Smart Growbox

Jonas Valentin Rose

2020

Inhaltsverzeichnis

1 Einleitung

Dies ist der schriftliche Teil meiner Maturarbeit, bei welcher ich mich mit der Konstruktion einer möglichst smarten Growbox auseinandersetze. Ich stelle hier so viel möglich und so wenig wie nötig alle meine Gedanken vor, welche mich während dieser Zeit motivierten oder plagten. Angefangen bei der Idee selbst, über die pflanzlichen Anforderungen, weiter zu den drei Kernelementen des Konstrukts bis hin zum esten Webserver, ich verhoffe mir, Ihnen einen möglichst nahen Einblick in meine letzten paar Monate zu geben.

1.1 Idee

Mögen Sie Tomaten? Ich tue es, im Sommer 2019 habe ich meine eigenen gezüchtet, direkt auf meinem Balkon. Sie stellten sich als recht gut gereift heraus. Allerdings plagte es mich, während des vierwöchigen Korsika-Urlaubs die Verantwortung bezüglich der Bewässerung abzugeben. Nicht dass ich gerne selbst Pflanzen bewässere, klar es hat durchaus etwas magisches an sich, doch nein, ich in faul. Magisch faul. Faul lässt es sich am besten clever leben, ergo: Es muss eine richtige Lösung her. Ich will in den Urlaub fahren können, ohne die Verantwortung meiner Stecklinge abgeben zu müssen, aber ich will mir auch nicht die ganze Zeit Sorgen machen müssen. Ich will sozusagen den *Föifer* und *s'Weggli*. Zur Motivation komme ich erst im nächsten Kapitel, doch lassen Sie mich Ihnen eins versichern: Das Internet kann sehr inspirierend wirken. Lange Rede, kurzer Sinn - Ich will Tomaten essen, ohne mich um Sie kümmern zu müssen. Sie sollen wachsen, aber ohne meine Bemühungen. Klar doch, eine Maschine muss her! Eine solche, mit der man dem Tomätchen die perfekten Wachstumsbedingunen geben kann, ohne sich auch nur darum zu bemühen.
Das mit der Bemühung ist so ein Thema, denn zuerst will die Maschine gebaut werden. Ich stelle sie mir so vor: Eine Growbox in meinem Zimmer mit elektronischer Beleuchtung, automatischer Bewässerung und einem sich selbstregulierenden Belüftungssystem. Klingt gewagt, aber nicht unmachbar. Ich beweise es Ihnen.

2 Ziel und Abgrenzung

2.1 Das Konstrukt

Im praktischen Teil konstruiere ich eine Maschine, welche mir Esspflanzen züchtet. Sie soll die drei Hauptaufgaben übernehmen, welche ich als Belüftung, Beleuchtung und Bewässerung identifizere. Sie soll selbst gebaut sein, d.h. keine fremde Regelungstechnik wie Zeitstromschalter oder Thermostaten. Die finanzierung erfolgt selbst. Ich werde die Bauteile selbst kaufen/zusammenbauen/konstruieren/programmieren und das dazugehörige Wissen beziehe ich grösstenteils aus dem Internet. Ich mache auch von der Erfahrung meines Vaters gebrauch, er steht mir in besonders abstrakten Problemstellungen bei.

2.2 Schriftlicher Teil

Hier im schriftlichen Teil versuche ich, wie in der Einleitung bereits erwähnt, dem Leser einen möglichst direkten Einblick in die Lösungsfindung der Teilprojekte zu bieten. Was ich nicht vorhabe, ist es, Ihnen Python, C++ oder Löten beizubringen. Sie werden hin und wieder auf Codesnippets treffen, bei denen es um die Funktion des jeweiligen Ausschnitts geht. Aus all diesen Gründen ist es von Vorteil, wenn Sie schon etwas mit Technik zu tun hatten. Nebst dem

Arbeitsprozess schildere ich im nächsten Kapitel auch meine Motivation, dieses Projekt nicht *nur* als Maturarbeit zu verfolgen. Das Kapitel *Arbeitsprozess* ist als Hauptteil des Kommentars gedacht. Anschliessend werde Ich mit möglichst wenigen aber klaren Sätzen das *Konstrukt* selbst noch kurz erklären, wie es funktioniert, wo die Tücken liegen und worauf ich besonders Stolz bin. Im Kapitel *Perspektiven* gehe ich auf alles ein, was mich rund um die Arbeit beschäftigte. Zu guter Letzt ziehe ich ein Fazit und bedanke mich bei allen Menschen, welche mir geholfen haben, mein Vorhaben in die Realität umzusetzen.

3 Motivation

Ich verbringe (zu) viel Zeit auf YouTube. Eine mich inspirierende Gattung sind die Art Videos, welche sich mit Offgrid-Versorgungssystemen beschäftigen, also der Nahrungs-/Energieversorgung eines Lebensraums ohne Teil des Hauptnahrungs-/energienetzes zu sein. Dieses Autarksein fasziniert mich. Des weiteren stiess ich vor etwa 10 Monaten auf den Kanal von *Jeb Gardener*. Ich fasse die Gemeinsamkeiten seiner Videos im folgenden Satz zusammen: Verrückter Gärtner züchtet mit komplett unkonventionellen und (laut einigen Stimmen seitens des *traditionellen Gärtnermilieus*) ethisch kritischen Methoden Pflanzen. Seine Bewässerungsmethoden sind überaus selten traditioneller Art, bedeutet, dass er fast immer auf Erde verzichtet. Er stellt die Pflanzen direkt ins Wasser. Beleuchtet wird künstlich, mit pinken LEDs und die Nährstoffe sind synthetisiert und werden einfach ins Wasser geworfen, wo sie sich auflösen. Konventionalität interessiert ihn nicht. Dazu kommt noch, und das ist fast das Beste, dass seine Videos durch die musikalische Begleitung von epischer Orchestermusik eine fast zerreissende Spannung erreichen. Stellen Sie sich einen durchgeknallten Gärtner vor, der von einem Streichorchester begleitet, Wurzeln filmt. Seine Unverdrossenheit im Umgang mit Anbaumethoden ist sehr ansteckend. *Whatever gets the job done.*
Segelvideos sind seit ein paar Jahren ebenfalls sehr *in*, sie verdanken ihren Erfolg wohl den Lebensumständen, in denen alle Landratten leben, welche gerne zur See fahren würden: Der landrattige YouTube-Zuschauer sieht durch seinen Bildschirm den unendlichen Horizont, den Sonnenuntergang und versucht sich den Wind in den Haaren vorzustellen, der Segler lebt diesen Moment und fängt es mit der Kamera ein. Es ist ein Zusammenspiel und ich bin ein Teil davon. Die Freiheit welche man durch das Segeln erlangt wirkt auf mich ansteckend. Was aber hat das mit meiner Maturarbeit zu tun? Hier der Link: Bei einer Ozeanüberquerung hat man nach ein paar Wochen keine frischen Lebensmittel mehr, die Rede ist von Früchten, frischen Kräutern und Gemüse. Was aber, wenn man sich seine Lieblingstomaten einfach selbst auf dem Boot züchtet? In einer Miniaturversion meines Konstrukts, mit Solarpanels als Energieversorgung. Wäre das nicht einfach genial? Wenn ich das als Startup-Projekt gut verkaufen kann, lässt es sich markttechnisch sogar verkaufen. Ich könnte DIY-Anleitungen machen, sodass mehr Segler mehr frische Sachen essen können. Durch *Der verschenkte Sieg* von *Bernard Moitessier* habe ich gelernt, wie sehr frisches Gemüse oder Früchte die Lebensqualität auf einer Pazifiküberquerung heben können. Die oben genannten Erlebnisse motivieren mich, das *Konstrukt* nicht nur als Maturprojekt sondern als Lebensprojekt zu sehen. Wer weiss, vielleicht studiere ich etwas in diese Richtung.

4 An die Arbeit

4.1 Wachstumsbedingungen

Das Hauptziel in dieser Arbeit ist die Lebenserhaltung von Pflanzen durch eine künstliche Umgebung. Pflanzen bergen gewisse Ansprüche und um die geht es jetzt kurz. Ich erkläre den Anspruch und weswegen er besteht - mit einer für die Arbeit tragbaren Tiefe.

4.1.1 Wasser

Pflanzen brauchen Wasser für den Transport von Nährstoffen, für die Photosynthese und um den Zelldruck und die damit verbundene Stabilität zu erhalten - unbewässerte Pflanzen lassen bekanntlich ihre Köpfchen hängen.

4.1.2 Licht

Licht ist die Energieform, welche sich die Pflanze bei der Photosynthese zunutze macht, um energiearme Baustoffe in Energiereiche zu verwandeln. Diese werden für den Bau und die Energiespeicherung verwendet.

4.1.3 Luft

Luft, auch wenn eigentlich überall vorhanden, muss einen genügend grossen CO_2-Anteil bergen, um der Pflanze die Photosynthese zu ermöglichen. Sie wird wie das Licht grösstenteils über die Blätter aufgenommen.

4.1.4 Nährstoffe

Nebst der selbst hergestellten Glucose verwendet die Pflanze auch noch andere Bausteine um sich Struktur zu geben und den Stoffwechsel zu betreiben. Diese können nicht aus der Luft gefischt werden, sondern werden in gelöster Form von den Wurzeln aufgenommen. Dazu zählen Elemente wie Phosphor, Kalium und Stickstoff.

4.2 Wie weiter?

Ich kenne nun die Ansprüche meiner Pflanzen, aber wie baue ich eine Maschine, die denen Genüge tut? Dies ist der Kern der Arbeit, alles dreht sich darum. In diesem Kapitel stelle ich vor, wie ich mir die Konstruktion vorgestellt habe, während ich im Internet recherchiert habe. Zuerst brauche ich Bauteile, doch von wo nu- Ach ja genau, das Internet, es ist voller Möglichkeit, Wissen und unnötigem Zeugs, wussten Sie, dass es eine Bewässerungsmöglichkeit gibt, welche komplett auf Substrate wie Erde oder Tonpellets verzichtet? Nennt sich Hydroponik und ist recht simpel: Eimer nehmen, Wasser rein, Nährstoffe rein, Pflanze rein. Alles ganz fein. Doch es geht besser. Mehr dazu im nächsten Kapitel ;)
Materialsuche ist ein anstrengendes Unterfangen, ich bin aber nochmal glimpflich davongekommen, denn ich fand eine gute Menge Material in unserer Kellerwerkstatt. Mein Vater ist ein *Techie*, der gerne Elektroteile sammelt, zu meinem Glück.

4.3 Hornbach, Cannabis-Shops und mein Keller

Hier sind die essenziellen Bauteile des *Konstrukts*.

Grow Box Die Hülle stellt meine Grow Box dar, sie besteht aus einem Plastikgewebe, das an der Innenseite mit Mylar beschichtet ist - es reflektiert zu einem hohen Bestandteil Licht. Sie gibt Form und ermöglicht es den Pflanzen, ein eigenes kleines Klima zu errichten, eine eigene, kleine Ökosphäre sozusagen.

LEDs Ich beleuchte meine Pflanzen künstlich mit zwei 50 Watt LEDs. Beide sind auf die für die Pflanze essentiellen Lichtspektren eingestellt, was sich für das menschliche Auge als ein pinkes Helligkeitswunder herausstellt. Es schmerzt, ich hätte besser nicht reingeschaut..

Fogger Ein Wundergerät, es macht aus flüssigem Wasser trockenes Wasser in Nebelform. Damit kann ich die Wurzeln über die Luft mit Wasser und Nährstoffen versorgen, ich habe es vorhin schon angedeutet, mehr dazu gleich.

Wasserbecken Hornbach hat ein beachtliches Sortiment an *Zementmischeimern*, einen davon habe ich erworben.

Sensoren Um zu wissen, wie fest ich Lüften muss, brauche ich Sensoren. Zu meinem Erfreunis sind sie äusserst günstig.

Microcontroller Die meisten Microcontroller, namentlich Arduinos und einen Raspberry Pi hatte ich dank meinem *Geekvater* schon zuhause.

Sonstige Elektronik Kleinere Bauteile für die verschiedenen Schaltungen hat ebenfalls die Kellerwerkstatt meines Vaters beigesteuert. Darunter befinden sich vor allem Drähte, Kabel, Widerstände, Transistoren und Dioden.

Lüfter Aus der edlen CPU-Lüftersammlung meines Vaters konnte ich vier Gleichstrom-Lüfter ergattern.

Pflanze Das ganze Projekt würde ohne Pflanzen nicht funktionieren. Beim *Gartenmeier Center* bescherte ich mich mit Tomatenstecklingen, zweimal, denn das Projekt hat länger gedauert als angenommen, wodurch der erste Satz Stecklinge leider dran glauben musste.

Nährstofflösung/Dünger Mit einer Universalnährstofflösung (die sogar den richtigen pH-Wert mit sich bringt) kann ich nichts falsch machen. Auch darauf komme ich nochmal zu sprechen.

4.4 Trockenes Wasser

Durch Jeb Gardner wusste ich von der Hydroponik schon vorher und durch Zufall stiess ich auf Fogponics, was sich als recht eleganten Ansatz herausstellt. Man zählt Fog-, respektive Aeroponik als eine Unterkategorie der Hydroponik. Konkret geht es dabei um die Bewässerung der Pflanze, während sich ihre Wurzeln *in der Luft befinden*, also weder im Wasser, noch in einem Substrat. Düsen werden gerne in grossen, industriellen Gewächshäusern verwendet, doch nach endloser Recherche musste ich mir eingestehen, dass Düsenn einfach keine Lösung sind, ich bekam es einfach nicht hin, in absehbarer Zeit die richtigen Düsen und das Röhrenset dafür zu bestellen. Ich erweiterte also meinen Horzizont. Fogponics sind genial, man nimmt den Fogger (ein recht schweres, wasserfestes Metalding mit fünf Porzellanscheiben), setzt ihn ins Wasser und schaltet ein. Die Porzellanscheiben vibrieren mit 1.7MHz, also etwa etwa alle 0.6 Microsekunden einmal, und spicken somit feinste Wassertröpchen in die Luft - Ein Nebel entsteht. Wenn im Wasser dann noch die essentiellen Nährstoffe gelöst sind, hat man einen Nährstoffnebel! Die Tröpfchen sind ca. ein Zweihundertstel Millimeter, was es ihnen ermöglicht, direkt in die Wurzeln einzudringen, ohne dabei an der Wurzelwand zu kondensieren. Eine so direkte Nährstoffaufnahme ist *giga-effizient* und nur bei solch kleinen Tröpfchen möglich.
Kleiner Seitenfakt: Beim Einatmen von solch kleinen Tröpfchen muss man Husten, denn sie ähneln mehr einer Art Staub als Wasser. Das ist wahrscheinlich auf ihre eingeschränkte Kondensfähigkeit zurückzuführen, was sie für uns *nass* machen würde. Diese Technologie wurde durch die NASA markttauglich gemacht und wird deshalb heutzutage auch in der Raumfahrt eingesetzt.Ich verwende es nun um Pflanzen *intrawurzulös* mit trockenem Wasser und Nährstoffen zu versorgen, auch gut.

4.5 Belüftung..

Die Lüfter waren ebenfalls ein lustiges (*trockenes, kaltes Lachen...*) Teilprojekt:
Um ein Lüftungssystem zu bauen, braucht man offensichtlich einen Lüfter, am besten gleich mehrere. Es gilt zwischen EC- und DC-Lüftern zu unterscheiden, wobei EC-Lüfter generell leiser und energieeffizienter sind. Naheliegenderweise hätte ich gerne einen Lüfter der EC-Gattung, doch da mein Vater wie schon erwähnt ein leidenschaftlicher Sammler jeglichen Elektroschrotts ist, konnte ich mich einfach im Keller bedienen. Um es jetzt kurz zu fassen: Bei der Belüftungskonstruktion habe ich meine Seele verloren, es ist das Teilprojekt mit dem meisten Aufwand. Das mit der Seele ist sinnbildlich gemeint, versteht sich. Im Nachhinein habe ich mich wirklich gefragt, ob es dieser Aufwand nicht die 120Fr. wert gewesen wäre, was so ein EC-Lüfterchen kostet. Ich erläutere das mal: Einen DC-Lüfter (Gleichstrom) runterzuregeln, so dass er nicht volle Pulle lüftet, ist eine Wissenschaft für sich, ich bin recht gut geworden, doch zu welchem Preis? Einfach die Spannung runterzudrehen geht nicht wirklich, denn sie ist nicht proportional zur Umdrehungszahl. Was bleibt mir also? Ganz einfach - *Pulsweitenmodulation.* Hier das Prinzip: Man versorgt den Lüfter entweder mit 12V oder mit 0V, aus dem zeitliche Verhältnis zwischen diesen beiden Zuständen erfolgt die effektive Lüfterleistung. Ist der *Puls* also 75% eines ganzen Zyklus', läuft der Lüfter auch mit 75%, klingt einfach, ist es aber nicht. Die Krux liegt in der Frequenz dieser Zyklen, es macht nämlich einen Unterschied, ob man ihn für 7.5 mit 12V versorgt, dann 2.5s Pause macht oder ob diese Zyklen mit einer Frequenz von

40'000Hertz aufgetischt werden. Beim ersten Beispiel stockt der Lüfter und beim Zweiten wird der Saft so unglaublich schnell an und abgestellt, dass der Motor gar nicht mehr zwischen An und Aus unterscheiden kann. Voilà, Pulweitenmodulation.
Die Theorie an sich ist einfach, denn hier kommt die technische Umsetzung: Die von mir verwendeten Microcontroller besitzen eine PWM-Funktion, jedoch hat diese eine Maximalfrequenz von 1kHz, also 1000Hertz und das ist zu langsam, denn damit *kann* der DC-Motor noch zwischen An und Aus unterscheiden und es kommt zu hörbaren Vibrationen, was einem nach 2min recht auf den Wecker schlägt. Es muss demnach eine andere Lösung her. Der Arduino (der Microcontroller) ist rein technisch gesehen dafür ausgerüstet, höhere Frequenzen zu takten, ein gutes Beispiel dafür ist der Prozessor, der, wenn er auf 1kHz laufen würde, *schnarchend* langsam wäre. Dem ist natürlich nicht so. Und auch wenn der eingebaute ATmega32U4 Prozessor mit seinen 16MHz das weitaus schnellste Glied auf der Platine ist, weilen die internen Timer-Register nicht viel hinten. Die C++-Bibliothek *TimerOne.h* beherbergt die nötigen Werkzeuge. Man kann sich einen eigenen Timer stellen, sagen wir auf 25 Mikrosekunden, und wenn immer dieser Timer abläuft, schalten wir entweder an oder aus. So ungefähr lautet das Prinzip, effektiv existiert aber schon eine eigens dafür entwickelte Funktion. Man muss nur den Pin angeben, die Frequenz vorher eingestellt haben und sagen, welches Verhältnis man wünscht. Perfekt für meine Ansprüche.
Soviel zum Softwareteil, der Hardwareteil ist auch eine rechte Herausforderung. Ich brauche für jeweils vier Lüfter eine Schaltung, welche den Strom auf einer so hohen Frequenz an- und ausschalten kann. Also gut, Platine her, Lötkolben und Oszilloskop raus und runter in die Werkstatt. Ich habe für diese Schaltung etwa einen Monat gebraucht, natürlich war ich nicht jeden Abend dran, doch die alleinige Dauer, welche ich vor dem Oszilloskop verbracht habe, um die richtigen Verhältnisse zu sehen, hat mir die Seele wahrlich etwas betäubt. Im Kapitel Perspektiven gehe ich näher auf Gelerntes ein, deshalb verrate ich hier einfach, dass am Ende der sogenannte *PWM-Controller* geboren war.

4.6 Es soll hell sein

Die Belichtung ist um unser allen Dankbarkeit um einfaches simpler. Nichtsdestotrotz muss ich kurz ausholen:
Um sowohl die Belichtung, als auch die Bewässerung zu steuern, brauche ich eine Schaltung, welche mal an und mal aus geht - und das bitte in einem von mir vorgegebenen Takt. Swoosh, geboren ist der *Relay-Controller*! Auf den Bauprozess dieses Babys wird später eingegangen, Sie sollen einfach wissen, dass es recht gut funktioniert. Die ist Belichtung echt keine Hexerei, die LEDs gehören einfach eingesteckt und der Relay-Controller vorprogrammiert und es läuft. Ein solcher LED-Schweinwerfer kostet 65Fr und ist rein von der Leistung gesehen (50W) für meine zu belichtende Grundfläche von 0.36m^2 hell genug. Warum habe ich also zwei davon? Erstens sehen diese LEDs total schön aus - sie haben auch so ein cooles Gehäuse, sollten Sie mal sehen, und andererseits hat mich - ob relevant oder nicht - die Frage beschäftigt, wie lang diese Babies denn so halten. Das Internet gibt keine genaue Angabe, deshalb habe ich mir, so genial wie ich bin, überlegt, dass ich, wenn ich den Arbeitsaufwand auf zwei verschiedene Lampen aufteile, die Haltbarkeit von diesem Teil der Machine wohl doppelt so hoch sein muss. Und falls eine den Geist aufgibt, habe ich Ersatz da. Ich habe in der Einleitung schon von Sorgenfreiheit gesprochen, Ersatzteile sind ein Aspekt davon. Wer jetzt denkt, dass ich einfach beide einschalten kann, um meine Pflanzen zu boosten, muss ich leider enttäuschen, so einfach funktioniert die Biologie nicht, mehr dazu in *Technische Hintergründe*.

4.7 Das Hirn!

Irgendjemand, oder besser gesagt irgend*was* muss diesen gutmütigen Controllern eine Richtung weisen, ihnen sagen, was sie überhaupt zu tun haben. An diesem Punkt würde ich mich gerne bei Herrn Upton, dem Gründer der Raspberrypi-Foundation bedanken, ohne ihn wäre dieses ganze Vorhaben nämlich recht *hirnlos*, im wahrsten Sinne des Wortes: Ich verwende einen RaspberryPi als Zentrum für alle Steuerprozesse. Der Raspi (umgangssprachlich auch *Pi*) beherbergt alle Funktionen welche man von einem normalen Computer erwarten würde, nur kleiner, denn er passt Problemlos in die Hosentasche. Die Leistung ist logischerweise geriner als bei Computer mit normalen Dimensionen, doch Rechenleistung steht bei vielen Projekten erst an zweiter Stelle. Ich z.B. brauche einfach ein kleines Linux-Compüterchen und genau das ist der Pi. Mit Linux bin ich gleichermassen aufgewachsen wie mit einem Fahrrad, es war einfach da. Deshalb ist es für mich kein Problem, dieses Schätzchen auf meine Bedürfnisse anzupassen. Da es in diesem Kapitel generell nur um die erste Herangehensweise geht, fasse ich mich kurz, denn der Pi kommt zeitlich erst recht am Schluss ins Projekt: Er steuert die Sachen, welche andere Sachen steuern und nimmt Inputs vom Benutzer - das bin Ich - entgegen.

4.8 Meiers Tageshit: Universaldünger, leicht säuerlich.

Dieses Unterkapitel mag Sie etwas überraschen, doch es sei angemerkt, dass sich mir dieses Thema recht lange zur Last machte. Die Frage, *wie* und mit welcher Nährstofflösung (eng. *Nutrientsolution* oder einfach *Solution*) ich meine Pflänzchen füttere, hat mich lange geplagt. Eine einfache Antwort darauf lässt sich im Internet **nicht** finden. Auch Jeb hat darauf keine wirklich einfache Antwort, denn er misst die Menge an Kaliumsubstraten mit einer Küchenwage und schmeisst sie dann in den Wasseareimer. Da ich ein Laie in Sachen Substratmixtur bin, brauche ich also eine simplere Lösung, welche mir das Internet nicht gab. Bei einem meiner Besuche in Meiers Gartenmeiercenter in Dürnten stiess ich auf *the man himself*, Herr Meier.
Wahrlich ein Engel; Er stand hinter dem Kundenberatungsthresen und als ich endlich dran war, ging es keine halbe Minute und er wusste *welche Solution* ich *wo* zu suchen hatte. Gesagt, getan, gefunden, in den Händen hielt ich eine grüne 1L Flasche mit Wunderzeugs. Sie enthält, soweit mich meine eigenen Sinne und Kenntnisse nicht trügen, um weitaus mehr Nährstoffe, als alle anderen Dünger aus unserem Gartenschuppen. Das Zeug mit Leitungswasservermischt soll angeblich auch direkt denn perfekten pH-Werte haben, 5.5, also leicht auf der sauren Seite. Zu so etwas sage ich nicht Nein.

4.9 Löcher im Boden

Die Pflanzen finden Halt in sogenanannten Steinwolleblöckchen, diese widerrum stehen in kleinen Netzbechern/Netcups und *diese* hängen durch ein Brett durch, sodass die Würzelchen im Nährnebel hängen. Das Brett ist aus beschichtetem Holz. Ich habe es zusammen mit dem Zementeimer im Hornbach gefunden. Zum Leid jeder Nase stank diese Brett so wahnsinnig bestialisch nach dieser chemischen Beschichtung, dass ich mehr als einmal keinen Appetit verspürte, nachdem ich die Growbox aufmachte, wo ich alles Material verstaute. Es stank, aber es nicht unhygienisch wie gewisse andere Sachen, doch es stank. Eines Tages wusste ich, dass ich Löcher in das Brett machen musste. Logisch, das ist ja der Sinn des Brettes. Also ab in den Keller und - Hmm, wie macht ohne es je einmal gemacht zu haben 8cm breite Löcher in chemisch verstärktes Holz? Die Werkstatt offerierte mir einen Bohrer und ein Löcherbohrset, eines mit einer dieser Scheiben, vo man verschiedene, kreisrunde Sägeblätter einstecken kann. Das Problem dabei war jedoch die vorhandenen Durchmesser, 6cm war das grösstmögliche. Mir fehlten

also 2cm. Haben sie schon einmal von einer *Fräse* gehört? Nun, mein Vater hat so ein Ding und ich habe es gefunden. Dieses Stück Maschine funktioniert im Prinzip wie ein normaler Bohrer, aber seitwärts. Anstatt nach unten zu bohren, ist der Fräsenkopf an der Seite mit Klingen bewaffnet, welche sich sterbenshungrig ins Holz fressen. So fest, dass man das ganze Konstrukt leicht verlieren kann, wenn sich die Fräse erst mal festgebissen hat, gibt es kein entkommen. Das ganze Gerät, welches man an zwei (!) massiven Haltern hält, rennt praktisch durchs Holz und es stäubt, oh wie es stäubt. Für die Atemmaske war ich dankbar. Diese Fräse ist ein Erlebnis, das sieht man an der tiefen Kerbe in einem der Löcher, der Netzbecher fällt fast raus.. Den Fräsenabend werde ich nicht so schnell vergessen, ich bin noch nie mit so viel Sägestaub aus dem Keller gekommen, meine Augen enthielten mehr Sägemehl als Tränenflüssigkeit.

5 Arbeitsprozess

Wie genau bin ich wann vorgegangen? Auf diese Frage gibt es viele Antworten, hier sind die wichtigsten. Ich unterteile den Arbeitsprozess in Kategorien, welche widerrum Beispiele enthalten.

5.1 Recherche

Am Anfang jeder Maturarbeit steht die Recherche. Ich habe, vom Interne ja schon inspiriert, vorallem im Web recherchiert. Mein Wissen, sei es Bewässerung, Optimaltemperatur, Nährstoffe, Belichtungsmethoden, Bewässerungssysteme oder einfach Pflanzen selbst, kam zu einem *sehr* grossen Anteil von Cannabis-Blogs, -shops und -foren. Jeder, der meinen Suchverlauf inspiziert, wird direkt glauben, dass ich vor habe, ganz gross ins Business einzusteigen. Ich kann nicht verleugnen, dass das *Konstrukt* cannabis-bezüglich durchaus Potenzial hätte, aber ich segle lieber satt als high. Deshalb spezialisiere ich es bewusst *nicht* auf sensible Marihuana-Pflänzchen, denn es sei auch angemerkt, dass diese Praxis eine Wissenschaft für sich ist, welcher ich mich nicht auch noch widmen kann.
Recherche fand eigentlich konstant statt, wenn auch nicht immer aufgrund der gleichen Motivation: Am Anfang wollte ich wissen, wie ich so eine Growbox baue, die Bauteile standen also im Zentrum. Dann hatte es mich mit der Physiologie der Pflanzen an sich gepackt und am Ende verbrachte ich 80% meiner Web-Zeit auf *stackoverflow.com*, wem diese Webseite ein Name ist, schmunzelt jetzt womöglich :) - Wem nicht, hier ist die Erklärung: Wenn ein Programmierer auf eine Fehlermeldung stösst, welche er nicht versteht, gibt es nur eine Website, welche bei der darauffolgenden Google-Suche *praktisch immer* zuoberst erscheint, es ist *stackoverflow.com*. In der Humorszene (richtig, ich meine Memes) der Coding-Communities hat sich dieser Witz schon recht gefestigt. Hin und wieder musste ich auch am Ende nicht-programmiertechnische Sachen googeln, aber das sind wie gesagt nur etwa ein fünftel, wenn nicht *noch* weniger. Recherche, so trist es auch klingen mag, ist in Sachen wie Debugging anstrengend und nur das Mittel zum Zweck, jedenfalls für mich. Es hilft, den jetzigen Fehler zu finden, aber ehrlich gesagt ist das auch schon alles. Zum Schluss fand ich die Zeit einfach nicht mehr, mich über Pflanzen zu informieren und alles aufzuschreiben. Ich habe anfangs noch alles auf Google-Docs zusammengetragen, habe die Date *MA-Knowledge* genannt, was für schöne Erinnerungen. Tja - Mit der Suchleiste baue ich keinen Relay-Controller. Dafür brauche ich Kabel, Steckdosen, Relays und einen Lötkolben - und um den geht es jetzt.

5.2 Löten, testen, löten

Löten ist, wenn man mit einem heissen Metalkolben Lötzinn, das ist ein Zinndraht mit Flussmittel in der Mitte, zwischen zwei Kontaktstellen verteilt, sodass anschliessen Strom durchfliessen kann. Ich habe zwei eigene Schaltungen gebaut, den PWM- und den Relay-Controller. Beide besitzen als Grundbaustein eine Platine (Eine Plastikscheibe), auf welche ich die Stromleitungen drauflötete. Das an sich ist super cool; Sich selbst einen Schaltplan zu zeichnen und diesen dann in die Realität umzusetzen macht einfach Spass. Mein Vater ist ehemaliger Elektroniker und ich habe mich wie ein Lehrlich gefühlt, als er wie ein Prüfer mit dem Finger versucht hat, meine Lötstellen zu durchtrennen, wenn sie hielten war er zufrieden, wenn sie nachgaben und aufsprangen, bedeutete das, dass sie auch keinen Strom durchlassen würden - oder nur begrenzt. Es war die härteste aber effizienteste Art, meine Kreationen zu testen. Er vertritt generell die Arbeitsmoral, etwas *einmal, dann aber richtig* zu machen. Und das hat er mir beigebracht, es gibt keine patzige Lötstelle in keinem Controller, jedes Tröpfchen Zinn sitzt.

Der Prozess bestand meistens darin, dass ich im Keller sass, lötete, testete, wenns gut kam, den Fehler selbst fand, nochmal neu lötete und es ihm dann zeigte. Oft musste er mir aber helfen. Ich muss zugeben, die Souveränität und die Art, in welcher ein erfahrener Elektroniker an eine nicht sauber funktionierende Schaltung heran geht, ist beeindruckend. *Ist die Masse überall gleich? Ist die Diode verkehrt drin? Schau mal diese Lötstelle hier.* Das alles sind oft gehörte Sätze und wenn ich dann mit einer funktionierenden Schaltung hochschritt um sie ihm zu zeigen, war das Gefühl ein besonders schönes. Das Lob eines stolzen Vaters. Ich kann mich wirklich glücklich schätzen, einen Pro im Haus zu haben.
Was auch recht half, war die ungeheuer grosse Sammlung an Elektroteilen, welche er noch aus alten Tagen hat. Er hat alle Kisten sortiert, ich konnte ihn einfach fragen, wo die Transistoren sind und er zeigte mir den richtigen. Von Transistoren habe ich vor diesem Projekt zwar schon gewusst, allerdings blieb mir ihre Bedeutung für die Menschheit verborgen. Das änderte sich eines Tages, recht am Anfang, als er mir gefühlt den ganzen Keller erklärte, angefangen von Dioden, über Transistoren, bis hin zu Spulen. Nicht dass ich jetzt noch besonders viel weiss, dass muss ich ehrlich gestehen, doch es ist superspannend und recht nützlich zu wissen. Ich weiss jetzt zum Beispiel, wie der Transport via Hochspannungsleitungen funktioniert, respektive *warum* es Hochspannung ist - weniger Verluste, so einfach ist das.
Ich habe noch etwas anderes gelernt, etwas schmerzhaftes. Aber ganz von vorn. Ich sass eines Nachmittags im Keller, lötete und testete den PWM-Controller und brauchte eine Stromversorgung für die 12V-Lüfter, also brauchte ich ein Netzteil. Netzteile sind oft schwer, was bedeutet, dass es welche sind, welche mit Spulen funktionieren. Ich hatte ein solches rumliegen und prüfte die Ausgangspannung, 12V, passte also. Einmal eingesteckt wollte es anschalten, es hatte einen Schalter dran, welcher jedoch nicht *im* Gehäuse, sondern ausserhalb war. Das Netzteil hatte nämlich kein Gehäuse, es war einfach ein Paket Spulen, auf jeder Seite ein Kabel, eins mit 230V und das andere mit 12V und dazu kam der Schalter. Nun ist es so, dass sich der Schalter auf der 230V-Seite befand, also da, wo es richtig weh tut - und das tat es. Der Schalter ist nicht ganz abisoliert worden, beide 230V-Pins waren offen, ich wollte den Schalter mit einer Hand anmachen, musste ihn also gleichzeitig stabiliseren, damit ich draufdrücken konnte und ich berührte dadurch direkt die beiden Pins. Es war nicht lebensgefährlich, der Strom floss nur durch meinen Finder, aber es zuckte trotzdem recht schnell. Ich bin noch nie so schnell aufgestanden, wie ein Blitz. Den Adrenalinflash den ich nachher empfand können sich nur andere 230V-Opfer vorstellen; *Huii, das zuckt!* Danach lacht man hysterisch und ist ganz aufgeputscht. Ich muss jetzt noch lachen, einfach irre dieses *Zuck*.

5.2.1 Meine Hassliebe zu PWM

Um nun etwas konkreter zu werden, was ich im Keller genau so trieb, werde ich Ihnen nun kurz den Frust beschreiben, welchen meine PWM-Odyssey in mir auslöste. Die Situation ist simpel: Man sitzt im dunklen Keller, vor sich der Laptop, der ganze Tisch voller Drähte, Kabel und winzigen Elektroteilchen, es ist *voll*, eine richtige Unordnung. Das ist normal. Ich teste die Funktionalität der neuen PWM-Schaltung und es geht nicht. Ich sende also genau die Bytes über die serielle Schnittstelle zum Arduino, welche er in PWM-Signale für einen bestimmten Lüfter umwandeln soll. Aber der Lüfter dreht nicht, oder immer noch gleich. Das System hat also nicht funktioniert. Die Frustration ergibt sich nicht aus dem simplen *Nicht-Funktionieren*, sondern aus dem *Wo-ist-der-Fehler?*-Problem. Denn mit einer seriellen Schnittstelle, die sind heikel, glauben Sie mir, einem Arduino, der ist auch heikel, und vier Transistoren hat man reichlich potenzielle Fehlerquellen. Das serielle Signal, also ein paar Bytes müssen vom Arduino registriert, gelesen, geparsed (mehrere Werte voneinander unterscheiden), ins richtige Format konvertiert und dann an die PWM-Funktion weitergegeben werden. Diese an sich ist recht

schlicht, doch dann erst beginnt der elektronische Teil, also die Verteilung der vier PWM-Signale auf die Transistoren, welche den Hauptstrom für die Ventilatoren regeln. Wenn eine Lötstelle nicht richtig verbindet, ist die ganze Leitung unterbrochen, das heisst ich muss neu löten. Und wenn ein ganzer Transistor aufgrund einer zu hoch angesetzten Spannung durchbrennt (ich nenne es *frittieren*), darf man alle drei Pins des Transistors ablöten und einen neuen dranlöten. Puh, das sind eine Menge potenzieller Fehler. Aber ich habe es geschafft, der PWM-Controller tut genau das, was ich ihm jetzt sage. Das ist äusserst erfüllend, auch wenn ich jedesmal noch ein Bisschen Angst habe, auf ein unidentifizierbares Problem zu stossen. Eine technisch genauere Beschreibung dieser Schaltung finden Sie im nächsten Kapitel.

5.2.2 Relays als nette Abwechslung

Im Vergleich zum PWM-Karussell war der der Relay-Controller um einiges einfacher, wenn man die Software ignoriert... Die Relays, für alle non-Techies, sind wie Lichtschalter, sie lassen entweder Strom durch oder nicht, je nachdem, ob man an einem dritten Pin eine Spannung anlegt oder nicht - bekanntlich 5V. Der Lötaufwand war hier um einiges kleiner. Denn ich verwende, weil ich mit 230V arbeite, dickere Kabel, also normale Haushaltskabel, genau die, welche in ganz neuen, noch lampenlosen Häusern immer aus der Decke hängen. Sie sind um einiges steifer als meine bisherigen *5V-Drähtchen*. Für den Arduino brauchte ich eine eigene Stromversorgung, welche nicht 230V sondern 5V entspricht. Eigens dafür bestückte ich die Platine mit einem kleinen Netzteil, es ist wirklich niedlich. So richtige Schwiriegkeiten gab es bezüglich der Löt- und Elektronikpraxis beim Relay keine. Was für eine nette Abwechslung.

5.3 Oszilloskopisches Wunder

Das, lieber Leser, ist eine eigens von mir benannte Emotion, welche sich ergibt, wenn man das erste Mal einen Fehler findet, indem man sich hinsetzt und versucht, die Stromsignale *selbst* zu verstehen. Wenn man bedarf für ein solches Gerät hat, weiss man, dass man ganz unten angekommen ist, bezüglich den elektronischen Sphären - Mental ist man nämlich ganz oben. Ein *Oszi* ist ein Gerät mit einem Bildschirm und sogenannten *Klemmen*, welche man an bestimmten Punkten einer Schaltung ansetzt, um anschliessend den Verlauf der Spannung beobachten zu können. Es ist ein Wundergerät, welches jedem richtigen Elektroniker das Herz schneller schlägen lässt. Das beste ist wohl, dass ich jetzt weiss, wie man es bedient. Es hat mir womöglich den Rest meiner Seele gerettet, nachdem ich einen grossen Teil schon verloren hatte und mein Vater mir sagte, wo diese Wundermaschine versteckt ist. Ich benutzte das Oszi für die Veranschaulichung der PWM-Signale. Das funktioniert sehr gut, denn die Verhältnisse lassen sich direkt ablesen. So wusste ich immer, wie stark das PWM-Signal an einem Punkt ist. Bei der Fehlerfindung in einem Labyrinth wie meinem PWM-Controller schätze ich das Oszi besonders, es wirkt wie eine Leiter, mit welcher man auf die Mauern draufklettert und über die Gänge in Richtung Ausgang balanciert. Das Oszilloskop ist eine Art des *Debuggings*, einer Praxis, welche einen grossen Teil der Arbeit ausmacht.

5.4 Code

Der Softwareteil des *Konstrukts* ist nicht zu unterschtzen. Ich habe 95% der Programmierherausforderungen mit C++ und Python erledigt.

5.4.1 C++/Arduino

Der Titel mag etwas verwirren, denn die Low-Level-Programmiersprache für alle Arduinos ist eine leichte Abänderung der Sprache C++, welche wiederrum eine objektorientierte Weiterentwicklung des Klassikers *C* ist. Mit dieser Sprache habe ich die *Backends* programmiert, namentlich die Arduinos, ich verwende zwei, einen für jeden Controller. Beide müssen Signale entgegennehmen können und dann dementsprechend handeln. Am Anfang der Arbeit, ab dem Moment, wo ich den ersten Arduino herausnahm, lag meine Hauptbeschäftigung fast nur darin, herauszufinden, was ich alles erreichen kann *und wieviel meiner Seele ich dafür opfern muss*. Spass, aber ich glaube Sie verstehen langsam, dass mir die Komplexität von Backend-Problemen leicht zuwider ist. Ich erläutere das mal: *C* und das davon abstammende *C++*, respektive *Arduino* sind sehr *lowelevel*, d.h. sie arbeiten nah am Prozessor, für mich als Programmierer bedeutet das, dass ich mich um mehr Probleme sorgen muss, als ich es mir von *highlevel* Sprachen wie Python gewohnt mit. Ein kleines Beispiel habe ich hier. Während man bei Python die Deklarierung einer Variable nicht mehr nötig ist,

```
number = 5
```

muss man bei C++ höllisch aufpassen, nichts falsch zu machen, sei es die Deklaration mit `int` oder das `;` am Ende jeder Zeile.

```
int number = 5;
```

Das ist nur einer vieler kleiner Syntaxfinessen, welche einem C++ abverlangt, für mich aus der Python-Ecke ist das kompliziert. Nichtsdestotrotz ist die MA nicht das erste Mal, bei dem ich C++ programmiere, ich kannte auch Arduinos schon davor und habe mit ihnen herumgespielt, z.B. um LED-Streifen anzusteuern. Komplett verloren war ich also nicht. Zudem kommt noch, dass mein Vater jahrelange Erfahrung mit *C* hat. Mich beeindruckt besonders der Fakt, dass er seinen eigenen C-Compiler geschrieben hat. *Wie macht das denn?* Wie baut man einen kompletten Compiler nur in Assembly, einer Sprache, die den Namen *Sprache* nichtmal mehr richtig verdient.
Den kompletten C++-Code für Arduinos schreibt man in der eigens dafür vorgesehenen Arduino-IDE. Diese hat den Vorteil, dass man das Script, wenn der Arduino per USB mit dem Gerät verbunden ist, direkt hochladen kann. Der Workflow an sich ist somit sehr schnell, was ich schätze. Das grösste Stück Code was ich schrieb, waren die sogenannten *Relay-Timer*, sie steuern jedes Relay unabhängig voneinander. Der Kniff bei der Sache, wenn man das Problem mithilfe von C++ löst, sind die dynamischen Updates. Ein Beispiel: Angenommen ein Relay ist für 20 Sekunden an, lässt also Strom durch, und für 20s aus, lässt keinen Strom durch, und ich verspüre das Verlangen, die Auszeit auf 10s zu ändern; Dann muss die Software, welche den Relays mitteilt, wann sie zu schalten haben, das mitbekommen und seine Konfiguration ändern. Mit C++ ein Protokoll zu erstellen, womit man über die serielle Schnittstelle Kommandos absetzt, in diesem Beispiel eine neue Konfiguration für ein bestimmtes Relay, ist durchaus machbar, hab es ja selbst gemacht. Allderdings, und das tut mir in der Seele leid, denn ich verbrachte noch so einige Stunden vor der Arduino-IDE, lässt sich ein so komplexes Problem auf Python einfach besser angehen. Nicht nur sind die Debugging-Möglichkeiten, also das Fehlerfinden, grösser, sondern ich bin schlicht einfach ein besserer Python- als C++-Coder. Was ich also tat, war, das komplette Relay-Timer-Script auf Python zu übersetzen.

5.4.2 Python

Nicht, dass das Relay-Timer-Script lange gehalten hätte, ich hab's zwei Tage nach der Übersetzung von C++ wieder verworfen und ein komplett neues Konzept zu realisieren versucht. Zu solch code-lastigen Beispielen wie dem Relay-Timer und dem vorher erwähnten PWM-Controller erfahren sie mehr im nächsten Kapitel *Technische Hintergründe*.
Python ist bekanntlich eine, wenn nicht *die* einfachste Einsteigerprogrammiersrpache, vorausgesetzt man will auch wirklich etwas anstellen. Gerade in Felder wie *künstlichen Intelligenzen* und *Big Data* wurde Python in den letzten Jahren immer stärker. Mittlerweile lässt sich fast alles damit machen. Eine Opensource-Bewegung namens *pySerial* hat es sich zum Ziel gemacht, auch noch in den Lowlevel-Sektor vorzudringen, was für mich bedeutet, dass pySerial vielleicht sogar eine ernste Alternative zu C++ dargestellt hätte. Effektiv weiss ich das nicht, denn ich kam mit C++ genug gut klar, als das ich mich hätte umgewöhnen wollen. Python hat also, wie schon mitgeteilt, einen grossen Teil der Anforderungen meines Vorhabens erfüllt. Der Workflow ist etwas anders, das Debugging ist einfacher, vorallem mit IDEs wie *PyCharm*. Für alle die das nicht wissen, das ist der Tesla unter allen Python-Entwicklungsumgebungen und für meine Vorhaben noch fast zu gross. Das bedeutet, dass ich den letzten grossen Teil Python nicht mit PyCharm sondern mit Sublime bestritt. Sublime ist eine gute Mittellösung, er hat einen eingebauten Compiler, was bedeutet, dass der Code direkt in der IDE ausgeführt werden kann, und ausserdem sind die Editorfunktionen genau nach meinem Geschmack. Jeder Coder entwickelt nach ein paar Jahren so seinse Präferenzen, welche sich natürlich immer wieder ändern können, aber *mein Gott*, das Sublime-Team hat da ein paar feine Sachen eingebaut! Die intensiven letzten Tage vor der Abgabe habe ich mit Sublime verbracht - Unter Zeitstress muss eine IDE *alles* raushauen können, vorallem *Refactoring*: Wenn man dann einfach mal schnell den Namen einer bestimmten Variabel oder Funktion im *gesamten* Projekt flux ändern kann, dann, lieber Leser, *ist das Premium.*

Eine rechte Lobeshymne an Sublime, Sie merken es schon. Was aber *genau* habe ich mit Sublime genau gecodet? Einen Webserver. Schonmal von *Flask* gehört? Das ist eine ganz feine Kreation: Ein Pythonmodul, mit dem man, wie erwartet, einen kompletten Webserver aufsetzen kann. Serversprachen wie PHP verlieren damit recht ihre Wichtigkeit. Das trifft sich in meiner Situation recht gut. Ich muss wohl etwas ausholen, denn Sie wundern sich womöglich, warum genau mein Projekt bedarf einer Website hat. *Will er etwa schon sein Smart-Growbox-Business eröffnen?* Nein, das habe ich nocht nicht vor, dafür brauchts noch etwas Forschung. Ich könnte die Belüftungsregelung von einer KI übernehmen...
Eine Website brauche ich ganz einfach, um einen direkteren Einfluss in die ganze Regelungstechnik zu haben. Beispiele dafür sind die Relays oder manuelle Steuerung des Lüftungsystem. Mit einer Website kann ich von überall der Welt - auch von Korsika aus - auf die Zentralsteuerung meiner Pflanzenlebenserhaltung Einfluss ausüben. *Aber Jonas, was ist mit SSH?* (SSH ermöglicht es mir remote Textbefehle auf einer Maschine abzusetzen) Ja, liebe Bedenkstimme, SSH, wäre auch eine Möglichkeit, aber damit lassen sich keine Graphen darstellen und ein anständiges, grafisches Benutzerinterface ist einfach was feines. Ausserdem setzt *Flask* auch nur auf Python auf! Eine Website ist einfach cooler und die eben erwähnte Funktion von einer Graphendarstellung *wäre* auch golden. Ich muss Sie jedoch enttäuschen, bis jetzt haben es die Graphen noch nicht auf die Webseite geschafft. Der alleinige Fakt, dass ich mit Python einfach eine Webseite aus dem Boden stampfe ist schon ansehnlich, vorallem weil das ich das vorher noch nie gemacht habe. Und *nur* Python ist es dann eben doch nicht; Das Frontend, also das, was der Browser darstellt, ist ganz traditionell in HTML und CSS geschrieben - und es ist lange her, seit ich das letzte mal Webseiten designed habe. Es wird ja auch immer seltener, dass sich

eine Person hinsetzt und echte HTML-Tags in ein Dokument *töckelt*, wenn man die ungeheuere Menge an *Mach-deine-eigene-Webseite*-Angeboten beachtet. *Genau so eine Lösung wäre doch perfekt gewesen, ich hätte mir die Webseite einfach zusammengeklickt.* In erster Linie ja, das Frontend hätte ich so innerhalb eines Tages am laufen gehabt. Den Server allerdings mit den Arduinos zu verbinden, wäre irre schwierig bis unmöglich. Ich brauche meinen eigenen Server. YouTube enthält genug Tutorials, um in zwei Tagen eine solche Web-Instanz ins Leben zu rufen. Ich tat genau das. Zack, hatte ich einen Ordner mit allen nötigen Scripts, um mir selbst einen Seite zu hosten. Das Interface hatte ich auch in *absehbarer* Zeit am laufen. Naja, da fängt eben dann das Backend des Server an, welches die *wirklich* grossen Aufgaben des *Konstrukts* übernimmt

Die Website, wer es noch nicht erraten hat, wird vom Raspi gehostet. Dieses kleine Linux-Gerät kann so einiges, einen Webserver zu hosten ist, wenn man keine ressourcen-lastigen Webapps hostet, ein gutes Beispiel dafür. Jetzt habe ich den Webserver und ein Python-Script, welches die Relays ansteuert. Was fehlt noch?
Klar doch, der *Daemon*. Nicht erschrecken, das ist kein echter Dämon, das ist nur ein Hintergrundprozess der, deshalb der Name, *nie stirbt*. Er läuft 24/7. Der Webserver ist auf eine Art auch ein *Daemon*, er läuft auch immer, sonst wäre es eine seltsame Webseite. Eine Website mit Öffnungszeiten, komische Vorstellug... Der Daemon ist das Herz, das Hirn, nennen Sie es wie Sie wollen, des *Konstrukts*. Er läuft immer, weiss und steuert alles. Ich habe den natürlich auch in Python geschrieben. Der Webserver und das Relay-Script waren im Vergleich zum Daemon kleine Engel. Das ganze Ding ist so komplex aufgebaut, dass ich einiges meiner Seele opfern musste, bis ich den Dämon bezwang... Im Technik-Kapitel erfahren Sie genauer, wie er funktioniert, jetzt geht es weiter mit Git - dem nächsten Biest.

5.5 GitHub, the Beast

Der Titel soll nicht lügen. Wenn Sie jemals mit Git gearbeitet haben, dann stimmen Sie dem Titel zu, wenn nicht, erkläre ich es kurz. GitHub ist ein Onlinedienst, mit welchem man seine Code-Projekte hochlädt um mit seinem Team oder der Opensource-Community daran zu arbeiten. Es erlaubt einem eine recht fortgeschrittene Kontrolle über verschiedene Versionen des Projekts. Ich habe den kompletten Datenteil des *Konstrukts* mit Git verwaltet, also jedes Stück Code, Bilder, Konzepte und auch alle LaTeX-Dateien, respektive die Buchstaben welche Sie *jetzt* gerade lesen. Es funktioniert wie eine grosse Cloud, einfach besser und mit einer Menge mehr Funktionen. So viele, dass ich einen guten Teil meiner Zeit damit verbrachte, `git` *zu lernen*. Man achte auf die Kleinschreibung, denn ich rede vom *Bashprogramm* `git`. Was *Bash* bedeutet? Das ist die *born again shell*, eines der mächtigsten Stücke Technologie des Planeten, man braucht es, um in einer textbasierten Umgebung Befehle abzusetzen. Die *Bash* ist das Zuhause jedes richtigen Linux-Users. Das gesamte Thema *GitHub-Versioncontrol* ist eine Welt für sich. Eine Wissenschaft so wie die Materie der Elektronik, die Pulsweitenmodulation(PWM), die erwähnte Marihuana-Praxis derer, die es ernst damit meinen, dem Coden von Python, C++ oder HTML. Auf die erlernten Fähigkeiten wird im Kapitel *Perspektiven* noch eingegangen. Das es in diesem Kapitel um den Arbeitsprozess und meines Erachtens deshalb auch um den *Workflow* geht, sind hier die wichtigstes `git`-Befehle.

```
git clone
```

Gecloned wird ein Repo (eine Code-Sammlung, wenn man zum ersten Mal runterlädt.

```
git commit
```

Committen tut der Programmierer, bevor er die vorgenommenen Änderungen am Projekt hochlädt. Das Hochladen erfolgt mit

```
git push
```

Und falls es aus der Cloud gespeicherte Änderungen lokal zu synchronisieren, benutzt man

```
git pull
```

Um zwischen verschiedenen Versionen, auch *Branches/Äste* genannt, umherzuswitchen, benutzt man

```
git checkout
```

Dies sind die wichtigsten Befehle, welche ich während meiner gesamten Developement-Phase verwendet habe. Natürlich ist das nur die Spitze des Eisbergs, Git birgt noch so manche, deshalb nenne ich es auch *the Beast*. GitHub ist einer, wenn nicht *der* grösste Förderer der Opensource-Community, ich würde es sogar wagen zu behaupten, dass ein Code-Projekt, welches ich nicht auf GitHub finde, kein Opensource-Projekt ist. Sie merken, bezüglich OpenSource bin ich nicht minder leidenschaftlich wie mit der Refactoring-Funktion von Sublime: Es gibt einfach gewisse Dinge, welche einem Developer das Leben ungemein erleichtern. Das nächste ist ebenfalls eines davon.

5.6 Bash

Nebst all den Programmiersprachen, dem Lötkolben und dem Oszilloskop verdanke ich der *Bash*einen ungemein grossen Teil meiner Effizienz.

```
jonas@pop-os:~$
```

Jedes Mal, wenn ich das Terminal öffne, grüsst mich dieser Prompt, was dann folgt sind meistens Kommandos wie `cd` oder `ls`. Unter Zeitdruck werden sie so schnell in die Tasten genagelt, dass ich fast Mitleid mit meiner Tastatur habe. Auch hier wird mir jeder geneigte *Linux-Pinguin* einfach nur zunicken und kaum merklich schmunzeln. Die oben genannten Commands, für alle welche nicht vom Südpol stammen, sind die Augen und Beine in der Bash, mit `cd` läuft man durchs Dateinsystem und mit `ls` verschafft man sich einen Überblick über alle vorhandenen Dateien in einem bestimmten Ordner. Die Bash hat eine *so* grosse Präsenz in meinem Leben, wie kein anderes Programm, naja YouTube vielleicht: Jeden `git`-Command setzt man über die Bash ab. Man navigiert damit durch den Computer, tötet überflüssige Prozesse (aber ja nicht die Dämonen, versteht sich) oder programmiert mit `vim` oder `nano` kurz ein kleines Bash-Script. Ich habe genau einmal ein Bash-Script gebaut und zwar, als es mir zu blöd wurde, via GitHub meinen Daemon-Code auf den Raspi zu laden.

5.6.1 rsync

`rsync` ist Dropbox für die Kommandozeile/Bash, einfach cooler: Es kann verschlüsselt und rekursiv Daten von einer Quelle zum Ziel kopieren, alles was schon da ist, wird nicht doppelt kopiert. Es ist äussert flink und einfach zu bedienen, wenn man sich dann mit den *SSH-Keys* richtig anstellt, muss man nichtmal das Passwort des Zielrechners angeben. Das von mir vorher erwähnte Bash-Script hat alle 5 Sekunden meine Änderung, welche ich mit Sublime, Gott hab es seelig, am Dämonen (ich verwende absichtlich das böse Wort) vorgenommen hab, auf den Raspi kopiert. Was habe ich mir damit gespart? Einen kompletten `git commit -a && git push`

Befehl, zusammen mit dem *Commit-Kommentar*, den `git` jedes Mal von einem verlangt, wenn man die Änderungen committet. Rundherum sind das grob 10s und eine Menge unnötiger Tastenanschläge, denn ein GitHub-Commit für lediglich die Änderung eines Variabelnamens war in meiner Situation einfach ein *Overkill*. Vielleicht sollte ich den Workflow zusammen mit dem Raspi kurz erläutern: Den Dämon habe ich auf meine Laptop mit Sublime und Seelenopferung beschworen. Ich glaube wir kommen humorstechnisch langsam auf eine Welle. Wirklich funktioniert hat der Dämon aber nur auf dem Raspi, denn da sind auch die Sensormodule installiert, mit welchen der Dämon die Lüfterstärke berechnet (mehr technische Details im nächsten Kapitel). Da diese Module auf meinem Laptop keinen Sinn machten, war die Beschwörungsumgebung auf meinem Laptop, die Testumgebung aber auf dem Raspi. `rsync` war quasi die Geisterbrücke zwischen seiner Höllenschmiede und seiner Herrschaftsdomäne, um es ganz theatralisch auszudrücken. Das mit dem Schmieden ist so eine Sache, denn die Anzahl Bugs und Imperfektionen im Dämonen trieben mich fast in den Wahnsinn. Deshalb musste ich ja erst so oft hin- und herswitchen, *testen, fixen, testen, fixed* - ein höllisches Karussell, versteht sich. Bitte verzeihen Sie mir die, aus gutchristlicher Sicht, viellieicht respektlosen Bemerkungen bezüglich des ganzen Höllen-Tatas, ich bin nicht gläubig, also teufelsfürchtig und möchte keinem auf die Füsse stehen. `rsync` hat mir das Leben sehr erleichtert.

5.6.2 Secure Shell - SSH

Es ist so weit. Eine weitere Lobeshymne findet ihren Anfang. Ich habe das leichte Gefühl, dass dieses ganze Kapitel eine Danksagung ist. SSH bedeutet, wie im Titel schon angemerkt, *Secure Shell* und der Name wird der Software gerecht. Ich bin nicht hier, um Ihnen klar zu machen wie massiv der Einfluss von SSH auf die heutige Internetkultur ist, deshalb fasse ich mich kurz: Es ist wie der Sicherheitsgurt beim Auto. Denken Sie bitte darüber nach und lesen Sie dann weiter. Die Lobeshymne können Sie sich wahrscheinlich auch schon selbst zusammenreimen. Doch fragen Sie sich bestimmt auch, wie und warum ich dieses Protokoll einsetzte. Zuerst den genauen Sachverhalt: SSH benutzt praktisch jeder Sysadmin dieser Welt, um sich auf einem Remotesystem einzuloggen oder darüber Daten zu verschieben. Des weiteren wird das Protokoll selbst für *Portforwarding- und Tunnelingzwecke* eingesetzt, das muss Ihnen nichts sagen und ich werde es auch nicht erklären, aber googeln Sie es einfach. Das mit dem remote einloggen ist der Grund warum ich es benutze, hat man auf dem Zielsystem, in meinem Fall der Raspi, einen `openssh-server` am laufen (ja das ist auch ein Daemon :), kann man auf dem lokalen System einfach so tun, als hätte man eine Bash im Remotesystem - Man setzt also verschlüsselt Bash-Befehle auf ein anderes System ab. Simpel. Der Raspi hat zwar eine grafische Oberfläche, das RaspbianOS, doch ich hatte nur einen Monitor, welchen ich schon als Workstation für meinen Laptop verwendete. Deshalb war SSH der einzige Zuga- *Aber Jonas, was ist mit VNC?* Ach ja, die Bedenkstimme. Klar, VNC hätte auch funktioniert. Das ist SSH, einfach grafisch. (Benutzt den SSH-Tunneling-Aspekt, aber psst!) Ich habe es mit VNC versucht, es hat nicht geklappt und ich habe es aufgegeben. Wahrscheinlich wäre es auch sehr stockend gewesen und Lags kann ich in der letzten Developement-Phase nicht gebrauchen, ich verlange Sublime-Standarts und die hätte mir VNC sehr wahrscheinich nicht geliefert. Aber SSH tat es, so wie immer. Durch die textbasierte Oberfläche, müssen keine grafischen Daten durch die SSH-Leitung, also ist alles schneller, wenn man den Raspi dann noch mit einem Ethernet-Kabel betreit, also nicht über WiFi, fliegt man zum Mond. Die konventionelle Benutzung von SSH kannte ich schon von früher - ich bin ein Linuxkind - aber das *Keymanagement* hat während diesem Projekt eine neue Anekdote bekommen. Im vorherigen Unterkapitel `rsync` habe ich kurz von SSH-Keys geschrieben und ich möchte den unwissenden Leserseelchen genüge tun, indem ich Ihnen die Genialität hinter dem Private- und Publickeyprinzip von SSH erklären. Sie liegt nämlich in der

Einfachheit des Systems. Ich generiere als erstes meine Keys. Es sind zwei, ein privater und ein öffentlicher. Der öffentliche kann ich auf dem Raspi in eine Datei schreiben, womit er alle Daten, welche er mir schickt, verschlüsselt. Die von meinem öffentlichen Key verschlüsselten Daten werden, bei mir angekommen, von meinem privaten Key einfach wieder entschlüsselt. Deshalb sollte man den Privatekey auch nie weitergeben, denn sonst könnten andere Leute *meine* Daten entschlüsseln. Um bei `rsync` also nicht immer das Passwort des Raspis eingeben zu müssen, hinterlege ich dort meinen Publickey und mache mir um Sicherheitsrisiken keine Sorgen mehr. Der Datentransfer ist verschlüsselt und zwar mit so hohen Verschlüsselungsraten, dass schon bei einem, heute fast überall gebräuchichen *RSA-2048 Algorithmus* keine menschliche Technologie ausreicht, um ihn in absehbarer Zeit zu knacken. Ich behalte es mir vor, diese Aussage auch auf Quantencomputer zu beziehen, aus Unsicherheit ob meine Aussage korrekt wäre.

5.6.3 Tilix

Tilix ist weder ein Bashprogram, noch ein Protokoll. Es ist vielmehr ein Program für die Bash, ein grafisches Fenster, worin man die Bash benutzt. Das spezielle an *Tilix* ist das *Multiwindow-Feature*, anstatt dem traditionellen *Eine-Bash-Ein-Fenster*(von Tabs mal abgesehen) kann ich hier den ganzen Bildschirm ausfüllend ein Bashfenster neben dem andern haben, alles noch im Tilix-Fenster selbst. Damit habe ich mich in verschiedenen Bashfenstern per SSH auf dem Raspi eingeloggt, um parallel Kommandos abzusetzen und Daten auszulesen, z.B. die Statusmeldungen des Flask-Webservers. Es ist recht gemütlich, alle Terminals (alias Kommandozeilenfenster) auf einem Bildschirm zu haben.

5.7 Forschung

Ich habe nicht wirklich geforscht, nicht wissenschaftlich jedenfalls, ich habe lediglich verschiedene Setups aufgebaut um zu sehen, welches am besten funktioniert. Konkret ging es um die Bewässerungszyklen, also um die Frage, wie lange ich den Fogger ausgeschalten lassen kann, bevor ich ihn wieder anstellen muss, damit genug Nährnebel für die Würzelchen da ist. Sie sollen ja nicht austrocknen. Der Nebel, so trocken er auch ist, kondensiert nach einer Weile und die Trockenheit bekommt den Pflänzchen nicht gut.. Ich nenne es Forschung, weil es sich gut anfühlt, behaupten zu können, *dass man jetzt in den Garen forschen geht.* Ich hoffe Sie verstehen meine Kindlichkeit.¨

5.8 LaTeX

Wie man am Layout dieser Arbeit vielleicht schon gemerkt hat, schreibe ich alles mit LaTeX. Beim Rechercheprozess und im Gespräch mit meinem Vater stiess ich zuerst auf den Namen. Der Name ist *catchy*, ich hatte ihn zuvor also schon mal gesehen, mich aber immer gewundert, was es bedeutet. Auf gutes Zureden meines Vaters wagte ich es schlussendlich, mir dieses Wunderwerkzeug genauer anzuschauen und es überzeugte mich. Der Fakt, dass ich mir um das Layout keine Gedanken machen muss und es dennoch gut aussieht, ist mir sehr willkommen. Es hat alledings ein paar Tage gedauert, bis ich den Workflow verstand und auch jetzt beim wirklichen Schreibprozess habe ich immer wieder Fragen.§

6 Technische Hintergründe

Also gut, es geht los. Ich hoffe Sie sind bereit, eine Menge Technik zu lesen. Hier schildere ich jedes Modul und dessen Funktionsweise. Ich denke, ich erkläre Ihnen am besten zuerst die Arbeitsweise des *Konstrukts*. Zwischendurch lockere ich es mit einem kleinen Seitenfakt auf.

6.1 Ganz von vorn.

Das *Konstrukt* stellt für entweder vier oder acht Pflanzen möglichst optimale Wachstumsbedingungen her. Diese Ansprüche habe ich bereits geschildert, nun geht es darum, wie sie garantiert werden.

6.2 Bitte nicht reinschauen

Schauen Sie ihren eigenen Netzhäuten zuliebe *nicht* direkt in Growing-LEDs, für Ihre Augen ist es etwa so, als müssten Sie mit der Hand ein Panzergeschoss stoppen. Die LEDs habe zwar keine Laser-Klassifizierung, aber das ändert nichts am Leuchten, das Sie während den anschliessenden 5Minuten auf der Netzhaut spüren, nachdem Sie kurz neugierig waren.
LEDs sind cool. Sehr Effizient in Sachen Energie, was zur Folge hat, dass weitaus mehr Elektrizität in Licht und nicht in Wärme umgewandelt wird, als bei herkömmlichen Halogenlampen oder dergleichen.. In der Natur gewöhnen sich Pflanzen an das Abwechselnde Hell-Dunkel von Tag und Nacht. Es bewirkt eine stabilisierung des Pflanzenkreislaufs, denn, und das habe ich vorher nicht gewusst, Pflanzen können zu viel Licht bekommen. Es klingt Merkwürdig, aber es macht Sinn: Ein Blatt in all seiner Genialität, ist bei weitem nicht perfekt. Bei zuviel Sonneneinstrahlung geraten gewisse Arten an Teilchen innerhalb des Kreislauf ausser Kontroller und blockieren z.B. die Zufuhr von Kohlendioxid. Genau will und kann ich das selbst nicht erklären. Kurz gesagt: Wir müssen diese LEDs einfach mal abschalten können, sonst tue ich den Blättchen das Gleiche wie meinen Netzhäuten an. Die LEDs sind einfach Lampen in einem flachen Gehäuse, an dem ein Kabel angeschlossen ist. Dieses Kabel stecke ich in den Relay-Controller, dessen Relays den Stromfluss auf die jeweilge Steckdose kontrollieren. In den Weed-Foren habe ich von einem Belichtungszyklus von 18h zu 6h gehört, was ungefähr auch meinem Schlafzyklus entspricht. Meine Pflanzen gehen also mit mir ins Bett. Per Webinterface programmiere ich die *ontime* auf 18h * 60min * 60s, was insgesamt 64800 Sekunden sind. Ausgeschaltet sind sie demnach 21600 Sekunden. Vorerst benutze ich die LEDs immer abwechselnd, es muss hier ja nicht noch jemand/etwas die Seele verlieren.. Das Gute an meinem System ist jetzt schlicht der Fakt, dass ich die Belichtungszeit einfach ändern kann. Ich kann sie so einstellen, dass die Lichter aus sind, wenn ich ins Bett gehen will. Die Growbox ist zwar recht dicht, doch bei den Reissverschlüssen leuchtet es dennoch etwas durch, von den Lüftungslöchern mal ganz abgesehen. Wirklich heiss werden die LEDs auch nicht. Das Lichtspektrum ist wie schon in der Bauteilliste so eingestellt, dasses genau dem von den Pflanzen benötigten Wellenlängen entspricht. Es gibt eine ganz einfache Methode um herauszufinden, ob die Blätter das Licht absorbieren oder nicht. Wenn sie es *nicht* absorbieren, dann müssen sie es reflektieren. Wenn die Blätter also schwarz erscheinen, weil sie nichts reflektieren und alles absorbieren funktionieren die eingestellten Wellenlängen.

6.3 Das Ding atmet ja!

Wir Sauerstoff-Atmer existieren aus dem alleinigen Grund, dass Pflanzen CO_2-Ein- und O_2-Ausatmer sind. Was sie als O_2 *ausatmen* brauchen wir, damit unsere Mitochondrien aus Glucose

ATP, einen chemischen Energieträger machen können. Die Growbox hat ein Leervolumen von $0.612m^3$, das ist nicht viel und man sollte auch bedenken, dass da noch ein kompletter Zementeimer drin steht, welcher von einem Brett bedeckt wird, welches die Box glatt in zwei Teile teilt. Von den 170cm Höhe verliere ich also gute 35cm, was mich mit nichtmal einem halben Kubikmeter zurücklässt. Das CO_2 ist deswegen schnell aufgebraucht und es wird auch schneller heiss. Wenn das CO_2 aufgebraucht ist, betreiben die Pflanzen keine Photosnythese mehr. Diese Situationen gilt es demnach zu vermeiden. Der komplette Luftzyklus ist aufgrund dem sich ausdehenden Verhalten von Luft, aufwärts gerichtet. Unten kommt die kältere, CO_2-reiche Luft und oben wird die wärmere, CO_2-arme Luft rausgeblasen. In richtig grossen Anlagen wird nicht nur aus CO_2-Gründen gelüftet, sondern auch, weil es zu heiss werden kann. Bei mir ist das mit meinen 50W-LEDs aber komplett nicht der Fall. Für die Photosynthese essenzielle Proteine verlieren ihre Tertiärform, die Form die ihnen ihr Funktionieren ermöglicht, ab etwa 35°C. Ich gehe stark davon aus, niemals an diesen Wert zu gelangen. Auch wenn mein Zimmer, der Standort des *Konstrukt* im Sommer recht heiss werden kann, reden wir von maximal 30°C. Ich behaupte nicht, dass es unmöglich ist, die Schmerzgrenze von 35°C in meinem Zimmer zu erreichen, aber durch die eingebaute Thermostatenfunktion des Dämons werden wir ein *Meltdown* der kleinen, guten Proteinchen meiner Pflanzen hoffentlich nie erleben.

6.3.1 Die Zauberformel mag keine Luftfeuchte

Während der gesamten Konstruktionsphase und einem guten Teil der Schreibphase war ich im Glauben, schon bei niedrigen Temperaturen durch lüften, kühlen zu müssen. Jüngste Erkenntnisse widerlegen diesen Glauben, dennoch möchte ich Sie mit meinen Gedanken im Bezug auf den Relevanzunterschied zwischen Temperatur und relativer Luftfeuchte bekannt machen. Die Signifikanz der Luftfeuchte auf das Wachstum *meiner* Pflanzen in *meiner Konstruktumgebung* hat sich ebenfalls geändert, doch wir sind hier im Kapitel des Arbeitsprozesses, in dem ich Ihnen direkte Einblicke in meine Gedanken ermögliche. Lesen Sie das folgende mit dem Wissen im Hinterkopf, dass sich *auch* die Signifikanz der Temperatur verringert hat.

Ein komplett anderer Luftaspekt ist die relative Luftfeuchte, ich kürze das mit rel. LF, ab. Ich nehme an (!) dass es Pflanzen beim Stoffwechsel hindert, wenn die Luft zu feucht ist. Hier ein Erklärungsversuch: Innerhalb der Pflanze werden Bausteine und Wasser über den Stängel wie auf einem Fliessband transportiert. Das Wasser an sich ist schon ein essenzieller Faktor in jedem Stoffwechsel. Die ganze Überlegung macht mehr Sinn, wenn man weiss, dass der Sog innerhalb der Pflanze nur dadurch entsteht, weil Wasser an der Unterseite der Blätter kondensiert und so *neues* Wasser *nachgesogen* wird, das hat mit sogennanten Kapillarkräften zu tun, welche es dem Wasser erleichtern, durch sehr dünne Kanäle zu fliessen. Wenn die rel. LF aber zu hoch ist, kondensiert das Wasser an der Blattunterseite nicht mehr, weil es, vereinfacht angeschaut, keinen Platz mehr in der Luft hat, wo sich noch mehr Wassertröpchen aufhalten könnten. Der Kapillarfluss kommt also ins Stocken und die Pflanze wird am Wachstum gehindert. Ich bin mir bei dieser Überlegung nicht sicher, was mich dazu bewegte, die rel. LF einfach aus der Gleichung zu entfernen. Ich mag es simpel. Es ist von einer ganz bestimmten Gleichung die Rede, nämlich der Vorgang, bei dem der Dämon, da ist er wieder, berechnet, wieviel Saft (in Form von PWM) er denn Lüftern geben muss. Schlussendlich, auch wenn ich die nötige Technologie besass und zur Hand hatte, habe ich mich einfach dazu entschieden, diesen kompletten Aspekt zu streichen. Vorallem auch deswegen, weil ich nach einer Beispielsberechnung gemerkt habe, dass die LF einen *viel* zu geringen Einfluss auf das PWM-Signal hat. Es ist wohl am besten, wenn ich das kurz erläutere und da wir hier im technischen Teil sind, werden Sie sich alle Mühe machen, es auch zu verstehen. Vorerst möchte ich anmerken, dass alle von mir genannten Werte

meine eigenen Annahmen sind und womöglich komplett daneben sind, ich habe noch nie ein Belüftungssystem gebaut. Los geht's
Die Lüfter laufen konstant auf 40% PWM, um den Pflanzen genügend CO_2-Zirkulation zu geben. Alles was daraufgerechnet werden, also die restlichen 60% hängen von der Temperatur ab. Mit der Temperatur mache ich es so, dass ich einen Optimalwert halten will, ich ging in meinen Berrechnungen meistens von etwa 24°C aus, wie gesagt, dieses Milieu ist mir neu. In Relation zu diesen 24°C gehe ich von einer Temperatur aus, ab welcher ich auf voller Stufe lüften will, sagen wir 27°C, es bleiben uns also 3°, welche wir auf unsere 60% verteilen: Einfache Proportionalität, wir gehen also davon aus, *pro jedem Grad über dem Optimalwert 20% PWM-Saft auf die schon bestehenden 40% zu addieren.* So einfach habe ich mir das Leben gemacht. Eine ähnliche Überlegung bin ich bei der rel. LF eingegangen, allerdings sind meine Kenntnisse da noch weniger und der effektive Wert, um welchen ich mehr lüften muss, war ca. ein Zehntel dessen der Temperatur-Überlegung. Ich hatte anfangs beide in der Berechnung drin, da aber die Temperatur einen grösseren Impact hat, musste ich auch diese Einflussdifferenz berücksichtigen, was den Dämonen nurnoch verkomplizierte. Und lassen Sie mich Ihnen versichern, dass dieses Dämonenwerk auch ohne Lufteuche-Berücksichtigung schon kompliziert genug ist, jedenfalls für mich. Ein realistischer Daten-Ablauf sieht folgendermassen aus: Der Dämon läuft in einer Endlosschlaufe und fragt den an den Raspi angeschlossenen DHT22-Temperatur & rel. LF-Sensor nach der Temperatur. Diese zusammen mit den Werten der Standart-PWM-Frequenz, dem Wunschwert und dem angenommenen Full-PWM-Wert (Ab da wo die Lüfter voll durchschalten) verrechnet er zum sogennanten *Auto-PWM-Wert.* Die automatische Berechnung dieses Werts lässt sich per Web-Interface deaktivieren. Dieser Auto-PWM-Wert wird anschliessend vom Dämonen an den PWM-Controller weitergegeben, über eine serielle Schnittstelle, alias USB-Kabel, welche ich beim Start des Dämons iniziiere und dann offen behalte, bis der Dämon stirbt - was natürlich *nie* passieren wird, versteht sich.
Wer aufmerksam gelesen hat, bemerkt vielleicht, dass ich bisher nur von zwei Lüftern schrieb, jedoch erwähnte ich in der Bauteilliste *vier*, was also machen die zwei anderen Lüfter? Von einem kann ich es Ihnen schon verraten, beim anderen müssen Sie sich noch etwas gedulden. Der dritte Lüfter hat eine ganz einfache Aufgabe, er innerhalb der Growbox für Turbulenzen sorgen und die Pflanzen direkt beblasen. Ich habe diesen Tipp ganz tief aus einem dieser 420-Portalen, anscheindend werden die Pflanzen damit stabiler und gesünder, was an sich Sinn ergibt. Ein Mensch ist auch gesünder wenn er Widerstand leisten muss, zum Beispiel beim Sport treiben. Wenn einem Organismus etwas abverlangt wird, entwickeln sich daraus Reserven, um dieser Situation nicht mehr ausgeliefert zu sein. Um also nicht einfach umgewindet zu werden, baut die Pflanze stabilere Wände.

6.4 Vom Gravitropismus und der Zukunft

An dieser Stelle passt es wohl am besten, um Ihnen vom *Gravitropismus* einer Pflanze zu berichten. Dieses Wort beschreibt die Reaktion von Pflanzen auf unterschiedliche Richtungen in denen die Gravitation auf sie einwirkt. Haben Sie schonmal einen schrägen, oder sogar ganz umgewehten Baum gesehen, bei dem die Äste dennoch gen Himmel wachsen? Ich mache es Ihnen noch einfacher: Wenn immer eine junger Baum an einer sehr steilen Stelle wächst und sich, einfach aus Platzgründen, senkrecht zum Untergrund emporhebt, was aber *nicht* senkrecht zum Horizont ist, dauert es nicht lange und er wächst wieder gerade in den Himmel. Dieses Ausrichten trägt den Namen *Gravitropismus* und es funktioniert, indem die Pflanze an der Stelle mehr Zellen produziert, von welcher sie weg möcht, was für uns als *unten* bekannt ist. *Unten* an einer Pflanzen bildet sich mehr Masse, was den gesamten Stamm/Stiel/Ast in die

andere Richtung drückt. An der Aussenseite einer Kurve hat es mehr Material als an der Innenseite, deshalb ist die Innenseite einer Kurve auch der kürzeste Weg. Stellen wir uns kurz die Frage, wer von diesem *Gravitropismus* Gebrauch machen könnte. Es sind neue, vertikale Farmen, mit dem Ziel, möglichst viel Nahrung auf möglichst kleinem Raum zu produzieren. Diese neuen Anlagen bringen ihre Pflanzen in unterschiedliche Ausrichtungen, wodurch sich die Einwirkung der Gravitation ändert und somit der Gravitropismus. Im Ende versuchen die Pflanzen, nebst dem normalen Wachstum auch noch nach oben zu wachsen, was zu mehr Pflanzensubstanz führt. Genial, nicht? Gerade auf einem Planeten wie der Erde, mit ihrer immer grösser werdenden *Wie ernähren wir alle 11 Milliarden Erdlinge im Jahre 2100?*-Frage, sind Erkenntnisse wie diese sehr erwünscht.

6.5 Wurzelfütterung

6.5.1 Nährnebelpegel

Wenn die Wurzeln noch jung, also noch *kurz* sind, muss ja rein theorethis der Pegel an Nährnebel höher sein, damit er sie erreicht. Aber das hätte zur Folge, dass ich die Relays, auch wenn es super einfach ist, umprogrammieren muss. Hier kommt der vierte Lüfter ins Spiel. Eher gesagt ist es en Lüfterchen, auf dem QUERSCHNITT ist er ebenfalls zu finden und dort erkennt man auch schon seine Funktion. Er schafft wie der dritte Lüfter Turbulenzen, allerdings im Nährnebel, um ihn damit weiter nach oben steigen zu lassen. Ich habe dieses Phänomen schon das erste Mal, als ich den Fogger eingestellt habe, beobachtet. Wenn man ganz heiss darauf ist, ein neues Spielzeug auszuprobieren, wartet man nicht lange. Meine Mutter wartete darauf, dass ich zum Essen komme, doch der frisch mit der Post angekommene Fogger war da. Einmal eingesteckt und in eine Wasserschüssel gesetzt schaltete ich ihn ein. Es sieht wie ein kleiner Vulkan aus, wenn das Wasser so hoch spritzt. Die ganz kleinen Tröpfchen, die mit einem Zweihundertstel Millimeter Durchmesser machen sich als Nebel bemerkbar und der Rest spritzt als ganz herkömmliches Spritzwasser hoch. Nach ein paar Sekunden Laufzeit hat sich eine ordentliche Menge Nebel angesammelt und jeder Mensch, der auch mal ein Kind war, spürt das unbekämpfbare Verlangen a) hineinzufassen und b) hineinzupusten. Gesagt, getan, die ganze Küche sieht leicht milchig aus. Mit künstlichem Nebel herumzuspielen ist irre witzig. Die Erkenntnis aus diesen Spielerein lag ganz einfach darin, dass man den Nebel recht einfach anheben kann, indem man ganz leicht in die Mitte bläst, auf der Darstellung ist das genauso illustriert. Die Bretter, durch welche ich die Netzbecher stecke, in denen die Steinwolleblöckchen sind, in denen widerrum die Pflanzen halt finden, haben entweder fünf oder neun Löcher. Ich zweckentfremde bei beiden das mittlere Loch und setze einfach das Lüfterchen drauf. In einem meiner *Forschungsaufbauten*, ich das vorhin kurz erwähnt, fand ich heraus, dass 35% PWM für das Lüfterchen genau im perfekt Nähnebelpegel resultiert. Das einzige Problem ist jedoch folgendes: Beim PWM-Controller-Arduino brachte ich es nicht hin, für vier Pins, weil es ja vier Lüfter hat, jeweils einen eignen *TimerOne*-PWM-Timer zu stellen. In meinem Fall haben nur Pin 9 und 10 miteinander funktioniert. Das alleine für vier Lüfter reicht nicht, ich brauchte noch mindestens einen mehr. Ich habe also die eingebaute PWM-Funktion, mit 1kHz hinzugenommen. Und zwar auf Pin 3, von dort aus wird zurzeit das kleine Lüfterchen gesteuert. Die geringe Frequenz führt unter anderem dazu, dass es lauter summt und quietscht als bei den vierzigfachen Frequenzen der Pins 9 und 10. Auf Pin 10 ist der Turbulenzlüfter angeschalten und auf Pin 9 *zwei* Lüfter, welche die Zentrallüftung übernehmen, einer oben in der Growbox und einer unten. Das Quietschen beim Lüfterchen kommt höchstwahrscheinlich vom Stocken, also dem Motor, der noch recht gut zwischen An und Aus unterscheiden kann. Das führt zu Vibrationen, welche sich gut im Holz ausbreiten. Das will ich jedoch nicht, denn mein Schlaf ist

mir heilig. Was ich also tat, war es, die Lüfterchenvibrationen einfach mit Watte und *Ducttape* abzufangen, dadurch wurde es schon viel besser.

6.5.2 Klick, klick

Die LEDs mit ihrem *18h zu 6h-Rhythmus* könnte ich noch fast von Hand abschalten, wenn ich ins Bette gehe und wenn ich wieder aufstehe. Doch den Fogger, den kann ich nicht alle 20s einschalten und ihn nach 10 wieder ausmachen. Das wär mir zu blöde und das würde das Ziel dieses Projekts verfehlen, das Giessen muss ja eben automatisch geschehen. Den Relay-Controller habe ich genau für das konstruiert. Einstecken, online den Rhythmus eingeben und einfach sein lassen. *Set-and-Forget* ist hier das Motto. Das funktioniert sehr gut. Doch Sie fragen sich vielleicht *wie*. Nachdem ich den complettenk Relay-Timer-Code von Arduino/C++ auf Python übersetzt hatte, habe ich ihn verworfen und einen neuen Ansatz verfolgt. Es ist vielleicht gut zu wissen, dass es ganz grob zwei Arten gibt, mit welchen man an dieses Problem herangehen kann. Die erste liegt darin, sich für jedes der vier Relays einen Timer zu stellen, der individuell abläuft. Ist er abgelaufen stellt man sich einen neuen Timer und sendet ein Flip-Signal an den Relay-Controller. Dieser weiss was er damit zu tun hat und schaltet das gewünschte Relay um. Die zwei verwendeten Zeiten sind die *offtime* und die *ontime*, welche man in Sekunden definiert. Zusätzlich hat gibt es noch den *starttimer*, welcher einfach, wenn er nicht 0 beträgt, am Anfang abgewartet wird. Somit kann ich sichergehen, dass der Lichtzyklus erst am Morgen anfängt und nicht zum Zeitpunkt, an dem man es im Interface eingibt. Diese Variante birgt den Nachteil dass sie besonders schlecht mit Änderungen an den Werten klarkommt, wenn der Timer erst mal läuft. Man könnte den laufenden Timer natürlich einfach abwarten lassen, doch bei 18h macht das keinen Sinn. Der andere Ansatz ist das 1-Sekunden-Verfahren. Es ist eine bei weitem nicht perfekte Lösung und ein richtiger Informatiker mag sie auf als *dirty* bezeichnen, doch sie funktioniert und das ist die Sache, die unter Zeitstress zählt. Anstatt volle 6h oder 20s oder 10s oder 18h abzuwarten, wartet man einfach immer nur eine Sekunde, danach zieht man eine Sekunde vom Timer ab. Wenn der Timer auf 0 steht, wechselt man einfach, sendet also wie vorher beschrieben das Flip-Signal und macht stellt den Timer, auf die *ontime*, wenn es vorher die *offtime* und umgekehrt wenn nicht. Hier ist der Code der `relays.py`-Datei

```
from daemon import relays

def update_rl_arr(ID):
timer = 0
ontime = 1
offtime = 2
state = 3
#expecting a list as input with the following format: [timer, ontime, offtime, state]
relays[str(ID)][timer] = relays[str(ID)][timer] - 1
if relays[str(ID)][timer] < 1:
relays[str(ID)][timer] = relays[str(ID)][ontime] if relays[str(ID)][state] == "off" else
# set timer to ontime if the old state was "off"
relays[str(ID)][state] = "off" if relays[str(ID)][3] == "on" else "on"  #set state to of
send_flip_token(ID)

def send_flip_token(serial_object,ID):
serial_object.write(ID.encode())
```

Der Import ganz oben macht das `relays`-Objekt für den Code in dieser Datei zugänglich. Dieses Objekt ist ein sogenanntes Dictionary. Diese Datenstruktur funktioniert mit dem *Key - Content*-Prinzip. für einen bestimmten `key` habe ich gewissen `content`, in diesem Fall ist es eine Liste mit Werte, drei Zahlenwerte und ein Wort. Die Funktion `update_rl_arr`, was soviel wie *update-relay-array* heisst, wird nach jeder Sekunde einmal aufgerufen und subtrahiert 1 vom `timer`. Der `timer` steht an der ersten Stelle der Liste/des Arrays. Diese List stellt den Content dar, welche man mit dem `key`, in diesem Fall `ID`, zugänglich macht. `ontime` steht demnach an zweiter Stelle (man beachte, dass man bei Arrays/Listen bei 0 zu zählen beginnt), `offtime` and dritter und `state` an vierter. Das Format sehen sie nochmal im Kommentar auf der achten Zeile. Was ich ihnen vorher mit Worten beschrieb, findet sich hier als Python-Code. Ich benutzte mehrere Python-Finessen um den Code zu verkürzen, was es womöglich etwas komplexer macht, doch versuchen sie ihn wie einen Englischtext zu lesen. Die Funktion `send_flip_token(serial_object, ID)` ganz am Schluss macht das, was der Name schon aussagt. Ich übergebe ihr via `serial_object` die serielle Schnittstelle und `ID` das zu flippende Relay. Den Rest übernimmt der Relay-Controller in der kleinen, grauen Box.

6.5.3 Die kleine, graue Box mit Steckdosen drin

Die kleine, graue Box mit Steckdosen drin heisst auch Relay-Controller, jetzt allerdings ist von der Hardware die Rede, die Software haben gerade besprochen. Sie besteht aus einem Kunstoffquader, der auf den Seiten Gummi-Deckel hat, diese Deckel lassen sich entfernen, um Kabeln Zugang zu bieten. Ich glaube, dass es normalerweise ebenfalls dafür benutzt wird, Stromkreise zu verbinden. Eine Platine reinzuschrauben ist wahrscheinlich aber nicht vorgesehen. Das ist mir egal, denn es hat die perfekt Grösse und und lässt sich sehr gut abdichten. Der abnehmbare Deckel ist mit einem Abdichtungsring versehen. Deshalb war es etwas schwer, zwei riesige Löcher reinzuboren, durch welche ich die beiden Steckdosen stecken konnte. Sie haben gerade noch so Platz, viel Freiraum bleibt nicht. Eine solche Steckdose hat drei *Steckdosen*, also insgesamt hat es 9 Pins, war heisst, dass man drei schweizer Stecker damit verbinden kann. Das habe ich zweimal, was es mir erlaubt, sechs Geräte mit Strom zu versorgen. Das etwas unlogische an diesen Steckdosen ist der Fakt, dass nicht alle Steckerauffassungen (da wo *ein* Stecker reingeht) einen eigenen Stromkreis besitzen, sondern dass nur eine Auffassung einen eigenen hat und sie die anderen beiden einen Stromkreis teilen. Das bedeutet, dass ich nur vier Auffassungen individuell ansteuern kann, von der Anzahl Relays passt das aber perfekt. Um den Stromfluss einer Auffassung zu unterbrechen, schalte ich einfach das dazugehörige Relay aus, welches Teil eben dessen Stromkreises ist. Diese ganze Verkabelung ist recht steif, aber ich habe genügend lange Kabel benutzt, damit ich den Deckel immer noch abnehmen kann. Die Platine, welche die Relays, den Arduino und das kleine Netzteil beherbergt ist am Boden angeschraubt, bombenfest. Es ist eine dieser *einmal machen, dann aber richtig*-Lösungen, welche mir mein Vater immer eingetrichtert hat. Sie funktioniert. Daran angeschaltet sind beide LEDs und der Fogger, alles liess sich per Webinterface vorprogrammieren.

6.6 Der Dämon

Den Beschwörungsprozess des Dämons habe ich bereits geschildert. Die Geisterbrücke mit `rsync`, Sie erinnern sich? Jetzt geht es darum, *was* ich mit Sublime genau heraufbeschwört habe. Die Aufgabe des Dämons ist es, alle Hintergrundprozesse abzuarbeiten, dazu gehören: Die Entgegennahme von neuen Werten, welche ihm der Flask-Webserver mitteilt, die Berechnung der Lüfterstärke basierend auf der Temperatur, die Mitteilung dieser PWM-Werte an den PWM-Controller und zu guter letzt der komplette Relay-Algorithmus.

Wenn man den Dämonen das erste Mal startet, iniziiert er die Kommunikation mit beiden seriellen Schnittstellen, also dem PWM- und dem Relay-Controller. Ein Problem, auf welches ich schon recht früh aufmerksam wurde, war die Besorgnis, wie ich dem Dämonen mitteile, welcher Port zu welchem Controller gehört. Die USB-Schnittstellen erscheinen unter einem Linux-System mit den Namen `ttyACM0` und `ttyACM1` im `/dev`-Verzeichnis, dort werden alle echten und virtuellen Geräte aufgelistet. Der Raspi hat einen gewissen Prioritätscheck, welchen er bei jedem Start durchführt, darauf basierende verteilt er den USB-Schnittstellen ihre Namen, entweder `ttyACM0` oder `ttyACM1`. Da ich diesen Prioritätscheck erstens nicht verstehe und ihm zweitens auch nicht voll vertraue, habe ich meinen eigenen kleinen Check geschrieben, womit der Dämon, rausfinden kann, welcher Controller auf welchem Port zu erreichen ist. Um es ganz einfach auszudrücken, macht der Dämon beide Ports auf, schickt bei beiden das Signal `99` und weiss anhand der Antworten, wo welcher Controller angeschlossen ist. Die Controller sind natürlich darauf programmiert, bei sich einem eingehenden `99`-Signal zu identifizieren. Für diesen kleinen Algorithmus habe ich länger gebraucht, als mir lieb ist, allderdings habe ich eine Menge gelernt, vorallem in Sachen wie *Serial Communication*. In meinen Logs findet man nicht nur einmal die Aussage *serial com is a bitch*, denn es fühlte sich mehrmals danach an. Auf das Endprodukt dieser kleinen Spielerei bin ich allerdings recht stolz, sie ist wie jede andere Datei auf meinem GitHub-Repo zu finden, auf welches ich am Schluss noch zu sprechen komme.
Wenn der Dämon also weiss, wo er es mit welchem Controller zu tun hat, braucht er erstmal die Startdaten, welche sich in einer Datei im gleichen Verzeichnis befinden, diese Datei nennt sich `local_properties`, es gibt auch noch eine Datei namens `remote_properties`. Der ganze Clou hinter der Kommunikation zwischen dem Dämonen und dem Webserver liegt in diesen beiden Dateien. Die beiden Dateien sollen immer möglichst auf dem gleichen Stand sein. Mit `remote_properties` speichere ich jedesmal eine Änderung der Werte, wenn ich etwas im Interface verändere, zum Beispiel, wenn ich die Standart-PWM-Stärke für die Lüftung von 40 auf 50% anhebe. Der Webserver aktualisiert den Wert einfach in `remote_properties` mit einer recht genialen Funktion, welche sich `update_prop_or_relay` nennt.

```
 #key can be just a key for properties or the id for relays
def update_prop_or_relay(key, newvalue, path=REMOTE_PROPERTIES):
print(f"updating {key} in {path}")
properties = {}
relays = {}
with open(path, "r") as f:
for line in f.readlines():
values = line.split()
if values[0] == "relay":
if values[1] == key: #check if i can already overwrite the old stuff
relays[key] = newvalue #as we expect to change a relay, "newvalue" should be a list.
else:
relays[values[1]] = [values[2], values[3], values[4], values[5]] # if it didn't match, w
elif values[0] == key:
properties[key] = newvalue #and if it wasn't a relay, just save it as a property. Here i
else:
properties[values[0]] = values[1]
save_props_and_relays(relays, properties, path) #save them all
```

Alles was sie macht, ist, dass sie die bestehende `remote_properties` als funktionseigene Dictionaries, `relay` und `properties` speichert und den zu verändernden Wert direkt auf den neuen ak-

tualisiert. Anschliessend speichert sie alles wieder mit `save_props_and_relays()`. Wenn in einem Dateiensystem eine Datei verändert wird, ändert sich auch ihr `last modified timestamp`, also der Zeitpunkt, an dem man sie das letze mal änderte. Der Dämon merkt das und weiss dann sofort, dass sich etwas verändert haben muss. Was macht er also? Er synchronisiert `remote_properties` mit `local_properties` und hat somit eine Kopie von `remote_properties` zur Hand, aus welcher er die Werte ausliest, um mit ihnen weiterzumachen. Alle werte in diesen beiden Dateien sind diese:

```
relay 1 0 10 20 off
relay 2 0 64800 21600  off
relay 3 0 64800 21600  off
relay 4 0 64800 21600  off
opt_temp 24
max_delta_t 3
std_pwm 40
vent_mode auto
pwm_vent 40
pwm_fog 35
pwm_turb 50
```

Relay 1 und 2 sollten selbsterklärend sein. 3 und 4 werden nicht gebraucht, deshalb ist es egal, welche Werte sie besitzen, aus ästhethischen Gründen gebe ich einfach die Werte der Lampe mit. Wenn sie auf 0 stehen würden, würden sie jeden Zyklus umschalten, was einen Höllenlärm verursacht. Wenn sie aber ebenfalls im 18h-6h-Zyklus der Lampe laufen, ist mir das recht. `opt_temp` ist die Optimaltemperatur, `max_delta_t` die Temperaturdifferenz zwischen der 100%-PWM-Grenze und der Optimaltempereatur, `std_pwm` die Standart-PWM-Stärke für die Lüftung (damit der $C0_2$-Austausch gewährleistet ist), `vent_mode` ist entweder `auto` oder `manual`, je nachdem ob man die Lüftungstärke automatisch anpassen will, `pwm_vent` ist die Lüftungsstärke, dieser Wert wird automatisch berechnet, wenn `vent_mode` gleich `auto` ist, `pwm_fog` ist die Lüfterchenstärke für die Nährnebelturbulenzen und `pwm_turb` schliesslich die Stärke für den Turbulenzventilator.

Mit diesen Werten arbeitet der Dämon jede Sekunde, er schaut ob sich was verändert hat, wenn ja, synchronisiert er sie, liest sie ein und macht dann weiter. Diese Aktion läuft in einer Endlosschlaufe und hört erst auf, wenn ich den Prozess abschiesse.

6.7 Den Code selbst lesen

Wenn Sie den Code selbst lesen oder weiterentwickeln wollen lade ich Sie natürlich herzlichst dazu ein. Sie finden ihn auf https://github.com/schnippo/ssgb

6.8 Ungenauigkeiten

Welche Annahmen in meiner Maturarbeiten sind ungenau? Ich habe dieses Thema schon kurz bei der Lüftungssteuerung angeschnitten. Ich möchte hier aber dennoch noch kurz auflisten, welchen Werten ich nicht vertraue, respektive in welchen Bereichen ich aus dem Bauchgefühl handelte. Dies ist zwar eine gestalterische Arbeit, jedoch setzt sie sich mit einem wissenschaftlichem Thema, der Lebenserhaltung von Pflanzen, auseinander. Aus diesem Grund möchte ich dem wissenschaftlichen Teil mindestens dadurch Genüge leisten, indem ich Bereiche der Ungenauigkeit aufzähle.

Als allererstes könnte ich mich beim CO_2-Verbrauch der Pflanzen komplett irren, ich habe mir schlicht die Mühe nicht gemacht, diesen Aspekt zu behandeln, obwohl ich dies im Nachhinein bereue, denn es hätte gut in die Arbeit gepasst und mir mehr Sicherheit verschafft. Dass ich es bis jetzt nicht gemacht habe heisst allerdings überhaupt nicht, dass es *nie* geschehen wird. Die Abgabe dieses schriftlichen Einblicks in mein, ich würde es nicht *Lebens*projekt nennen, aber es geht in diese Richtung, in mein Projekt ist nur ein kleiner Punkt auf dem Zeitstrahl. Wenn Sie sich also wundern, wie ich an diese Fragestellung herangehe: Ich verspreche Ihnen, das README.md, also die Startseite der GitHub-Seite so zu aktualisieren, dass es ersichtlich ist, wenn ich den CO_2-Verbrauch berücksichtige.
Weitere Annahmewerte waren alle Temperatur- und Feuchtigkeitswerte. Proteine denataurieren ab etwa 35° Grad, bis unter diesen Wert funktioniert die Photosythese immer schneller. Rein theorethisch liegt es also in meinem Interesse, den Pflanzen genügedn CO_2 zu liefern, während ich die Temperatur möglichst hoch halte, aber unter 35°. Schon während ich das hier schreibe, kommt mir die Idee für eine neue Art Thermostat. Er wird wohl aber noch etwas warten müssen. Einer rein wissenschaftlichen Arbeit täte ich mit solch gravierenden Ungenauigkeiten kaum Genügen. Aber das ist eine gestalterische Arbeit und es geht um die Erfolge wie Fehler des Konstruktionsprozesses.

7 Perspektiven

In diesem Kapitel mache ich Sie mit harten Einsichten, erlernten Fähigkeiten und neuen Perspektiven bekannt.

Einhergehend mit den Arbeitsprozessen, bekam ich auch neue, mir zuvor völlig undenkbare Einsichten und Perspektiven

7.1 Technisches Upgrade

Es ist komplett offensichtlich, dass ich eine ganze Menge über Technik gelernt habe. Im Kapitel *Arbeitsprozess* bin ich darauf eingegangen was ich wie gemacht habe. Jetzt zeige ich Ihnen auch noch, was ich daraus gelernt habe. Ich kann Ihnen natürlich nicht einfach sagen dass ich gelernt habe, dass z.B. Dioden Strom nur in eine Richtung durchlassen, denn das ist selbst für mich etwas langweilig. Ich erwähne deshalb nur den groben Teilbereich, mache ein, zwei *spannende* Beispiele und gehe dann zum Nächsten.

7.1.1 Oszilloskop und Elektronik

Ich habe es bereits gelobt. Eine Wundermaschine mit der ich mein Verständnis fürs Verhalten von elektrischem Strom verbessern konnte. Um in meinem Beispiel ein PWM-Signal zu visualisieren, muss man den Strom durch *den Fühler* leiten, so nenne ich es. Der Fühler hat zwei Klemmen, eine kommt an die Masse-/Minusleitung und der andere an entweder die Basis oder den Emittor des Transistors. Den Transistor erkläre ich auch nur kurz: Er hat drei Pins, Basis, Emittor und Kollektor. Vereinfacht geht der Strom beim Kollektor rein, und verlässt ihn mit einer um, einen Transistor-spezifischen, Faktor verstärkten Spannung beim Emittor, aber *nur* wenn bei der Basis eine Minimalspannung von 0.7V angelegt wird. Es funktioniert also im Grunde wie ein Relay, einfach mit Spannungsverstärkung. Das Oszilloskop habe ich oft gebraucht. Es hat vier *Fühler*, wovon aber maximal zwei nötig waren. Die Darstellung des Signals, also Frequenz, Amplitude und Grundwert lassen sich alle entweder manuell oder automatisch einstellen. Man kann auch gewisse Triggerpoints setzen, welche das Oszi dazu bringen,

nur Werte anzuzeigen, welche diesen Trigger erreichen. Nach ein paar Abenden hatte ich den Dreh recht gut draussen.
Was ich auch zu schätzen gelernt habe sind Schaltpläne. Ohne Schaltplan hat sich mein Vater geweigert, eine Schaltung auch nur anzusehen. Ich kann es ihm nicht verübeln, denn es macht einfach keinen Sinn, planlos irgendwelche Sachen zu messen. Man baut ohne Bauplan ja auch kein Haus. Einen Schaltplan zu zeichnen ist guter Spass und schafft Klarheit. Die Software *Fritzing* erlaubt es einem sogar, echte Module wie einen Arduino für seinen Plan zu benutzen. Ich habe das ein- oder zweimal gemacht, es sah schön aus aber mit einem Bleistift und etwas Kritzelpapier ist man schneller und ausserdem auch noch authentischer.
Meine Lötskills habe sich ebenfalls um einiges verbessert, das wurde bereits klar. Viel kann ich dazu nicht sagen, man sollte darauf achten, dass der Lötkolben immer etwas Zinn dran hat, aber nicht zu viel. Ausserdem verzinnt ein richtiger Elektroniker zwei Kabel immer, bevor er sie zusammenlötet. Das Leben kann sehr einfach sein, wenn man nur weiss wie richtig gelötet wird - das habe ich gelernt.

7.2 Signifikanz

Das richtige Abwägen, ob eine bestimmte Funktion vom *Konstrukt* ausgeführt werden können soll, ist eine Angelegenheit für sich. *Decisionmaking* ist anspruchsvoll und entscheidet darüber, wieviel man arbeiten muss. Die durchführbarkeit eines Teilprojekts spielt wohl die grösste Rolle. Hier sind ein paar Beispiele

7.2.1 Bewässerung mit Düsen

Davon war die Rede im Kapitel *An die Arbeit*, es war zu diesem Zeitpunkt die mir einzig effizient erscheinende Option. Düsenbewässerung zählt sich zur Aerponik, welche den Ruf einer sehr effizienten und wassersparenden Bewässerungsmethode hat. Effizienz und der ökologische Fussabdruck sind beides Sachen, welche bei der Konstruktion berücksichtigen wollte. Von Fogponics war ich zuerst nicht angetan und zwar wegen dem nächsten aufgeführten Thema. Ich hatte mich falsch informiert und berechnete die Leistung eines Foggers auf mehr als 400W, was, wie Sie gleich erfahren werden, nicht tragbar gewesen wäre.

7.3 Solar overkill

Das Projekt heisst auf dem Maturarbeitsvertrag *Automatische Vertical Gardening Anlage*. Später kam mir noch die Idee, den kompletten Stromverbrauch mit Solapanels abzudecken. Diese Idee ist unglaublich cool und es tut mir in der Seele weh, dass ich es nicht geschafft habe. In diesem Bereich habe ich versagt. Es war wahrscheinlich 70% der ganzen Arbeitsphase geplant und ich musste mir irgendwann eingestehen, dass der Aufwand zu gross wäre. Geplant hatte ich eine eigene Solaranlage auf meinem Dach. Ich habe die Sonneinstrahlung berücksichtigt, mich im Web schlau gemacht, Preise und Produkte rausgesucht und - wie vorhin schon angedeutet - meinen gesamten Energieverbrauch zu berechnen versucht. Ich wollte also den Stromverbrauch der Lampen, des Foggers, der Lüfter und der Steuerungstechnik berechnen, damit ich wusste, wieviel Panels ich bestellen müsste und wieviel Stromspeicherkapazität es bedurfte. Dazu kamen auch Überlegungen wie schlechtes Wetter, Einstrahlungswinkel und daraus resultierend der *Energiesparmodus*: Wenn die Stromreserven in den Batterien zur neige gingten, wollte ich die Leistung der LEDs runterdrosseln, damit die wirklich wichtigen Lebenserhaltungssysteme wie Bewässerung und CO_2-Versorgung noch genung Strom hatten, um die Pflanzen bis zum nächsten Auftanken, in Form besseren Wetters oder Sonnenaufgang, durchzubringen. Der

Solaraspekt meines Projekts fand seine Präsenz sogar im Titel: SSGB - Smart *solar* GrowBox. So heisst das Projekt auch auf GitHub. Dass ich dem nicht nachgekommen bin und dass mein GitHub-Titel lügt wurmt mich erhlich gesagt. Rein vom Arbeitsaufwand wäre es *vielleicht* schaffbar gewesen, aber meine Erfahrung ist in zu vielen Bereichen, und dieses Projekt schneidet *einige* Bereiche an, einfach zu wenig gewesen. Angenommen ich hätte schonmal mit Pulsweitenmodulation gearbeitet, einen Pythondämonen beschwört oder einen einen Flask-Webserver aufgesetzt, dann hätte es vielleicht besser ausgesehen. Aber auch dann wären immer noch viele Fragen offen, einige habe ich oben aufgeführt. Wie genau installiere ich eine Solaranlage auf meinem Dach? Ich hätte niemals eine Firma dafür arrangiert, denn das wäre gegen meinen MA-Kodex, der besagt, dass ich alles selber mache. Die Befestigung der Panels, sodass sie sich auch bei einem Sturm wie *Sabine* nicht aus dem Staub machen hätte ich mit meinen Kletterkünsten wahrscheinlich noch hinbekommen, doch wie sollte ich die Hauptstromleitung *in* mein Zimmer bekommen? Das Fenster für immer offen zu lassen fand ich keine gute Idee und ein Loch in mein Dach zu bohren widerspricht der Natur eines Dachs. Dächer sind ja da, um dicht zu sein. Ich hätte es also abdichten müssen - wie? Meine Situation ist klar, ich habe viel geplant, es hat nicht in die Form der Maturarbeit gepasst, aber das hindert mich nicht, es dennoch einfach später zu versuchen. Vielleicht findet sich auf meinem Git-Repo bald ein Ordner mit dem Namen `solar`.

7.3.1 Python und Flask

Meine Begeisterung für Python war schon vorher da. Der simple Syntax und die unglaubliche Vielfalt dieser Sprache sind ansteckend. Als ich dann nach meiner C++/Arduino-Oddyssey an Land stieg und nach Hause in meine Python-Domäne ging, um den Dämonen zu beschwören, fühlte ich mich um einges wohler. Den Syntax von Python beherschte ich schon, hatte ja schon ein paar kleinere Projekte damit verfolgt, aber was mich dann so richtig überzeugte, waren die Oneliner - *Ternary Expressions*. Sie sind die einzeilige Alternative zu einem vierzeiligen `If - Else` wenn es darum geht, Variabeln zuzuordnen. Der Flow der damit in eine Funktion kommt ist schön anzuschauen. Es wird auf der einen Seite komplexer, da mehr Code in weniger Zeilen steht, aber gleichzeitig räumt es auch auf - Wenn man schon drei `If - Else`-Statements tief in der Funktion noch ein viertes beschwört, nur um einer Variabel einen neuen Wert zu verpassen, dann lässt man sich gerne auf den Einzeiler in Form einer *Ternary Expression* ein.
Ein weiterer Stern am Pythonhimmel ist selbstverständlich Flask. Die traditionelle Webseite wird durch `apache2` oder `nginx` zugänglich gemacht. Das Backend, in meinem Fall also der Teil, welcher dem Dämonen das Essen serviert, wird traditionell von der *PHP-Engine* dargestellt. Da ich aber noch keine einzige Zeile PHP-Code geschrieben habe, erscheint mir eine Alternative wie Flask wahrhaftig wie ein Stern am Himmelszelt.

7.4 Mögliche Weltverbesserung

asdf

8 Fazit

9 Danke sagen

Vielen Dank Papa für deine unendlich ansteckende Begeisterung für die Materie der Technik.

10 SWAP

Glossar: Oszilloskop CPU Timer-Register Relay Transistor

```
for i in range(19):
```

Alle haben es in dieser Schule gelernt, einige schon im U1; Pflanzen brauchen Wasser, Luft, Licht und Nährstoffe für richtiges Wachstum. Die ersten drei sind notwendig für die Photosyntese. Ausserdem ist auch ein ädequates Klima von grosser Bedeutung; Eine Temperatur von 25-35°C und nicht zu trockene oder zu feuchte Luft.

Das ist äusserst nützlich, denn somit absorbieren die Wände kein Licht und heizen sich auf, sondern das Licht, das nicht direkt auf die Blätter trifft, wird weiter umherreflektiert, bis es schliesslich beim Blatt ankommt.