglichen, wurde das Heroku Django Template *[7]* als Grundgerüst verwendet und mit hilfe der Dokumentation von Heroku *[8]*. In Kombination mit einem Github Webhook *[9]* ist das Deployment ausgeführt, sobald der Master Branch ein git push empfängt.

Außerdem wurde in der Repo Readme ein 

[11] Button eingesetzt. Um auch dritten die Möglichkeit zu geben, schnell und leicht diese Software auf ihrer Privaten Heroku Instanz einsetzen zu können.

3.1.4 Probleme

Zu kleine Datenbank

Da dei Gratis Instanz von Heroku nur eine 10.000 Zeilen PostgreSQL Datenbank erlaubt war die Datenbank innerhalb der ersten Tagen zu 100% voll und das Design musste Resourcen sparender geändert werden. Dadurch wurden mit jedem Datenupload die Werte zusammengefasst zu einen Tages durchschnitts Wert

Schnittstelle Absichern vor dritten

Das Ziel war das nur die Personen Daten hochladen dürfen, die den richtigen Auth Token + Device UUID senden. Dadurch aber das der Auth Token vom Django REST Framework integriert worden war, war es nur mit hilfe vom ausführlichen lesen der Dokumentation *[6]* möglich heraus zu finden, welcher User sich hinter welchen Request sich verbirgt.

KAPITEL 4

Projekt Auswertung

4.1 Projekt Auswertung

4.1.1 Daten

Die gesammelten Daten reichten nicht für die gezielte Auswertung aus. Da uns die Zeit fehlte um die Probleme von dem Webserver Datenbank Problem zu lösen (siehe: *Zu kleine Datenbank*).

Es ist nur der Tägliche Druchschnittswert eines jedem Gerätes zu erkennen, es ist aber nicht ab zu lesen wie lange Licht auf dem Gerät schien oder wie Intensiv, da auch die verwendete Hardware zu ungenau ist (siehe: *Raspberry Pi mit Sensor*)

4.1.2 Was mit mehr Zeit umgesetzt wäre

Server Umzug

Da die Gratis Instanz von heroku.com eine zu kleine Datenbank enthielt ist eine Bezahlpflichtige Heroku Postgres unausweichlich oder der Umzug auf dem HTW Berlin Studie Server bzw. auf das Kostenloses Kontingent für AWS doch beide Möglichkeiten würden einen erhöhten Zeitaufwand erzeugen.

Datenverarbeitung auf Server ausweiten

Die empfangenden Daten von den Klienten stüdlich automatisch auswerten und dadurch die Datenlast minimieren.

Ausführlichere Grafische Auswertung

Auf einer Open Street Map mit dem Leaflet Bibliothek *[1]* grafisch die Wolkendichte anzeigen und die Bar Charts eine Stündliche Anzeige integrieren mit der warscheinlichen Wolkendichte.

Client Log funtion integrieren

Fehler und Status log inteiren um schneller Fehler in Hard & Software aufspüren zu können.

Anpassung der Hardware

Mit einem Kondensator der mehr Kapazität besitzt, würden die Messergebnisse bei starker Helligkeit (in dem von uns beobachteten Zeitraum von 11:00 bis 14:00 an sonnigen Tagen) genauer werden, da eine längere Zeit von Nöten wäre um ihn aufzuladen.

Über den oben beschriebenen Zeitraum hatten wir Messwerte von „0“ was „sehr hell“ entspricht. Auf Grund dessen sind unsere Statistiken teils nicht aussagekräftig.

Auch orientieren sich unsere Messwerte an keiner physikalischen Größe, sodass man diese nicht für andere Zwecke weiterverwenden könnte.

4.2 Literaturverzeichnis

KAPITEL 5

Indices and tables

- genindex
- search

Literaturverzeichnis

[1] *Leaflet Dokumentation*. URL: http://leafletjs.com/reference-1.1.0.html.

[2] Geplaatst door Michel. Calulating sunrise and sunset in python. URL: http://michelanders.blogspot.com/2010/12/calulating-sunrise-and-sunset-in-python.html.

[3] *Forum: Suche Formel für Globalstrahlung*. URL: http://www.wetterstationen.info/forum/wetterstationsforum/suche-formel-fur-globalstrahlung/.

[4] Stefan Kraus. Raspberry pi - helligkeit mit einem LDR fotowiderstand erkennen. Online, 01 2016. URL: http://krausix.de/raspberry-pi-ldr-fotowiderstand/.

[5] *Spinx Dokumentation*. URL: http://www.sphinx-doc.org/en/stable/.

[6] *rest-framework-tutorial: Source code for the REST framework tutorial*. original-date: 2012-10-26T11:52:44Z. URL: https://github.com/encode/rest-framework-tutorial.

[7] Heroku-django-template: a django 1.10 base template featuring all recommended best practices for deployment on heroku and local development. original-date: 2014-10-17T14:28:13Z. URL: https://github.com/heroku/heroku-django-template.

[8] Deploying python and django apps on heroku \textbar heroku dev center. URL: https://devcenter.heroku.com/articles/deploying-python.

[9] GitHub integration (heroku GitHub deploys) \textbar heroku dev center. URL: https://devcenter.heroku.com/articles/github-integration.

[10] *Bar Charts*. URL: https://developers.google.com/chart/interactive/docs/gallery/barchart.

[11] Creating a ‚deploy to heroku‘ button \textbar heroku dev center. URL: https://devcenter.heroku.com/articles/heroku-button.

Stichwortverzeichnis