GBS-Fachschule für Technik und Wirtschaft Leipzig

Facharbeit

Thema: GPS Tracker für das Fahrrad

Name: William Gramsch und Marcus Leßmann

Klasse: 19.E2

Datum: 15.02.2021

Fachrichtung: Elektrotechnik

Schwerpunkt: Energie- und Automatisierungstechnik

Betreuer: Herr Danny Schreiter

Inhaltsverzeichnis

Abbildungsverzeichnis

Tabellenverzeichnis

Formelverzeichnis

1 Motivation

Das Fahrrad ist ein beliebtes Verkehrsmittel in unserer heutigen Gesellschaft. Nicht nur als Fortbewegungsmittel in Freizeit und Privatleben, sondern auch als Dienstfahrzeug ist es für viele Menschen eine umweltfreundliche Alternative zu Auto, Bus oder Bahn. Zudem ist das Fahrrad leise, effizient, kostengünstig und fördert die Gesundheit.

Besonders in großen Städten, wie zum Beispiel in Leipzig, ist man mit dem Fahrrad meist schneller am Ziel als mit dem Auto. Diese Aussage wird durch das nachfolgende Diagramm des Umweltbundesamtes bestärkt. Darin ist zu erkennen, dass 40 bis 50 Prozent der Autofahrten in Großstädten eine Strecke von unter 5 Kilometern betreffen. Bei dieser geringen Entfernung wäre allerding der Weg mit dem Fahrrad die schnellere Alternative.[1]

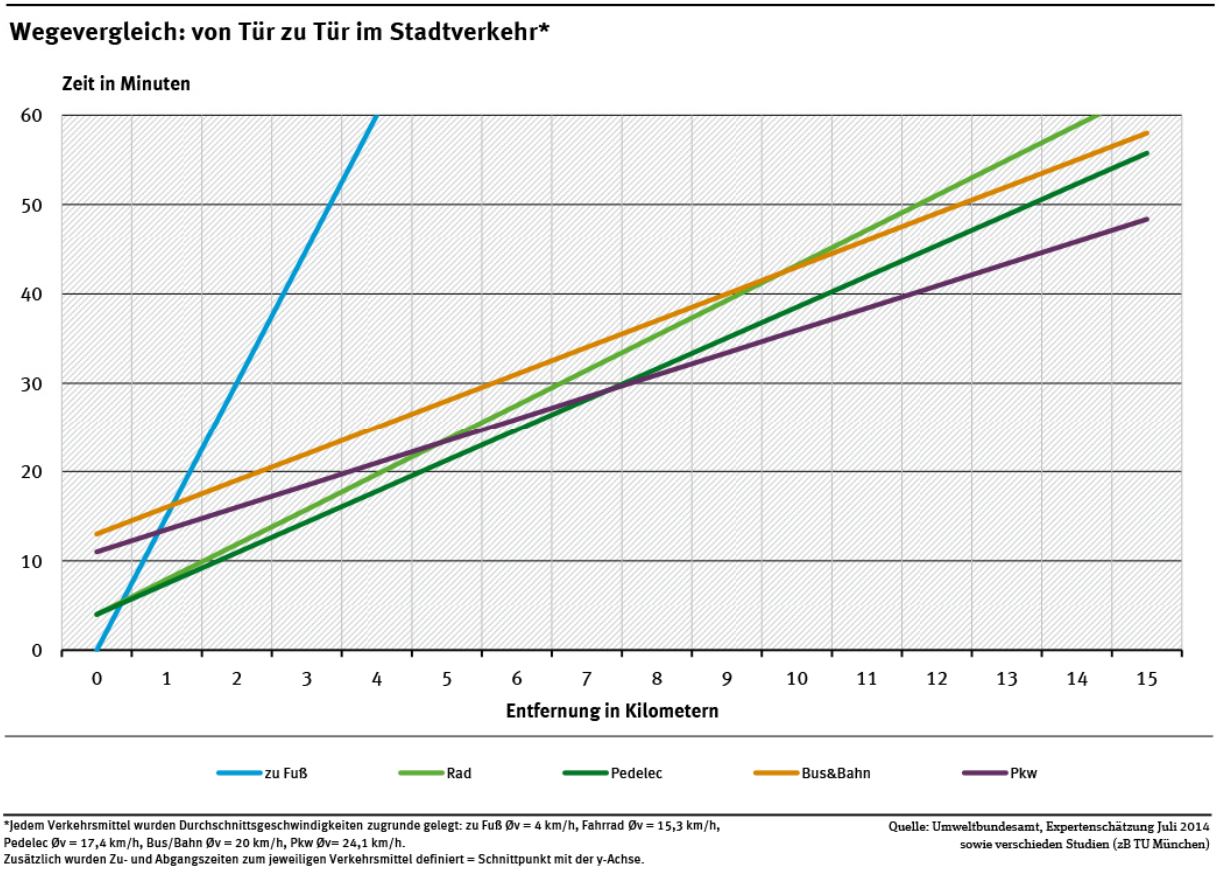


Abbildung 1: Wegvergleich
https://www.umweltbundesamt.de

[1] Corinne Meunier Jonas Stoll Laura Schoen (2021).

Doch die Euphorie des klimaschonenderen Verkehrsmittels wird gedämpft durch zahlreiche Nachrichten von Freunden, Bekannten oder der Presse. Denn es verschwinden laut Statistiken beinahe 800 Fahrräder in Deutschland am Tag. Im Jahr 2019 wurden insgesamt rund 155.000 Fahrräder als gestohlen gemeldet.[2] Besonders die Stadt Leipzig sticht dabei als Negativbeispiel heraus: Gemessen an der Einwohnerzahl werden in keiner deutschen Stadt so viele Räder gestohlen wie in Leipzig. Im Jahr 2019 wurden im Freistaat Sachsen über 21.000 Fahrräder geklaut.[3]Da die Verfasser dieser Arbeit selbst genügend Freunde und Kollegen kennen, denen teilweise sogar mehrmals die Fahrräder oder Teile davon entwendet wurden, haben sie sich Gedanken gemacht, wie dies verhindert werden kann.

Abbildung 2: Fahrraddiebstähle
https://de.statista.com

[2] Mirko Wenig (2020).
[3] MDR Sachsen (2020).

2 Themenfindung

Die Themenfindung der Projektarbeit ergab sich in einem fortlaufenden Prozess der Diskussion, wobei optionale Projekte seitens der Schule und privater Möglichkeiten abgewogen wurden. Im Juni 2020 fiel die Entscheidung auf das Thema „GPS-Tracker für das Fahrrad“. Das Projekt erschien besonders reizvoll, weil es sich um ein eigenständiges Projekt handelt, das gleichzeitig das Bedürfnis der Gruppenmitglieder, das eigene Fahrrad vor Diebstahl zu schützen, abdeckt.
Der GPS-Tracker umfasst eine über SMS ansteuerbare Auskunft über den Akkuladezustand und den Standort des Fahrrads. Außerdem soll ein Bewegungsalarm an- und ausgeschalten werden können.

3 Realisierung des Projektes

Zur Realisierung des Projektes wird ein Mikrocontroller, der mit der Eingabesprache C++ und der Software Arduino IDE programmiert werden kann, verwendet.
Darüber sind bereits Grundkenntnisse aus der Schule vorhanden und die Software bietet einen anschaulichen Einstieg in die Programmierung. Zur Kommunikation via SMS ist ein GSM-Modul nötig, welches mit einer SIM-Karte bestückt werden muss, um sich in das Mobilfunknetz einzuwählen. Um einen genauen Standort zu ermitteln, bedarf es außerdem eines GPS-Moduls. Für die Realisierung steht ein Rennrad Reynolds 501 der Marke Raleigh zur Verfügung. Hierbei wurde sich an den Maßen des Fahrrads orientiert, um mögliche Schwierigkeiten beim Einbau zu berücksichtigen. Die abschließende Präsentation der Arbeit erfolgt an der GBS Fachschule für Technik und Wirtschaft.

Abbildung 3: Rennrad Reynolds 501 der Marke Raleigh

4 Ziel des Projektes

Das Ziel des Projektes ist es, einen voll funktionsfähigen GPS-Tracker zusammenzustellen und zu programmieren. Es wurden diverse Zielsetzungen herausgearbeitet:

- Über verschiedene Befehle per SMS soll der Tracker ansteuerbar sein.
- Bei einer SMS mit dem Wortlaut „GpsDaten“ soll der Standort des Fahrrads in Form eines Google-Maps-Links an die Handynummer zurückgeschickt werden, welche den Befehl gesendet hat.
- Sollte nach zwei Stunden noch kein GPS- Signal gefunden worden sein, soll eine Nachricht dem Fahrradbesitzer mitteilen, dass keine GPS-Ortung möglich ist.
- Bei der Textnachricht „Akkustand“ soll der Arduino die Spannung des verbauten Akkus messen und eine ungefähre Prozentzahl des Akkus zurücksenden.
- Wird eine Nachricht mit „AlarmAn“ versendet, so soll der Arduino einen Operationsverstärker, der mit einem Piezo-Kristall gekoppelt ist, mit Spannung versorgen.
  Wird das Fahrrad nun bewegt, während der Alarm eingeschaltet ist, schickt der Tracker eine Alarm-SMS mit dem Inhalt „Achtung, Fahrrad wird bewegt!“
- Natürlich soll der Alarm auch mit dem Befehl „AlarmAus“ wieder deaktiviert werden können.
- Optional soll es möglich sein, den Akku über den Dynamo des Fahrrads zu laden, um eine längere Akkulaufzeit zu gewährleisten.
- Insgesamt sollte alles so verbaut werden, dass es nicht möglich ist, den GPS-Tracker zu enttarnen oder entfernen zu können.

5 Organisation

5.1 Zeitablaufplan

Arbeitsschritte	Erläuterung	Erledigen bis
Thema eingrenzen	➢ Ideensammlung ➢ Berücksichtigen persönlicher Interessen ➢ Machbarkeit prüfen ➢ Verfügbarkeit prüfen (gibt es ähnliche Systeme? Wenn ja, funktionieren diese?)	Anfang Juni 2020
Informationen/Material sichten	➢ Informationen aus dem Internet beschaffen bzw. Bücher erwerben ➢ mithilfe von YouTube-Videos erste Vorstellungen des Aufbaus sammeln ➢ Grobgliederung erstellen	Ende Juni 2020
Material sichten und bearbeiten	➢ alle Bauteile bestellen ➢ Bauteile auf Funktion prüfen ➢ erste Probeaufbauten ➢ Dokumentieren	Anfang August 2020
Aufbau Gleichrichter/Operationsverstärker	➢ Teile zusammensuchen ➢ Schaltplan erstellen ➢ Platine erstellen ➢ Funktionsprobe ➢ Dokumentation	Anfang September 2020

Aufbau Dynamo-Simulation	➢ Teile zusammensuchen ➢ Schaltplan erstellen ➢ Aufbau ➢ Funktionsprobe ➢ Dokumentation	Ende September 2020
Aufbau Hauptschaltung, Dokumentation, Zusammenfassen	➢ Bauteile/Komponenten zusammensuchen ➢ Schaltplan erstellen ➢ Bauteile auf einem Breadboard zusammenschalten und testen ➢ Dokumentationen zusammenfassen (Bilder, Skripte, Grafiken, Tabellen)	Anfang November 2020
Programm	➢ Ideen aus Internet/Büchern zusammentragen ➢ Planen des Programmes ➢ Programm übertragen und testen	Ende Dezember 2020
Entwurf erstellen	➢ Formulierung des Textes mit Anmerkungen für spätere Verbesserungen	Ende Dezember 2020
Endfassung	➢ Schreiben von: Einleitung, Hauptteil, Zusammenfassung, Verzeichnisse ➢ Überarbeiten aller Dokumente	Ende Januar 2021
Abgabe		Februar 2021

Tabelle 1: Zeitablaufplan

5.2 Durchführung

Bereits in der Planungsphase wurde der Zeitablaufplan entworfen, um einen Überblick zu erhalten. Es folgten Recherchearbeiten nach Komponenten zur Umsetzung des Projekts, die Bestellung aller Module und Materialien und die ersten Probeaufbauten und Versuche begannen.
Zu dieser Zeit wurde sich ebenfalls mit dem Schreiben des Programmes auseinandergesetzt. Beispielsweise welche Bibliotheken verwendet werden können und wie die Komponenten miteinander verbunden werden. Als dies geklärt war, wurden alle Einzelteile verlötet und auf mehrere Steckplatinen gebracht. Zum Schluss wurde das Programm aufgespielt und auf seine einzelnen Funktionen getestet.

5.3 Eigenanteil

Der Eigenanteil der Facharbeit besteht aus:

- der Organisation des Projektes,
- der Wahl der Komponenten,
- der richtigen Verkabelung der Bauteile,
- dem Schreiben des Programms mithilfe ausgewählter Bibliotheken,
- sämtlichen projektbezogenen Tests und Messungen,
- der Montage am Fahrrad,
- dem Erkennen von Problemen,
- der Problemanalyse,
- der Behebung dieser Fehler (wenn möglich)

6 Verwendete Bauteile und ihre Funktionen

6.1 Schaltung für den Anschluss eines Dynamos

Um den Anschluss eines Dynamos zu gewährleisten, ist eine Schaltung notwendig, die den Strom und die Spannung verändert. Damit ist es möglich, einen Akku zu laden, ohne dass dieser dabei Schaden nimmt. In den folgenden Punkten wird näher auf die dabei verwendeten Bauteile und ihre Funktionen eingegangen.

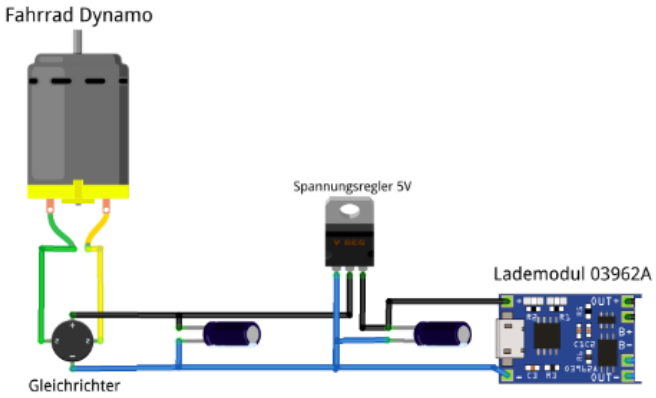


Abbildung 4: Dynamoschaltung

6.1.1 Allgemeine Beschreibung Spannungsregler

Spannungsregler stabilisieren und begrenzen eine Spannung. Sie kommen überall zum Einsatz, wo eine geregelte Gleichspannung benötigt wird. Wenn eine Spannung zu sehr schwankt und zu hohe Spannungsspitzen aufweist, können Bauteile nicht wie gewünscht arbeiten oder kaputt gehen. Mit dem Einsatz von Spannungsreglern können Spannungswerte konstant gehalten werden. Die wohl bekannteste Baureihe ist hierbei die 78xx, wobei die xx für die zu regelnde Spannung stehen. Bei dem verwendeten L7805 wird die Spannung also konstant auf 5V reguliert. Die ersten beiden Ziffern deuten darauf hin, ob es sich um einen positiven (78) oder einen negativen (79) Regler handelt. [4]

[4] Elektronik Kompendium (2021).

6.1.2 Funktionsweise von Spannungsreglern

Im Prinzip funktionieren Spannungsregler einfach und sind unkompliziert aufgebaut. Die Regler vergleichen die gewünschte Referenzspannung mit der Ausgangsspannung. Wird der Referenzwert unterschritten, wird ein Transistor stärker angesteuert, sodass der Stromfluss vergrößert wird. Dadurch steigt die Spannung, bis der gewünschte Wert erreicht ist. Überschreitet die Ausgangsspannung den Referenzwert, schließt der Transistor weiter, bis sich die Spannung eingepegelt hat. Der Transistor fungiert wie eine Art einstellbarer Widerstand. Der Anteil, der am Ausgang nicht gebraucht wird, verheizt nur am Gehäuse. Dadurch hat der lineare Spannungsregler eine relative Verlustleistung und ein Kühlkörper wird benötigt.[5]

6.1.3 Halbleitergleichrichter Funktionsweise

Gleichrichter haben den Zweck aus einer Wechselspannung, wie sie beispielsweise aus einem betriebenen Dynamo kommt, eine Gleichspannung zu erzeugen.
Im Prinzip lassen Gleichrichtiger den Wechselstrom nur in Durchlassrichtung fließen und sperren ihn komplett in der Sperrrichtung, womit eine pulsierende Gleichspannung entsteht. Da dadurch aber die negative Halbwelle fehlt und in der Zeit auch keine Elektronen fließen, ist die mittlere Stromstärke gering und für viele Anwendungen nicht nutzbar. Diese Schaltung nennt man die Einwegschaltung. Deutlich effektiver ist die Zweiweggleichrichtung, bei der auch die negative Halbwelle in dieselbe Richtung gebracht wird. Um diese pulsierende Gleichspannung besser nutzbar zu machen, sollte sie jedoch geglättet werden. Dabei werden zwei Elektrolytkondensatoren benutzt, auf die in Punkt 6.1.7 eingegangen wird.[6]

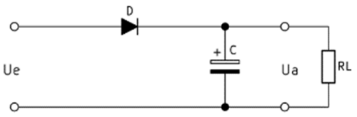


Abbildung 6: Einweggleichrichter
https://electronicbase.net

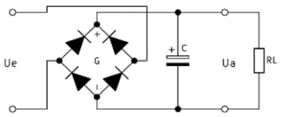


Abbildung 5: Zweiweggleichrichter
https://electronicbase.net

[5] SprinterSB BMS (2015).
[6] Bernhard Grotz (2018).

6.1.4 Der Gleichrichter in der Dynamo Schaltung

Der Gleichrichter dient dazu, die Wechselspannung vom Fahrraddynamo in eine pulsierende Gleichspannung zu richten. Das ist der erste Schritt, um die Spannung zum Laden des Akkus nutzbar zu machen. Verwendet wird hierfür ein handelsüblicher Zweiwegbrückengleichrichter

6.1.5 Kondensator Funktionsbeschreibung

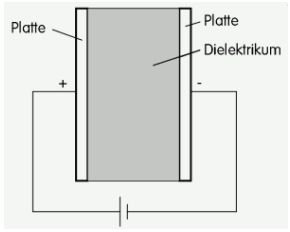


Abbildung 7: Aufbau Kondensator
http://blog.openptv.org

Der Kondensator ist ein Bauteil, welches elektrische Ladung speichern und wieder abgeben kann. Er besteht aus zwei voneinander getrennten Leiteranordnungen, die durch ein Dielektrikum voneinander getrennt sind. Wird zwischen den Leiteranordnungen eine Spannung angelegt, sammeln sich auf ihren Oberflächen, getrennt voneinander, positiv und negativ geladene Ladungsträger an. Zwischen den Leitern baut sich ein elektrisches Feld auf, in dem Feldenergie gespeichert wird. Die Energie kann später wieder als Spannungsentladung abgegeben werden.
Durch diese Funktion kann eine pulsierende Spannung geglättet werden. Bei Spannungsspitzen speichert der Kondensator Feldenergie und gibt sie bei den Spannungstälern wieder ab.[7]

[7] Duden Learnattack GmbH (2021).

6.1.6 Bauformen von Kondensatoren

Kondensator	Kapazität	Spannung	selbstheilend	gepolt
Papier	100 pF...1µF	125...1000 V	nein	nein
Metall-Papier	0,1...50 µF	160...600 V	ja	nein
Styroflex	2 pF...50 nF	50...500 V	nein	nein
Metall-Kunststoff	0,01...0,25 µF	300 V...5 kV	ja	nein
Metall-Lack	0,1...200 µF	60...120 V	ja	nein
Keramik	0,5 pF...100 nF	250...500 V	nein	nein
Elektrolyt	0,5...10000 µF	3...650 V	ja	ja

Tabelle 2: Kondensator Vergleich
https://www.elektronik-kompendium.de

Folienkondensator: Dies sind Kondensatoren, deren Dielektrikum aus nichtleitender Folie besteht. Man erkennt sie leicht an ihrer quaderförmigen/zylindrischen Form. Unter den Folienkondensatoren gibt es viele unterschiedliche Arten wie z.B. MKT, MTC, MKP uvm. Durch seine unzähligen Wicklungen weist er eine hohe parasitäre Induktivität auf.

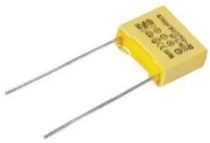


Abbildung 8: Folienkondensator
https://www.pk-components.de

Einsatzgebiete:

- Glättungskondensator
- Hochspannungskondensator
- Stoß- und Stützkondensator

Keramikkondensator: Wie auch schon bei dem Folienkondensator beschreibt der Name das Dielektrikum (in diesem Fall Keramikoxid). Diese Schicht ist sehr durchschlagsfest und aus diesem Grund sind Keramikkondensatoren auch sehr spannungsfest.

Abbildung 9: Keramikkondensator
https://www.pollin.de

Einsatzgebiete:

- Abschneiden von Spannungsspitzen
- Zwischenspeicher IC
- Schwingkreise

Elektrolytkondensator: Der größte Unterschied des Elektrolytkondensators zu den anderen Kondensatoren ist, dass hier das Dielektrikum nicht fest, sondern flüssig ist. Dies ermöglicht eine höhere Kapazität gegenüber den anderen Bauformen. Der Nachteil des Kondensators liegt bei seiner Kapazitätsbezeichnung. Diese ist eher eine Schätzung, denn hier kommen Toleranzen von bis zu +/- 50% vor. Im Laufe der Zeit kann es auch vorkommen, dass sie auslaufen und an Kapazität verlieren. Von großer Bedeutung ist bei diesem Kondensator die Polung. Wird diese falsch herum angeschlossen, verdampft das Elektrolyt und der Kondensator explodiert.[8]

Abbildung 10: Elektrolytkondensator
https://www.conrad.de

Einsatzgebiete:

- Stützkondensatoren
- Koppelelemente NF Signale
- Glättungsglieder

[8] Electronics-tutorials (2019).

6.1.7 Kondensatoren in der Dynamo-Schaltung

Die beiden Kondensatoren dienen zur Überbrückung von Spannungstälern und um Schwingungen zu minimieren (Glättung). In der Schaltung werden Elektrolytkondensatoren benutzt, um große Spannungstäler abzudämpfen und auch zum Glätten eignen sich diese Kondensatoren sehr gut.

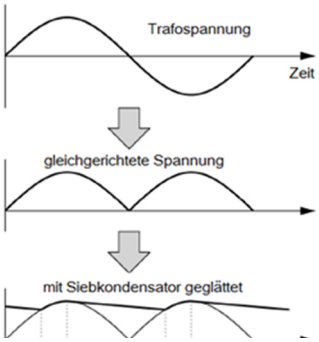


Abbildung 11: Glätten der Spannung
https://www.analog-forum.de

6.1.8 Lademodul TP4056

Bei dem Lademodul handelt es sich um ein TP4056. Dieses Modul wurde für 18650 Lithium-Ionen-Akkumulatoren entwickelt. Durch ihn ist es möglich, den Akku via USB-Stecker oder anderen Spannungsquellen zu laden. Sehr wichtig ist hierbei ein Tiefenentladeschutz und ein Überladeschutz.

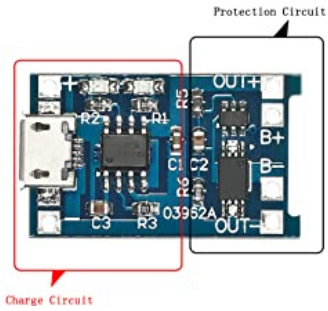


Abbildung 12: Lademodul TP4056
https://www.gjoshiben.com

Spezifikationen:

- Eingangsspannung: 5V
- Ladungsabschaltspannung: 4,2 V (+/-) 1%
- Maximaler Ladestrom: 1000mA
- Batterie-Überentladungsschutzspannung: 2,5V
- Überstromschutzstrom der Batterie: 3A
- Abmessungen: Ca. 27mm x 17 mm[9]

[9] Bastelgarage.ch (2021).

6.2 Boost Converter

Abbildung 13: Boost Converter
https://www.amazon.de

Der verwendete Lithium-Akku besitzt mit dem Lademodul nur einen Spannungsbereich von 4,2 bis 2,8 Volt. In diesem Bereich kann der verwendete Arduino jedoch nicht ordnungsgemäß arbeiten. Durch die Verwendung eines Boost Converters ist es dennoch möglich, mit einer eigentlich zu geringen Spannung den Arduino zu betreiben. Der Gleichspannungswandler kann mithilfe eines auf ihm befindlichen Potenziometers auf eine gewünschte Ausgangsspannung eingestellt werden. In der Schaltung wird er dazu verwendet, die Spannung auf das gewünschte Level von 5V zu erhöhen.

Spezifikationen:

- Verluste: 7%
- max. Ausgangsstrom: 2000mA
- Eingangsspannung: 2V-24V
- Ausgangsspannung: 3V-28V
- Abmessungen: 36mm x 17mm x 6,5mm

6.3 Der verwendete Microcontroller

Als zentrales Verarbeitungsorgan wird ein Arduino Nano genutzt, der die Anweisungen verarbeitet und die im Programm festgelegten Befehle ausführt. Er bietet dieselben Spezifikationen wie ein Arduino Uno. Ausschlaggebend ist jedoch, dass er trotz derselben Rechenleistung eine perfekte Größe für das thematisierte Projekt hat. Mit seinen 18mm in der Breite kann er leicht in einem Sattelstangeneinschub Platz finden.

6.3.1 Pinbelegung des Arduino Nanos

- D0-D13 Digitale Ein/Ausgangspins
- A0-A7 Analoge Ein/Ausgangspins
- Pins. 3, 5, 6, 9, 11 Pulsbreitenmodulation (PMW)
- Rx, Tx serielle Kommunikation
- Pin 10-13 SPI-Kommunikation

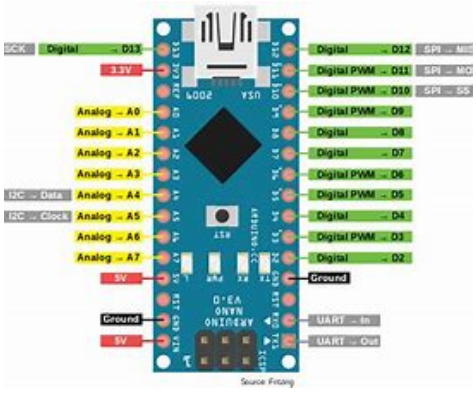


Abbildung 14: Arduino Pinbelegung
https://diyi0t.com

6.4 Akkumulator

6.4.1 Allgemeiner Aufbau

In der heutigen Zeit sind die meisten tragbaren Geräte mit einem Akkumulator ausgestattet. Akkumulatoren, kurz Akkus, sind wiederaufladbare Energiequellen.
Diese bestehen meist aus zwei Materialien, die als Elektroden dienen und in ein Elektrolyt getaucht sind. Beim Aufladen eines Akkus wird elektrische Energie in chemische Energie umgewandelt. Beim Entladen des Akkus wird aus der chemischen Energie wieder elektrische. Die Anwendungsgebiete von Akkus unterscheiden sich in ihrer Art und ihren Eigenschaften.[10]

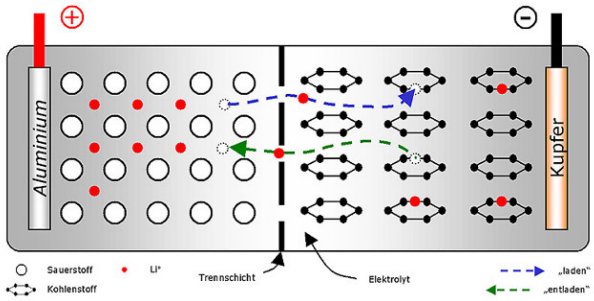


Abbildung 15: Aufbau Lithium Ionen Akku
https://www.fly2air.com

6.4.2 Akkutypen

Akkutyp	*Energiedichte Wh/kg*	*(Nenn) Zell-spannung*	*Ladewir-kungs- grad*	*Lebensdauer des Akkus*	*Selbstentla-dung im Mo-nat in %*
Bleiakku	30-40	2,0V	60-70%	4-8 Jahre 300-600 Z	5-10%
NiCd Akku	40-50	1,2V	70%	15 Jahre 800-1500 Z	10-15%
NiMH Akku	60-80	1,2V	70%	7-10 Jahre 350-500 Z	15-20%
NiMH LSD	60-80	1,2V	70%	7-10 Jahre 350-500 Z	1-2%
LiIon Akku	120-180	3,6V	90%	10-15 Jahre 500-800 Z	1-2%
LiPo Akku	130-150	3,7V	90%	7-10 Jahre 300-500 Z	1-2%

Tabelle 3: Akkutypen
http://www.akku-abc.de

[10] Chr. Caspari (2018).

6.4.3 Die richtige Wahl des Akkus

Die richtige Wahl des Akkumulators ist von sehr hoher Bedeutung. Er sichert die konstante Energiezufuhr zu den Bauteilen und bestimmt, wie lange das Fahrrad geortet werden kann, ohne geladen werden zu müssen. Wichtige Anforderungen wären in diesem Fall eine hohe Kapazität und eine geringe Selbstentladung, denn diese würde die Akkulaufzeit stark beeinflussen. Dieses Kriterium schließt den Nickel-Metallhydrid-Akku und den Nickel-Cadmium-Akku aus. Ein weiterer wichtiger Punkt ist die Bauform. Der Akku sollte so beschaffen sein, dass er mit in dem Fahrradrahmen Platz findet. Aus diesem Grund fiel die Wahl auch nicht auf einen Bleiakku. Die Energiedichte erwies sich ebenfalls noch als sehr wichtig, denn sie gibt Auskunft, wieviel Energie pro Gewicht (Masse) gespeichert werden kann. Diese gravimetrische Energiedichte beschreibt die KWh/kg, dort glänzen vor allem Lithium Akkus mit einer hoher Energiedichte.
So ist es möglich, auf relativ kleinem Raum einen Energiespeicher mit hoher Kapazität unterzubringen. Ein weiteres Kriterium, die Betriebstemperatur, schloss den Lithium-Polymer-Akku aus. Dieser kann nur bei einer Temperatur von 0-60 °C betrieben werden, ansonsten hat er starke Leistungseinbrüche, was ein Betrieb im Winter unmöglich machen würde.
Deswegen fiel die Wahl auf die 18650 Lithium-Ionenzellen. Sie besitzen die perfekte Bauform, eine hohe Kapazität, eine große Energiedichte und können bei einer Temperatur von -10 - 55°C betrieben werden.

6.5 GSM-Modul

6.5.1 Bedeutung GSM

GSM ist die Abkürzung für „The Globale System for Mobile Communication“. Diese umschreibt das weltweite volldigitale Kommunikationsnetz über Mobilfunk und Festnetz. Es ermöglicht den Datenaustausch über den Kurznachrichtendienst (engl.: Short Message Service=SMS) von Netztechnik und Mobiltelefonen.

6.5.2 Was ist ein GSM-Modul?

Ein GSM-Modul ist ein Funkmodul, welches Daten oder Informationen an einen GSM-Empfänger senden kann. Dafür ist es grundsätzlich mit einer SIM-Karte ausgestattet, welche das Einwählen in ein Mobilfunknetz ermöglicht. Die Bauform kann, je nach Anwendungsgebiet, unterschiedlich groß ausfallen. Bei der Ortung von Schiffen, Containern oder Fahrzeugen wird ein GPS-Modul eingesetzt. Dazu wird der Standort des Moduls über den Kurznachrichtendienst (SMS) übermittelt.
Auch Signale von ausgelösten Alarmen (z.B. in Häusern oder Räumen), hilfsbedürftigen Menschen in Notsituationen oder verunfallten Fahrzeugen, die per Knopfdruck einen Notruf absenden, werden mit einem GSM-Modul verschickt.[11]

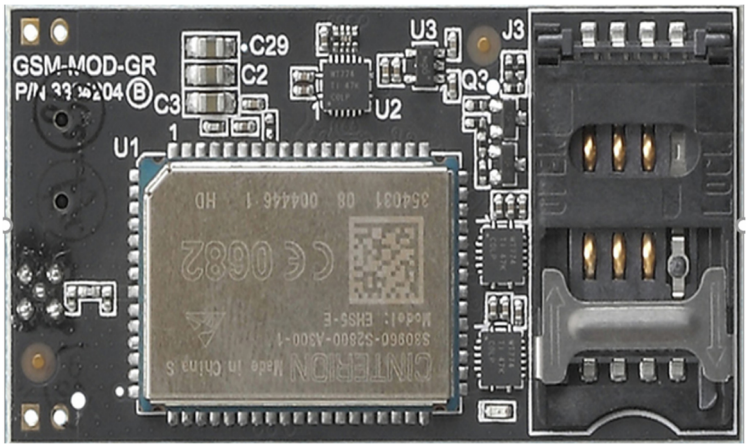


Abbildung 16: GSM-Modul
https://www.voltking.de

[11] Hannes Ühss (2020).

6.5.3 Das verwendete GSM Modul

Das GSM Modul Sim800l ist ein sehr kleines und leistungstsarkes Modul, welches durch seine Abmessungen(LxBxH: 23 mm x 35 mm x 5,6 mm) perfekt für das Projekt gemacht ist. Über dieses Modul werden später Informationen und der GPS-Standort an das gewünschte mobile Endgerät verschickt. Es dient zur Kommunikation mit dem GPS-Tracker. Der Arbeitsstrom im Schlafmodus liegt bei 2mA, das spart Strom und verlängert zusätzlich die Akkulaufzeit. Durch die Möglichkeit, eine externe Antenne anzuschließen, ist es gegebenenfalls möglich diese zu verlängern und so zu plazieren, dass ein guter Empfang gegeben ist.

Abbildung 17: Sim800l Modul
https://www.fambach.net

6.6 GPS Modul

6.6.1 Bedeutung GPS

GPS ist die Abkürzung für „Global Positioning System“. Das beschreibt ein weltweites System, welches zur Positionsbestimmung genutzt wird.

6.6.2 Was ist GPS-Tracking?

GPS-Tracking ist die Ermittlung der Koordinaten eines bestimmten Objektes.
Durch Ermittlung des Breiten – und Längengrades in regelmäßigen Abständen wird die Position des Objektes bestimmt.

6.6.3 Funktionsweise GPS-Tracking

Für eine einwandfreie GPS-Ortung sind Satelliten im Weltraum nötig, welche codierte Radiosignale und die dadurch folgenden Informationen zu Position, Uhrzeit und Umlaufbahn senden. Ein GPS-Empfänger, der die Laufzeiten der Signale von mindestens 4 Satelliten braucht, vergleicht die Uhrzeit der Aussendung und die Uhrzeit des Eingangs der einzelnen Signale. Dadurch errechnet er sich sogenannte Entfernungskreise. Dort, wo sich diese Entfernungskreise überschneiden, befindet sich in etwa der Empfänger. Je mehr Signale der Empfänger erhält, desto genauer kann er den Standort errechnen.[12]

[12] Kirstin Welther (2020).

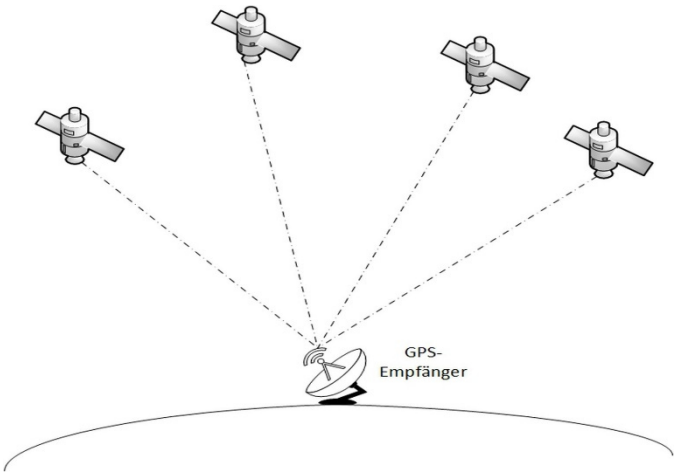


Abbildung 18: Funktionsweise GPS
https://praxistipps.chip.de

6.6.4 Das verwendete GPS-Modul

Das erste GPS-Modul, welches bestellt wurde, war das Neo-6M Modul.
Dies ist für etwa 5 Euro ein ziemlich günstiges Modell. Bei den ersten Tests und Versuchen wurde schnell klar, welche Probleme mit diesem Modul auftreten werden. Eine erfolgreiche Verbindung zu den Satelliten war in Räumlichkeiten nur an einem Fenster möglich und dauerte über 30 Minuten. Auch die Standortbestimmung erwies sich als ziemlich ungenau. Das entsprach ganz und gar nicht dem Ziel des Projekts. Nach eingehender Recherche wurde sich dann für das MTK3339 GPS Entwicklerboard entschieden. Dies wurde für 40 Euro erworben und ließ durch die Datenblätter auf eine Verbesserung der Genauigkeit, eine schnellere Standortbestimmung und eine Verbindung zu den Satelliten innerhalb von Räumlichkeiten hoffen.

Module:	Neo-6M Modul	Mtk3339Entwick-lerboard
Anzahl der Satelliten:	22 Satelliten auf 50 Kanäle	22 Satelliten auf 66 Kanäle
Aktualisierung:	Bis 5Hz	Bis 10Hz
Antennenempfindlichkeit:	-161dBm	-165dBm
Strom:	60mA	30mA
Vorteile:	-	-mit Störsendererkennung und Reduzierung -kann direkt am Ausgang des Arduinos und ohne MOSFET betrieben werden

Tabelle 4: Vergleich GPS-Module

Das Ergebnis der ersten Tests war großartig. Das Board stellte mitten in einem Raum innerhalb weniger Minuten eine Verbindung zu den Satelliten her und die Genauigkeit des Standortes hatte sich deutlich verbessert. Die Standortbestimmung ist für das Projekt das Wichtigste, hier sollten keine Einsparungen vorgenommen werden. Deswegen ist die Entscheidung für das deutlich teurere GPS-Modul genau die richtige gewesen. Noch ein Vorteil ist, dass das Modul direkt am Ausgang des Arduinos betrieben werden kann. Das Mtk3339 benötigt einen maximalen Strom von 30 mA. An einem Arduino Ausgang dürfen maximal 40mA fließen. Der Neo6m wurde aus diesem Grund vorher über einen MOSFET geschalten.

6.7 MOSFET

6.7.1 Allgemeine Beschreibung

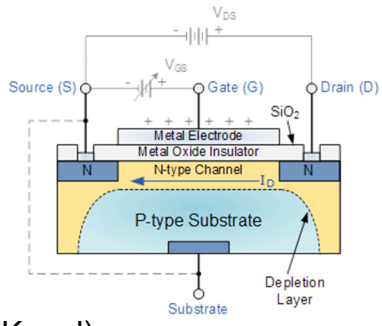


Abbildung 19: Aufbau MOSFET
https://www.electronics-tutorials.ws

Bei einem MOSFET handelt es sich um einen Metalloxid-Halbleiter-Feldeffekttransistor. Bei diesem Transistor besteht die Gate-Elektrode aus Metalloxid, welches durch eine hauchdünne Siliziumoxidschicht elektrisch von dem Haupthalbleiter (N-Kanal oder P-Kanal) getrennt ist.
Diese extrem dünne Schicht um das Metalloxid fungiert als eine Art Kondensator.

6.7.2 Verschiedene Typen

Bei den MOSFETs gibt es vier verschiedene Bauarten: den P-Kanal, den N-Kanal und den Anreicherungstyp beziehungsweise den Verarmungstyp. Der wesentliche Unterschied zwischen N-Kanal und P-Kanal ist, dass der N-Kanal durch Elektronen und der P-Kanal durch Löcher leitet. N-Kanal MOSFETs leiten bei positiver Gate-Source Spannung und P-Kanal MOSEFETs leiten bei negativer Gate-Source Spannung. Bei dem Verarmungstyp benötigt der Transistor eine Gate-Source Spannung, um das Gerät auszuschalten. Dies entspricht umgangssprachlich einem „Öffner"-Schalter. Bei dem Verarmungstyp benötigt der Transistor eine Gate Source Spannung, um das Gerät anzuschalten, dies entspricht einem „Schließer"-Schalter. [13]

[13] Electronics-tutorials (2020).

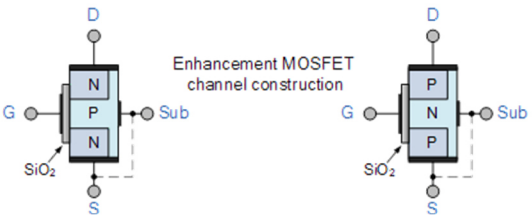


Abbildung 20: P. und N-Kanal
https://www.electronics-tutorials.ws

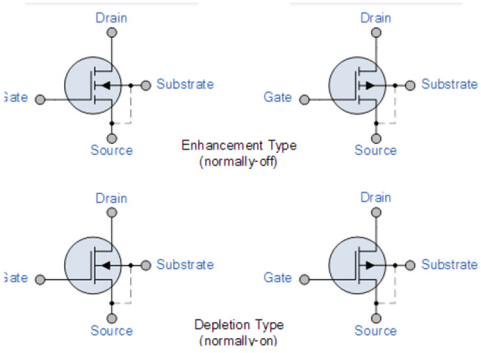


Abbildung 21: MOSFET Typen
https://www.electronics-tutorials.ws

6.7.3 Die Verwendung des MOSFETs

Anfangs lag die Verwendung des MOSFETs in einem elektrischen Schalter, der über einen Pin-Ausgang des Arduinos zum Durchsteuern gebracht wurde. Dadurch war es möglich, trotz höherer Stromaufnahme, das anfangs genutzte GPS-Modul Neo 6m an- und ausschalten zu können. Somit hat das GPS-Modul nur Strom verbraucht, wenn es gebraucht wurde. Verwendet wurde dafür ein N-Kanal Verarmungstyp MOSFET. Letztendlich wird mit dem neuen GPS Modul diese Schaltung gespart, da das Mtk3339 Entwicklerboard nur 30ma benötigt.

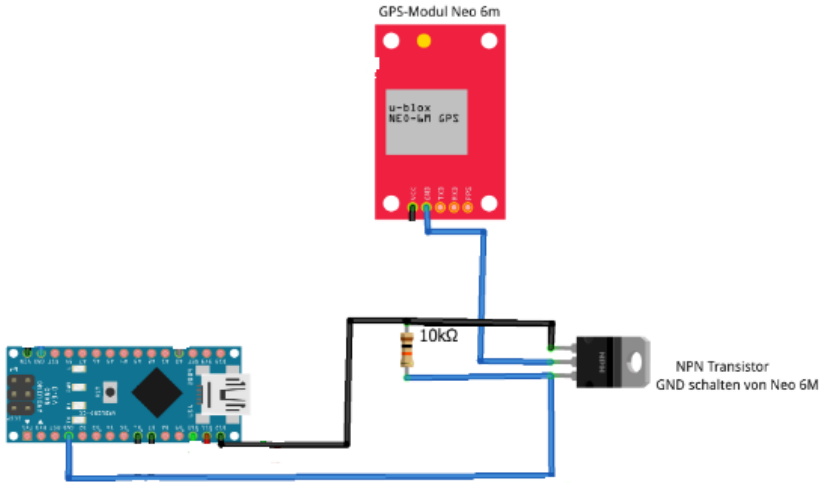


Abbildung 22: Mosfetschaltung Neo6m

6.8 Operationsverstärker

6.8.1 Allgemeine Information

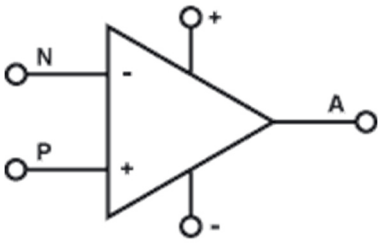


Abbildung 23: OPV
https://www.elektronik-kompendium.de

Der Operationsverstärker wird meist mit OP oder OPV abgekürzt. Er wird als eine der größten Erfindungen der Elektronik beschrieben. Ein Leben ohne ihn wäre nach heutigen Maßstäben kaum möglich. Ein OPV ist ein mehrstufig galvanisch gekoppelter Differenzverstärker. Der normale Operationsverstärker ist ein Spannungsverstärker. Er verstärkt eine differentielle Eingangsspannung auf einen massebezogenen Spannungsausgang. OPV werden unter anderem als Differenzierer, Integrierer, Verstärker, Filter, usw. eingesetzt.[14]

Schaltzeichen:

Veraltet aktuell

Abbildung 24: Schaltzeichen OPV
https://www.frustfrei-lernen.de

[14] Thomas Schaerer (2021).

6.8.2 Kenngrößen

Kenngröße	Idealer OPV	Realer OPV
Verstärkungsfaktor V	unendlich	1.000.000
Eingangswiderstand R_e	unendlich Ω	1 MΩ bis 1000 MΩ
Untere Grenzfrequenz f_{min}	0 Hz	0 Hz
Unitity-Gain-Frequenz-Bandbreite	unendlich Hz	> 100 MHz
Gleichtaktverstärkung V_{Gl}	0	0,2
Gleichtaktunterdrückung G	unendlich	5.000.000
Rausch-Ausgangsspannung U_{rausch}	0 V	3 µV

Tabelle 5: Vergleich OPV
https://www.elektronik-kompendium.de

6.8.3 Der verwendete Operationsverstärker

Im Projekt dient der Operationsverstärker als Spannungsverstärkung des Piezo-Kristalls. Der Piezo-Kristall fungiert als Bewegungsmelder im Falle eines Diebstahls. Wird das Fahrrad bewegt, erzeugt der Kristall eine Spannung im mV- Bereich. Um dieses Signal für den Arduino auswertbar zu machen, ist ein Operationsverstärker nötig. Dieser erzeugt bei kleinen Impulsen des Piezo-Kristalls ein für den Arduino wahrnehmbares Signal.

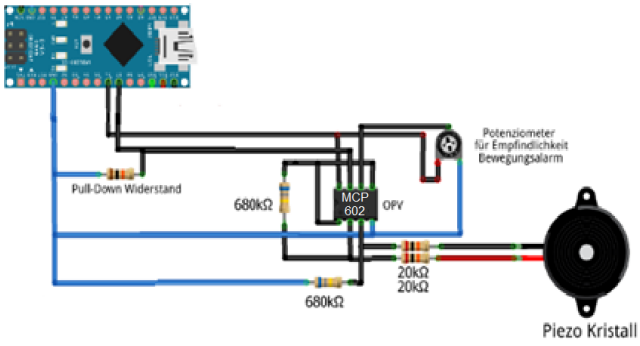


Abbildung 25: Operationsverstärker Schaltung

6.9 Piezoelement

6.9.1 Allgemeine Beschreibung

Das Piezoelement arbeitet auf Grundlage des Piezoeffekts, was bedeutet, dass durch mechanische Verformung bestimmter Kristalle (zum Beispiel Quarz und Turmalin) eine elektrische Ladung auf der Oberfläche der Kristalle entsteht. Ebenso verformen sich diese Kristalle bei dem Anlegen einer elektrischen Spannung. Dieser Effekt konnte erstmals im Jahr 1880 von den Brüdern Jacques und Pierre Curie nachgewiesen werden. Heutzutage werden meist sogenannte PZT-Keramiken (Blei-Zirkonat-Titanat) verwendet.

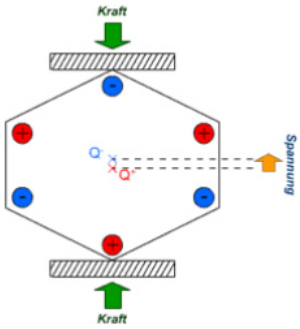


Abbildung 26: Piezoelement
https://vetsuisse.com

6.9.2 Funktionsweise

Wirken mechanische Kräfte entlang bestimmter Kristallachsen auf einen Piezowerkstoff ein, so verändern sich positiv und negativ geladene Kristallgitterpunkte. Daraus entsteht am Rand der angebrachten Elektroden eine elektrische Spannung, welche verhältnismäßig klein ist. In einem Piezofeuerzeug wird beispielsweise mithilfe eines Piezozünders ein großer Druck verwendet, um eine elektrische Spannung zu erzeugen. Der Funkenüberschlag entzündet dann eine Gasflamme.[15]

6.9.3 Die Verwendung des Piezoelements im Projekt

In dem Projekt wird das Piezoelement als eine Art Bewegungssensor genutzt. Durch das Hin- und Herbewegen im Fahrradrahmen wird das Element mechanisch belastet und gibt eine kleine Spannung aus. Da diese nicht ausreicht, wird sie mit einem OPV verstärkt, damit der Arduino diese auswerten und verarbeiten kann.

[15] chemie-schule.de (2021).

7 Schaltplan

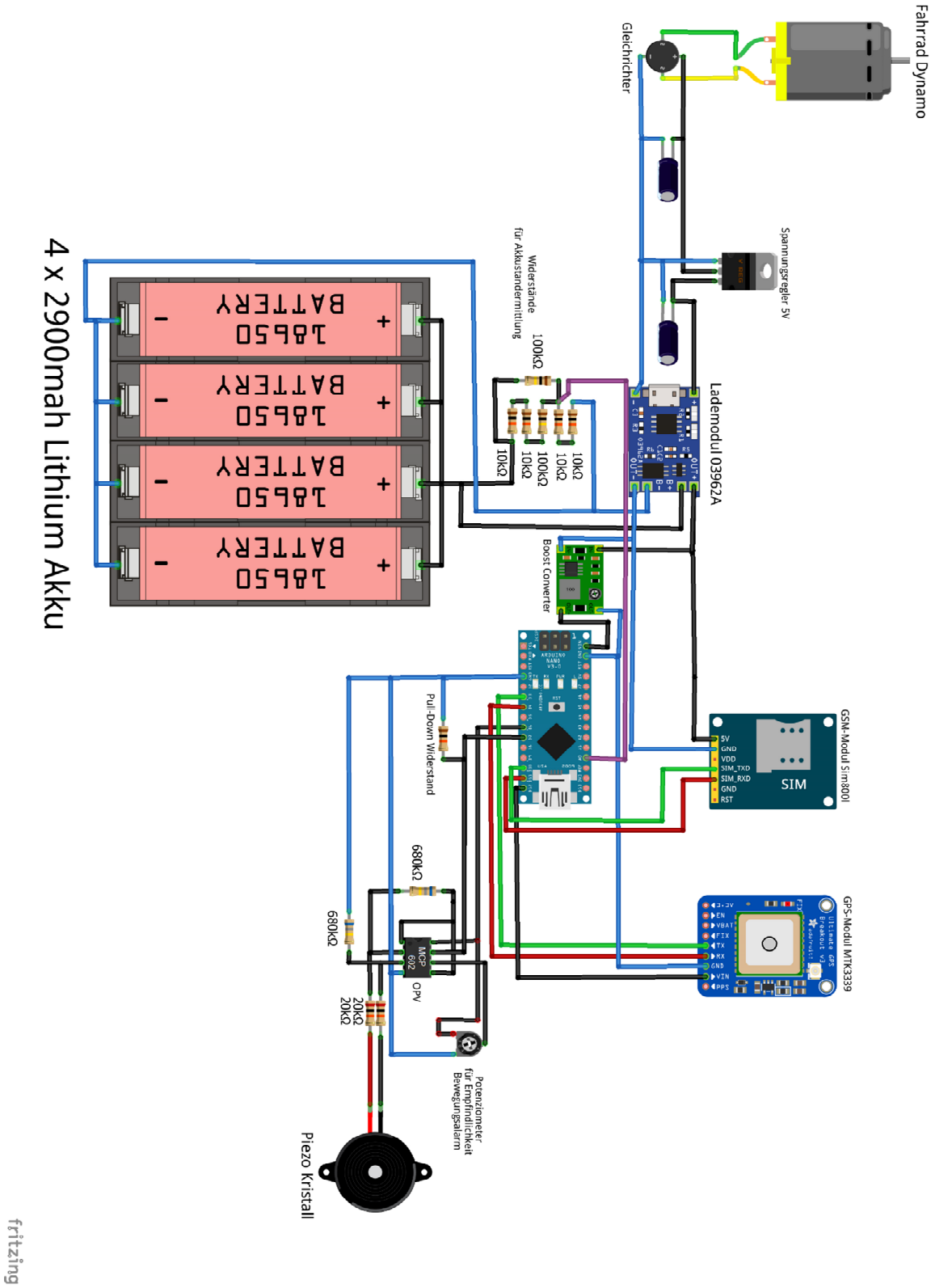


Abbildung 27: Schaltplan

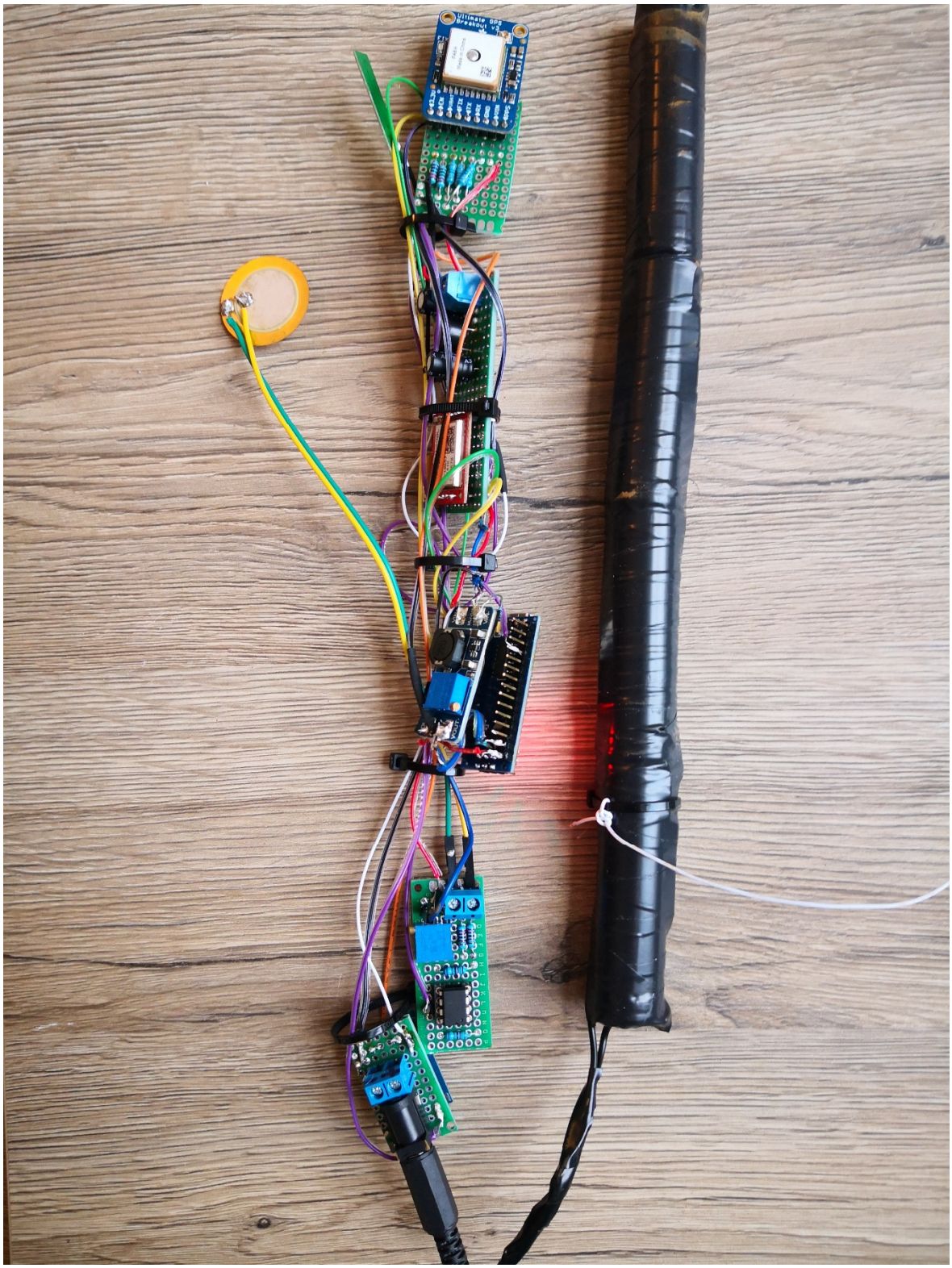


Abbildung 28: Realisierung des Schaltplans

8 Programm [16]

8.1 Bibliotheken und Zuweisungen

Am Anfang des Programms wurden die ausgewählten Bibliotheken hinzugefügt, die Module deklariert und die Pins für die serielle Kommunikation festgesetzt.

```
#include <Sim8001.h>                    // (1)
#include <SoftwareSerial.h>             // (2)
#include <TinyGPS++.h>                  // (3)
Sim8001 GSM;                            // (4)
SoftwareSerial SerialAT (10, 11);       // (5)
TinyGPSPlus gps;
SoftwareSerial ss(4, 3);
```

Abbildung 29: Bibliotheken

1. In der ersten Zeile wird die Bibliothek für das GSM-Modul hinzugefügt, diese erleichtert das Schreiben des Programms durch einfache kurze Befehle, welche ohne Bibliothek wesentlich komplexer wären.
2. Hier wird die Bibliothek *„SoftwareSerial“* hinzugefügt. Durch sie ist es möglich andere digitale Pins auch als serielle Pins dienen zu lassen. So ist es möglich, den Arduino mit mehreren Modulen kommunizieren zu lassen.
3. Nun wird noch die ausgewählte Bibliothek für die Kommunikation zum GPS-Modul eingepflegt. Diese erleichtert wieder das Schreiben des Programms. Damit sind alle Bibliotheken vorhanden, die benötigt werden.
4. Bei Punkt 4 wird nun bestimmt, wie das GSM-Modul künftig im Programm genannt wird.
5. Die beiden Zahlen in den Klammern (hier 10 und 11) bestimmen die seriellen Pins TX und RX. Dadurch weiß der Arduino dann, mithilfe welcher Pins er mit dem GSM-Modul kommuniziert.

[16] Danny Schreiter (2019).

8.2 Void Setup

Im *void setup* Teil wird die Kommunikation zu den Modulen oder dem Computer für eine Verbindung zum seriellen Monitor begonnen. Außerdem wird hier festgelegt, welche Pins des Arduinos als Ausgang oder Eingang dienen sollen. Der *setup*-Teil wird zu Beginn einmal durchlaufen.

```
void setup() {
Serial.begin(9600);          //(1)
pinMode(6,OUTPUT);           //(2)
pinMode(7,INPUT);
pinMode(12,OUTPUT);
GSM.begin();                 //(3)
GSM.delAllSms();             //(4)
}
```

Abbildung 30: Void Setup

1. In der ersten Zeile des Setups wird mit einer Baudrate von 9600 die serielle Verbindung zum Computer hergestellt. Diese ist nicht wesentlich für das Programm, jedoch ist die Fehleranalyse und die Möglichkeit Tests durchzuführen damit deutlich einfacher.
2. Hier werden die Pins des Arduinos als Ausgang oder Eingang deklariert. Auf die einzelnen Pins wird bei Punkten 8.4.2 und 8.4.3 eingegangen.
3. Nun wird die Kommunikation zu dem GSM-Modul aufgebaut. Diese verläuft auch mit einer Baudrate von 9600, jedoch ist dies in der Library festgelegt.
4. Diese Funktion kommt noch sehr oft im Programm vor. Hier wird der Speicher des GSM-Moduls gelöscht. Das ist sehr wichtig, da immer die erste SMS gelesen wird. Würde diese Funktion nicht vorhanden sein, werden weitere Befehle, die als SMS ankommen, auch als zweite, dritte und vierte SMS gespeichert. So würden sie in den Hintergrund rücken und das Programm könnte diese Befehle nicht ausführen, da es diese Nachrichten gar nicht registriert.

8.3 Void Loop

Im *void loop* Teil stehen alle Funktionen, die den Arduino und die angeschlossenen Module aktiv steuern. Dieser Programmteil wird immer wiederholt.
Die folgenden Variablen werden für den ersten Teil des Loops benötigt. Sie stehen noch über dem *void setup*-Teil und sind somit als globale Variablen definiert.

```
String smslesen,handynummer;                    // (1)
char Handynummer1[20];                          // (2)
```

Abbildung 31: Globale Variablen für den Void Loop

1. In der ersten Zeile werden zwei Variablen der Art String benötigt. Diese Variable setzt sich aus Zeichenketten zusammen.
2. Hier wird ein Char beziehungsweise ein Char-Array (eine Sammlung von Variablen des Typ Char) mit Namen *„Handynummer1“* und der Zeichengröße 20 deklariert. In diesem wird später die Handynummer der empfangenen SMS umgewandelt, damit sie für die Antwort des GSM-Moduls verwendet werden kann. Die Größe einer Handynummer aus Deutschland beträgt mit der Vorwahl 16 Zeichen. Zusätzlich benötigt ein Array vom Typ Char ein weiteres Element zur Initialisierung. So ist die Größe ausreichend gewählt.

```
void loop() {

     smslesen=GSM.readSms(1);                                  //(3)
          if (smslesen.length() > 4)                           //(4)
               {
                timerGps=0;                                    //(5)
                handynummer=GSM.getNumberSms(1);               //(6)
                Serial.println(handynummer);                   //(7)
                smslesen.toUpperCase();                        //(8)
                handynummer.toCharArray(Handynummer1,20);      //(9)

                                                               //(10)
```

Abbildung 32: Void Loop

3. In der ersten Zeile des *Void Loops* wird immer die erste SMS im Speicher gelesen.
4. Mit der IF-Bedingung wird nun geprüft, ob die SMS größer als 4 Zeichen ist, womit verhindert wird, dass der Arduino auf eine leere SMS reagiert. Ist die Bedingung wahr, wird alles in den geschweiften Klammern ausgeführt.
5. Hier wird die Variable *„timerGps“* der GPS-Funktion zurückgesetzt. Darauf wird im Punkt 8.4.3 genauer eingegangen.
6. Im diesem Punkt wird die Handynummer der empfangenen SMS als String gespeichert.
7. Zur Übersicht wird die Nummer auf dem seriellen Monitor ausgegeben.
8. Bei der Funktion *„toUpperCase()“* wird die empfangene Nachricht in Großbuchstaben umgewandelt, somit ist später die Groß- und Kleinschreibung der Befehle unwichtig.
9. Wie bereits erwähnt, wird hier die Handynummer in die Variable Char-Array umgewandelt. Dies ist zwingend notwendig, da die Funktion, die für das Verschicken einer SMS zuständig ist, nur den Variablen-Typ Char erlaubt.
10. Nun ist die Hauptfunktion, in der die SMS gelesen wird, fertig. Darauf folgen Funktionen, in denen die Befehle der SMS gefiltert werden und auf die reagiert wird. Dies wird im Punkt 8.4 tiefgründiger erläutert. Im Anschluss der Funktionen ist es wichtig, wieder alle SMS vom Speicher zu löschen. Sollte sonst eine SMS ankommen, die nicht zu den gewählten Befehlen passt, darf sie nicht den Speicher blockieren und das Lesen der ersten SMS verhindern.

8.4 Funktionen im Void Loop

Um die gewünschten Funktionen einzuleiten, braucht es genaue Befehle, die in einer SMS als Wortlaut formuliert werden.

8.4.1 Akkustand abfragen

Damit der Akkustand einwandfrei ausgelesen werden kann, braucht es erneut globale Variablen. Diese stehen noch über dem Setup-Teil.

```
unsigned int Pinlesen;                  //(1)
char sms_Akku[20];                      //(2)
unsigned int umwandelProzent;
String TextAkku ="";                    //(3)
```

Abbildung 33: Globale Variablen Akkustand

1. In der ersten Zeile wird eine Variable vom Typ *„unsigned int“* benötigt. Da der Wert des Analogeingangs 0 bis 1023 einnehmen kann, ist dieser Typ der geeignetste. Er kann Ganzzahlen ohne Vorzeichen von 0 bis 65.535 verarbeiten.
2. Hier wird wieder ein Char-Array verwendet, der für den späteren Text, der gesendet wird, zuständig ist. Dieses kann bis zu 19 Zeichen einnehmen.
3. Die Zeichenkette *„TextAkku“* des Typs *string* setzt die genaue SMS dann zusammen und wird danach in ein Char-Array umgewandelt.

```
if (smslesen.indexOf("AKKUSTAND")!=-1){                    //(4)
GSM.delAllSms();                                           //(5)
analogReference(INTERNAL);                                 //(6)
Pinlesen=analogRead(A0);                                   //(7)
Serial.println(Pinlesen);
umwandelProzent= map(Pinlesen,907,1023,0,100);             //(8)
Serial.println(umwandelProzent);
TextAkku="Akkustand="+String(umwandelProzent)+"%";         //(9)
TextAkku.toCharArray(sms_Akku,20);                         //(10)
Serial.println(sms_Akku);
GSM.sendSms(Handynummer1,sms_Akku);                        //(11)
Serial.println("Akkustand geschickt");

}
```

Abbildung 34: Programm Akkustand

4. Zu Beginn wird mit einer *„If-Bedingung“* geprüft, ob die empfangene Nachricht „Akkustand“ lautet. Sollte dies der Fall sein, wird alles in den geschweiften Klammern ausgeführt. Dabei ist die Groß- und Kleinschreibung durch die Funktion *„toUpperCase()“* aus Punkt 8.3 egal. Wichtig ist jedoch, dass in dieser *„If-Bedingung*“ alles groß geschrieben wird, denn die angekommene SMS wird in Großbuchstaben umgewandelt und mit dieser Bedingung verglichen. Sollte hierbei auch nur ein Buchstabe klein sein, wird die Bedingung nie erfüllt.
5. Hier ist wichtig, gleich zu Beginn den Speicher wieder zu löschen. Sollte der Arduino mitten im Programm unterbrochen werden und es befindet sich noch eine SMS im Speicher des GSM-Moduls, würde er auf weitere Befehle gar nicht reagieren. Daher wird immer direkt nach dem erfolgreichen Erkennen des Kommandos der Speicher wieder geleert.
6. Bei dieser Funktion wird die Referenzspannung konfiguriert. Durch den Wortlaut *„INTERNAL“* wird bei dem Prozessor ATmega328P ein Analogwert von 1023 bei einer Spannung von 1,1 V an Pin A0 zugeordnet. Es wurde hardwaretechnisch also ein Spannungsteiler aufgebaut, sodass bei einem vollen Lithium-Akku genau 1,1 V an diesem Analog Pin anliegen. Im Punkt 10.3 wird genauer darauf eingegangen, wie der benötigte Spannungsteiler dimensioniert wird.
7. Nun wird der Analogwert ausgelesen. Dieser kann einen Wert zwischen 0 und 1023 haben.

8. Als nächstes wird aus diesem Analogwert ein Prozentwert. Dafür ist der Wert 907 der 0 und der Wert 1023 der 100 zugeordnet. Nach diesem Verhältnis wird ein Prozentwert zwischen 0 und 100 gebildet. Wie diese Zahlen ermittelt werden, steht im Punkt 10.4 *Analogwert genau errechnen* und 10.5 *Analogwert messen.*
9. Der Prozentwert wird hier mit dem Wort „Akkustand“ einem „=“ und dem Prozentzeichen in eine String-Kette verpackt. Das geschieht, damit der Empfänger der SMS weiß, um was für einen Prozentwert es sich hierbei handelt.
10. Hier wird der String wieder zu einem Char-Array verwandelt, damit er als SMS verschickt werden kann.
11. Jetzt wird die SMS verschickt. Sie beinhaltet den Akkustand in Prozent und geht an die Nummer, die den Akkustand abgefragt hat.

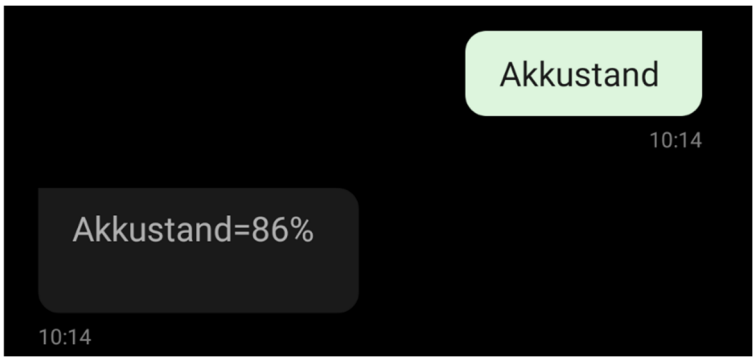


Abbildung 35: Akkustand per SMS abfragen

8.4.2 Alarm aktivieren und deaktivieren

Als Nächstes wird die Funktion in das Programm eingepflegt, welche den Bewegungsalarm des Fahrrads ein- oder ausschaltet.
Zuerst werden wieder die Variablen benötigt.

```
boolean ersteAusloesung;                    //(1)
boolean Alarm = 0;                          //(2)
unsigned long Alarmintervall= 1800000;      //(3)
unsigned long AlarmBisher = 0;              //(4)
```

Abbildung 36: Globale Variablen Alarm

1. Als Erstes wir eine Variable des Typs *boolean* benötigt. Diese wird gebraucht, um den Alarm erstmals auszulösen und zwar direkt, wenn das Fahrrad bewegt wird.
2. Diese Variable gibt im Programm eine 1 wieder, wenn der Alarm aktiviert wurde und eine 0, wenn er ausgeschalten ist.
3. Nun wird bestimmt, in welchem Zeitintervall der Alarm wiederholt ausgelöst wird. Dieser ist hier auf 1.800.000 Millisekunden gelegt, sodass bei Bewegung des Fahrrads eine Alarm-SMS gesendet wird und erst nach 30 Minuten eine weitere. So wird verhindert, dass mehrfach Nachrichten gesendet werden und das aufgeladene Geld auf der Prepaidkarte nicht sinnlos ausgegeben wird.
4. Dann wird noch eine Variable des Typs *unsigned long* gebraucht. Diese speichert den Zeitpunkt, an dem der Alarm schon einmal ausgelöst wurde.

```
if (smslesen.indexOf("ALARMAN")!=-1){                    //(5)
  GSM.delAllSms();
  Serial.println("Alarm ist an");
  digitalWrite(6,HIGH);                                  //(6)
  Alarm = 1;                                             //(7)

  }
if (smslesen.indexOf("ALARMAUS")!=-1){                   //(8)
  GSM.delAllSms();
  AlarmBisher = 0;                                       //(9)
  ersteAusloesung = 0;                                   //(10)
  Serial.println("Alarm ist aus");
  digitalWrite(6,LOW);                                   //(11)
  Alarm = 0;                                             //(12)
}
```

Abbildung 37: Programm Alarm

5. In diesem Punkt wird verglichen, ob die empfangene SMS mit dem Wortlaut *„AlarmAn“* übereinstimmt, Groß- und Kleinschreibung ist dabei wieder nicht zu beachten. Sollte dies der Fall sein, werden alle Befehle in den geschweiften Klammern ausgeführt.

6. Hier wird der Pin 6 des Arduino auf *High* geschalten, somit liegt am Operationsverstärker eine Spannung an und bei Bewegung gibt er ein *High* Signal auf Pin 7 des Arduinos zurück.
7. Im Punkt 7 ist softwaretechnisch abgesichert, dass der Alarm auch nur ausgelöst werden kann, wenn er wirklich angeschalten wurde. Das geschieht, indem die Variable Alarm des Typs *boolean* genutzt wird. Hier wird der Alarm angeschalten, dementsprechend wird sie auf 1 gesetzt.
8. Als Nächstes muss das Ganze noch andersherum gemacht werden. Passt die empfangene Nachricht mit dem Wortlaut *„AlarmAus“* überein, werden alle Befehle in den geschweiften Klammern ausgeführt.
9. Wenn der Alarm ausgeschalten wurde, wird die Variable *„AlarmBisher“* des Typs *unsigned long* zurück auf 0 gesetzt. Damit wird ein Überlaufen der Variablen verhindert. Mit Überlaufen ist gemeint, wenn der vorgesehene Speicher der Variable irgendwann voll ist, beginnt diese einfach wieder am Anfang (also bei 0). In diesem Fall ist die Variable für Millisekunden bestimmt, ihr Speicher wäre demzufolge nach 49,7 Tagen voll, wenn sie durchgängig zählen würde. In dieser Situation wäre der Fehler des Überlaufens jedoch gar nicht gravierend, es könnte unter Umständen zu 2 Meldungen beim Auslösen des Alarms innerhalb der 30 Minuten kommen. Jedoch ist das Zurücksetzen der Variable, wenn der Alarm ausgeschalten wird, eine Möglichkeit.
10. Hier wird die Variable *„ersteAusloesung“* zurück auf 0 gesetzt, damit der Alarm beim Aktivieren auch gleich wieder auf die erste Bewegung am Fahrrad reagieren kann.
11. Der Pin 6 wird auf Low geschalten und der Operationsverstärker bekommt keine Spannung mehr.
12. Nun wird die Variable Alarm auf 0 gesetzt, da der Alarm ausgeschalten wurde.

Des Weiteren muss im Programm festgesetzt werden, was passiert, wenn der Alarm ausgelöst wurde und an Pin 7 des Arduino ein *High* Signal anliegt.

```
if(digitalRead(7)==HIGH && Alarm ==1 ){                                    //(13)
unsigned long AlarmMillis= millis();                                       //(14)

if(digitalRead(7)==HIGH && Alarm ==1 && ersteAusloesung ==0){              //(15)
ersteAusloesung=1;                                                         //(16)
AlarmBisher=AlarmMillis;                                                   //(17)
GSM.sendSms(Handynummer1,"Alarm,Fahrrad wird bewegt!");                    //(18)
Serial.println("Alarm ausgelöst");
}
                                                                           //↓(19)
while(digitalRead(7)==HIGH && Alarm ==1 && AlarmMillis - AlarmBisher >= Alarmintervall){
AlarmBisher=AlarmMillis;                                                   //(20)
unsigned long AlarmMillis= millis();                                       //(21)
GSM.sendSms(Handynummer1,"Alarm,Fahrrad wird bewegt!");
Serial.println("Alarm ausgelöst");
  }
  }
```

Abbildung 38: Programm Alarm ausgelöst

13. Als Erstes wird mit einer „If-Bedingung“ geprüft, ob der Pin 7 High ist und die Variable Alarm auf 1 steht, also der Alarm an ist. Ist das der Fall, wird alles in den geschweiften Klammern ausgeführt.
14. Nun wird mit der Funktion *„millis()“* die Zeit in Millisekunden ermittelt, seitdem das Programm gestartet ist.
15. Mithilfe einer weiteren If-Bedingung wird wieder geprüft, ob Pin 7 und die Variable Alarm auf 1 stehen. Des Weiteren wird geprüft, ob es schon eine erste Auslösung gab. Wenn nicht, wird alles in den geschweiften Klammern ausgeführt
16. Die Variable *„ersteAusloesung“* wird nun auf 1 gesetzt, so wird verhindert dass der Alarm die ganze Zeit ausgelöst wird.
17. Als Nächstes wird in der Variable *„AlarmBisher“* gespeichert, zu welcher Zeit die Auslösung stattgefunden hat.
18. Hier wird per SMS geschrieben, dass der Alarm ausgelöst wurde und dass das Fahrrad bewegt wird.

19. In diesem Punkt wird eine *while*-Schleife benutzt, die solange läuft, bis die Bedingung in den Klammern falsch wird. Diesmal wird wieder geprüft, ob Pin 7 High und der Alarm an ist, jedoch wird jetzt auch die Zeit verglichen. Und zwar wird die Zeit der letzten Auslösung, also *„AlarmBisher“,* von der Zeit, seitdem das Programm läuft, *„Alarmmillis“* abgezogen. Ist dieses Ergebnis größer oder gleich des festgelegten Alarmintervalls von 30 Minuten, werden alle Befehle in den geschweiften Klammern ausgelöst.
20. Nun wird die Zeit der Auslösung wieder in der Variablen *„AlarmBisher“* gespeichert.
21. Wichtig ist die aktuelle Zeit wieder in *„AlarmMillis“* zu speichern.

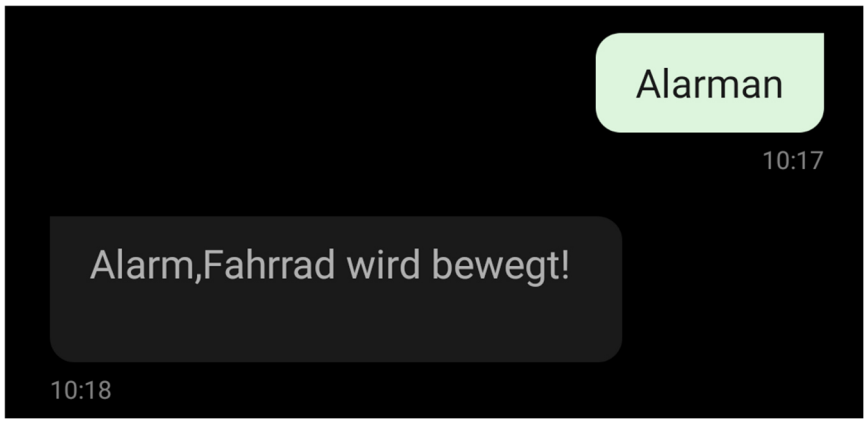


Abbildung 39: Alarm per SMS aktivieren

8.4.3 GPS Daten abfragen

Um die GPS-Koordinaten ordentlich abfragen zu können, werden folgende globale Variablen benötigt.

```
char sms_Gps[160];                    //(1)
float WertLaenge;                     //(2)
float WertBreite;
float WertHoehe;
String TextGps ="";                   //(3)
boolean timerGps = 0;                 //(4)
unsigned long timerEmpfang= 7200000;  //(5)
```

Abbildung 40: Globale Variablen GPS-Daten

1. Als Erstes wird ein Char benötigt, in dem später der Google Maps-Link inklusive der ermittelten Höhe verpackt wird.

2. Nun werden drei Variablen des Typs *float* gebraucht. Dies ist wichtig, da hier eine Kommazahl ermittelt wird. Der Längen- und Breitengrad sollte mindestens bis zur sechsten Kommazahl gehen, um den Standort optimal und genau anzeigen zu können.
3. Die Variable *„TextGps“* des Typs String wird verwendet, um die ermittelten Werte in einem Text zu formulieren, der dann einen Google Maps Link bildet.
4. Danach wird eine einfache Variable benötigt, die verhindert, dass die Koordinaten mehrfach ermittelt werden. Dies wird mit einer Variablen des Typs *boolean* erreicht.
5. Als Letztes sollte noch eine Zeit festgelegt werden, die verstreichen kann, bis das GPS-Modul die Koordinaten ermittelt hat. Sollte diese über die hier festgelegten zwei Stunden gehen, wird der Vorgang abgebrochen.

```
if (smslesen.indexOf("GPSDATEN")!=-1){              //(6)
   GSM.delAllSms();
    Serial.println("GPS Ermittlung eingeleitet");

  while(timerGps==0){                               //(7)

   digitalWrite(12,HIGH);                           //(8)
   GetGps();                                        //(9)

    }
    }
```

Abbildung 41: Programm GPS Daten abfragen

6. Zunächst wird die angekommene SMS überprüft, ob sie mit *„GPSDaten“* übereinstimmt. Ist das der Fall, werden alle Befehle in den geschweiften Klammern ausgeführt.
7. Hier wird eine *while*-Schleife dazu benutzt, alle Befehle in der geschweiften Klammer solange auszuführen, bis die Bedingung falsch wird. Die Bedingung kann nur falsch werden, wenn der *„timerGps“* wieder auf 1 gesetzt wird. Dies geschieht nur, wenn entweder ein GPS-Signal gefunden wurde oder die Zeit für den Empfang von zwei Stunden vorüber ist.

8. Wurde die Ermittlung der GPS-Koordinaten eingeleitet, wird der Pin 12 des Arduinos High gesetzt, somit wird das GPS Modul mit Spannung versorgt und beginnt die Ermittlung des Standortes.
9. Nun wird auf eine weitere *void*-Funktion namens *„GetGps()“* Bezug genommen. Damit wird diese ausgeführt.

Diese Funktion steht unter dem *void loop*, dies wurde aus übersichtlichen Gründen ausgewählt. Sie könnte auch komplett dort stehen, wo auf sie verwiesen und sie ausgeführt wird.

```
void GetGps() {
 unsigned long currentMillis= millis();                                    //(10)
 ss.begin(9600);                                                           //(11)
  while (ss.available() > 0)                                               //(12)
  gps.encode(ss.read());
  if (gps.altitude.isUpdated()){                                           //(13)
Serial.print("Breitengrad="); Serial.println(gps.location.lat(), 6);
Serial.print("Längengrad="); Serial.println(gps.location.lng(), 6);
Serial.print("Höhe="); Serial.println(gps.altitude.meters(), 6);
WertLaenge=(gps.location.lng());                                           //(14)
WertBreite=(gps.location.lat());
WertHoehe=(gps.altitude.meters());
TextGps = "Link: https://www.google.com/maps?q="+String(WertBreite,6)+","//(15)
+String(WertLaenge,6)+"height:"+String(WertHoehe,6);
TextGps.toCharArray(sms_Gps,160);                                          //(16)
GSM.sendSms(Handynummer1,sms_Gps);
digitalWrite(12,LOW);                                                      //(17)
timerGps=1;                                                                //(18)
GSM.delAllSms();
}
```

Abbildung 42: Programm GPS Daten ermitteln

10. Als Erstes wird die Zeit genommen, bei der die GPS-Ermittlung eingeleitet wurde.
11. Nun wird die Verbindung zum GPS-Modul mit einer Baudrate von 9600 aufgebaut.
12. Mit einer *while*-Schleife wird das GPS-Modul solange gelesen, bis es einen Standort ermitteln konnte.

13. Ist der Standort erfolgreich ermittelt worden, wird alles in der geschweiften Klammer nach der „If Bedingung“ ausgeführt.
14. Die *float* Variablen werden den richtigen Funktionen zugeordnet, um einen Längengrad, einen Breitengrad und die Höhe zu ermitteln.
15. Diese Variablen werden in einen Text als Zeichenkette des Typs *string* verpackt. Dieser Text wird so geschrieben, dass die SMS automatisch erkennt, wenn es sich um einen Link handelt.
16. Nun wird diese Zeichenkette zu einem Char-Array umgewandelt, damit sie auch als SMS verschickt werden kann. Es wurde eine Zeichenkapazität von 159 Zeichen gewählt, somit ist ausreichend Platz für alle ermittelten Werte, den Link und die Leerzeichen.
17. Mit diesem Befehl wird die Spannung am Ausgang Pin 12 des Arduino Low und damit auch das GPS-Modul ausgeschalten.
18. Zuletzt wird noch die Variable „*timerGps*“ auf 1 gesetzt, so wird die *while*-Schleife aus Punkt (7) beendet und die GPS-Ermittlung wird unterbrochen.

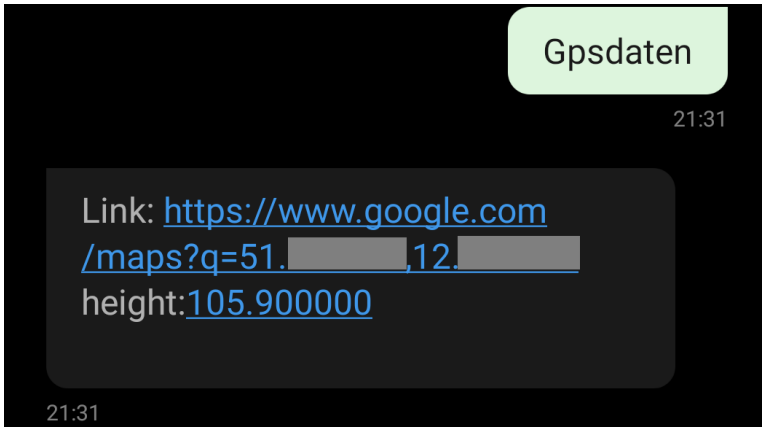


Abbildung 43: Abfragen des GPS-Standortes

Im Anschluss wird noch eine Funktion benötigt, die auftritt, wenn kein GPS-Signal gefunden wurde.

```
else if (currentMillis >= timerEmpfang){                    //(19)
GSM.sendSms(Handynummer1,"Kein GPS Empfang");               //(20)
timerGps=1;                                                 //(21)
digitalWrite(12,LOW);                                       //(22)
GSM.delAllSms();
}
```

Abbildung 44: Programm kein GPS Empfang

19. Als Erstes wird in der „*else if*-Schleife“ getestet, ob die Zeit, die am Anfang der GPS-Ermittlung genommen wurde, größer gleich der bestimmten Zeit von zwei Stunden ist. Diese Schleife folgt nach dem *if,* welches überprüft, ob die GPS-Daten ermittelt wurden.
20. Ist dies der Fall, wird eine SMS gesendet, die mitteilt, dass das GPS-Modul keinen Standort ermitteln konnte.
21. Die Variable *„timerGps“* wird auf 1 geschalten und so wird auch hier die *while*-Schleife aus Punkt (7) beendet.
22. Als Letztes wird der Ausgang des Arduinos Pin 12 auf *low* geschalten und so wird das GPS-Modul ausgeschalten.

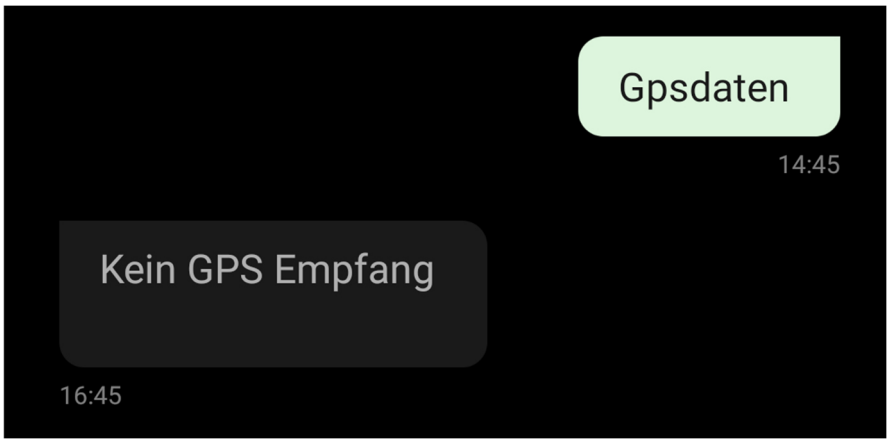


Abbildung 45: Kein GPS Empfang

9 Montage am Fahrrad

Die Montage am Fahrrad ist ein wichtiger Punkt des Projekts. Hierbei entscheidet sich, wie auffällig der GPS-Tracker ist und ob er dadurch leicht von einem möglichen Dieb entfernt werden kann. Das Ziel ist es also, den Tracker möglichst so zu positionieren, dass man von außen keine Veränderung an dem Fahrrad bemerkt. Durch das Einhalten des Aspektes der Dimensionierung fand der Tracker mit all seinen Bauteilen in der Sattelstange und deren Einschub Platz. So kann er problemlos in jedem Fahrrad verbaut werden und es ändert sich nichts am äußeren Erscheinungsbild des Fahrrades.

9.1 Das damit verbundene Problem des Empfanges

Ein wesentliches Problem, was sich damit auftat, war der fehlende Empfang von GSM- und GPS Modul. Dadurch, dass die Module mit in dem Fahrradrahmen sitzen und sich in einem geschlossenen Metallgehäuse befinden, ist das Einwählen in das GSM-Netz und das Finden von Satellitensignalen annähernd unmöglich. Behoben wurde das Problem mit dem Nach-außen-legen der jeweiligen Antennen. Dazu wurde ein Loch durch die Sattelstange gebohrt und die GSM- und GPS-Antenne hindurchgesteckt. Nun wurden die beiden Antennen unter den Sattel geklebt. Leider wurde damit nun doch etwas am äußeren Erscheinungsbild des Fahrrads geändert, jedoch bleibt der Tracker trotzdem gut versteckt und hat somit eine gute Empfangsqualität.

Abbildung 46: Antenne unter dem Sattel

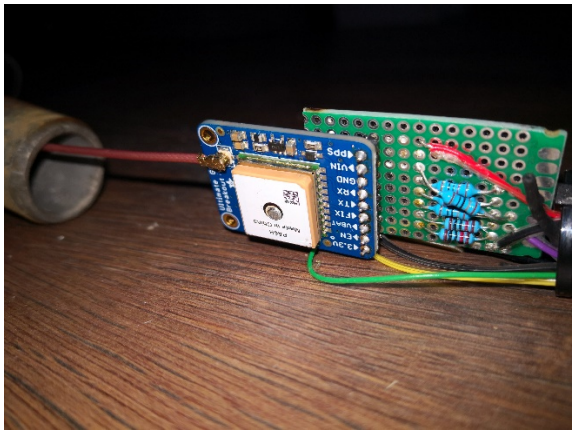


Abbildung 47: Anschluss der GPS Antenne

10 Akkuladezustand ermitteln

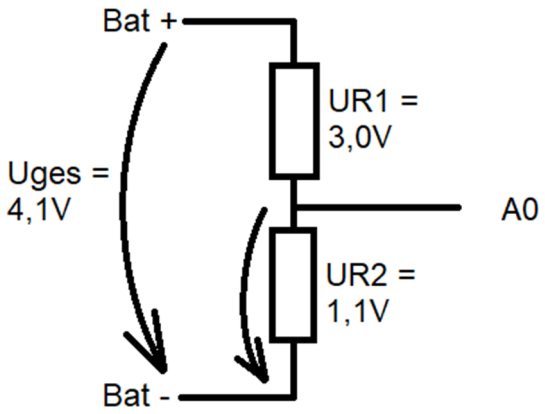
10.1 Spannungswerte

Das Ziel war es, mithilfe des Arduinos einen ungefähren Akkustand in Prozent zu erfassen, sobald dies per SMS gefordert wird. Der verwendete Akku besteht aus vier Samsung INR18650 2900mAh 3,7V Lithium Akkus. Diese wurden parallel geschalten und mit Isolierband umwickelt. Wichtig ist, dass die verwendeten Akkus nicht alleine betrieben werden, sondern zusammen mit einem Akkulademodul. Dieses schützt den Akku vor Überladung und Tiefentladung, welches beides zur Zerstörung des Akkus führen könnte. Nun werden die Eckdaten des Akkus benötigt, die wichtig für die Akkuberechnung sind. Zum einen die Spannung, welche herrscht, wenn der Akku voll aufgeladen ist und das GSM-Modul und den Arduino betreibt. Diese bezeichnet den Normalzustand, wie die Schaltung dann auch im Fahrrad betrieben wird. Und zum anderen die Spannung, bei der das erste Modul in der Schaltung nicht mehr betrieben werden kann. Dabei ist es wichtig, nicht den Spannungswert zu nehmen, bei dem der Akku leer ist beziehungsweise das Lademodul die Akkuentladung abschaltet. Sondern den Spannungswert zu nutzen, bei dem noch alle Module im ordnungsgemäßen Spannungsbereich arbeiten können.
Spannung des Akkus vollgeladen: 4,077 V
Spannung des Akkus leer (mit Schutz vor Tiefenentladung): 2,895 V
Spannungsbereich des GSM-Modul Sim800l laut Datenblatt: 3,5 – 4,4 V
Dadurch, dass der Arduino Nano von einem Boost Converter gespeist wird, ist das GSM-Modul das Bauteil, welches den höchstgelegenen Spannungsbereich hat.
Damit ist jetzt der Spannungswert fest, bei dem die Akkuanzeige null Prozent anzeigen sollte. Jedoch wurde sich dafür entschieden, einen kleinen Puffer von 0,1 V einzubeziehen, da mehrere Bauteile betrieben werden und man bei der Messung nicht von einer großen Genauigkeit ausgehen kann. Somit ist dem Spannungswert 3,6 V einen Akkustand von null Prozent zuzuordnen.

10.2 Analogwerte

Nun wurde überlegt, wie diese Spannungswerte am besten Analogwerte zugeordnet werden können. Mit der Funktion *„analogReference()“* kann die Referenzspannung der Analogeingänge des Arduinos konfiguriert werden. Wird die Analogreferenz auf *„Internal“* gestellt, so ist bei einer Spannung von 1,1 V der Analogwert der höchste und zwar 1023. Nun muss ein Spannungsteiler so dimensioniert werden, dass bei einer Gesamtspannung von 4,1V an einem der Widerstände 1,1V anliegen. Diese werden dann an den Analogeingang A0 geschalten.

10.3 Dimensionierung des Spannungsteilers



Zunächst musste ein Widerstand für R2 festgelegt werden. Dieser wird auf 20 kΩ dimensioniert. Somit fließt ein geringer Strom und die spätere Zusammensetzung des R1 fällt wesentlich leichter.

Abbildung 48: Spannungsteiler

Wird die Formel für den Spannungsteiler nach R1 umgestellt, lautet sie wie folgt:

$$R1 = \left(\frac{U_{ges}}{U_{R2}} - 1\right) * R2$$

Formel 1: Spannungsteiler

Nun werden die Spannungswerte und der Widerstand R2 in die Formel eingesetzt:

$$R1 = \left(\frac{4{,}1V}{1{,}1V} - 1\right) * 20k\Omega$$

Formel 2: Spannungsteiler eingesetzt

Laut der Rechnung braucht es somit einen Widerstand für R1 von 54545 Ω.
Um diesen Widerstandswert zu erhalten, reicht für diese Schaltung eine Parallelschaltung von 100kΩ und 120kΩ. Dabei wird ein Widerstand von 54500Ω gebildet. Nun sind alle Eckdaten des Spannungsteilers vorhanden und es können genaue Analogwerte der Schaltung ermittelt werden.

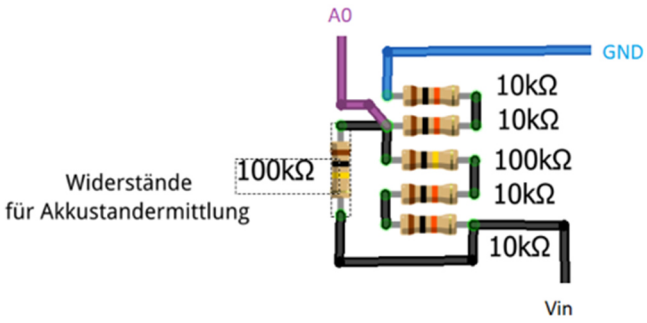


Abbildung 49: Spannungsteiler Schaltplan

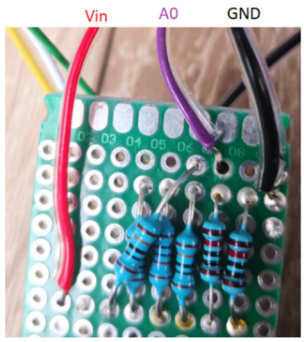


Abbildung 50: Spannungsteiler Bild

10.4 Analogwerte genau errechnen

Als nächstes wird ein genauer Analogwert benötigt, der bei 3,6V vom Arduino zugeordnet wird. Wird rein theoretisch an die Sache gegangen, könnte man über die Spannungsteiler-Formel zunächst ausrechnen, was am R2 anliegt.

$$U_{R2} = \frac{20k\Omega}{20k\Omega + 54{,}5k\Omega} * 3{,}6V$$

Formel 3: Spannungsteiler Akkuleer

Somit ist die Spannung für U_{R2} =0,972 V.
Um nun einen Analogwert zu bestimmen, kann man über den Dreisatz den Wert ausrechnen.

$$\text{Analogwert}_{\text{Akku leer}} = \frac{1023}{1{,}1V} * 0{,}972V$$

Formel 4: Dreisatz Analogwert

Damit sollte in der Theorie der Analogwert bei einer Gesamtspannung von 3,6V 904 betragen.

Jedoch müsste der Analogwert auch in der Praxis gemessen werden, da die Widerstände eine gewisse Abweichung haben und gemessen werden sollte, während das GSM-Modul und der Arduino betrieben werden.

10.5 Analogwert messen

Um einen Analogwert zu messen, wurde ein extra Arduino-Programm geschrieben.

```
unsigned int Pinlesen;
unsigned int umwandeln;
void setup() {
Serial.begin(9600);
}
void loop() {
  analogReference(INTERNAL);
  Pinlesen=analogRead(A0);
  Serial.println(Pinlesen);
  umwandeln= map(Pinlesen,904,1023,0,100);

  Serial.println(umwandeln);
  delay(2000);

}
```

Abbildung 51: Programm Analogwert messen

Wichtig ist es, den Akku genau auf 3,6 V zu laden oder zu entladen. Dann wird er mit der vorhandenen Schaltung verbunden und das Programm aufgespielt. Jetzt ist besonders auf die Variable *„Pinlesen“* zu achten, denn dies ist unser Analogwert, auf den es später ankommt. Nun wird der serielle Monitor geöffnet:

```
18:52:40.209 -> 907 ← Analogwert
18:52:40.209 -> 2
```

Abbildung 52: Serieller Monitor Analogwert

Der gemessene Analogwert ist beinahe genauso groß, wie unser errechneter Wert. Dieser Wert kann nun im Programm dafür genutzt werden, um unseren Akkuladezustand in Prozent anzugeben. Wie das im Programm realisiert wird, ist im Punkt 8.4.1 *Akkustand abfragen* beschrieben.

10.6 Auftretende Ungenauigkeit

Die auftretende Ungenauigkeit sollte nicht außer Acht gelassen werden. Diese hängt zum einen damit zusammen, dass die Akkuspannung nicht linear zum Ladezustand einhergeht. Ein weiterer Grund dafür ist die Benutzung eines Boost Converters. Bei sinkender Spannung zieht der Converter immer größere Ströme, da die Leistung gleich bleibt. Dazu sinkt auch der Wirkungsgrad des Converters. Somit entleert sich der Akku zum Ende hin immer schneller. Daher sollte bei dieser Akkuberechnung nur von Richtwerten ausgegangen werden.

11 Auftretende Probleme

11.1 Stromverbrauch

Nach der Fertigstellung und Testung der vollen Funktionen des GPS-Trackers erfolgte die Beschäftigung mit dem Stromverbrauch der Gesamtschaltung.

11.1.1 Messen des Stromverbrauchs

Beim alleinigen Messen des Stromverbrauches gab es schon ein Problem. Gemessen werden sollte der Verbrauch im Normalbetrieb der Schaltung, also beim Betreiben des Arduinos und des GSM-Moduls. Dabei stellte sich heraus, dass sich beim alleinigen Dazwischenschalten eines Amperemeters das GSM-Modul nicht mehr ins Netz einwählen ließ. Das Problem tritt dadurch auf, dass das Modul beim Einwählen und Senden kurzzeitig zwei Ampere zieht. Dadurch fällt an dem niederohmigen Innenwiderstand des Amperemeters so viel Spannung ab, dass das GSM-Modul kurzzeitig mit zu geringer Spannung versorgt wird. Ein ideales Amperemeter besitzt einen Innenwiderstand von 0 Ω. Ein reales Amperemeter hat jedoch immer einen geringen Widerstand. Für eine Beispielrechnung wurden 0,5 Ω angenommen. Wird das Amperemeter jetzt in Reihe zu unserm GSM-Modul geschalten und mit der vollgeladenen Akkuspannung von 4,1 V versorgt, ergibt sich folgender Spannungsabfall.

Spannungsabfall Innenwiderstand Amperemeter = $0{,}5\ \Omega * 2\ A = 1V$
Spannungsabfall von der Gesamtspannung abgezogen= $4{,}1V - 1V = 3{,}1V$
Formel 5: Spannungsfall Amperemeter

3,1 V liegen an dem GSM-Modul an, was zur Folge hat, dass es kurzzeitig unter der mindestens geforderten Spannung von 3,5 V liegt und sich so nicht einwählen kann. Das Beispiel zeigt, dass schon eine zu lange Anschlussleitungen und schlechte Übergangswiderstände zu Problemen des GSM-Moduls führen können. Daher ist wichtig, dass das Modul so nah wie möglich an der Versorgungsspannung platziert wird und auch kalte Lötstellen zu vermeiden sind. Um den Stromverbrauch der

Schaltung zu messen, wird eine Strommessung benötigt, die nicht den Widerstand verschlechtert.
Dazu wurde das Messgerät Fluke 773 mit Stromzange verwendet. Dieses misst Ströme bis 100mA mit einer Genauigkeit von 0,2 % ohne Unterbrechung des Stromkreises oder Beeinflussung des Widerstandes.

Abbildung 53: Messgerät Fluke 773

11.1.2 Stromverbrauch und Akkulaufzeit der Schaltung

Durch das Parallelschalten von vier 2900mAh Lithium Akkus wurde eine Kapazität von insgesamt 11600mAh erreicht. Diese Kapazität kann jedoch nicht voll ausgeschöpft werden. Durch eine Schutzschaltung wird vor Tiefenentladung geschützt und durch den Spannungsbereich des GSM-Moduls von mindestens 3,5 V kann nur die Kapazität bis zu dieser Entladespannung genutzt werden. Im Internet wurde folgendes Diagramm zur Entladekurve von Lithiumakkumulatoren gefunden:

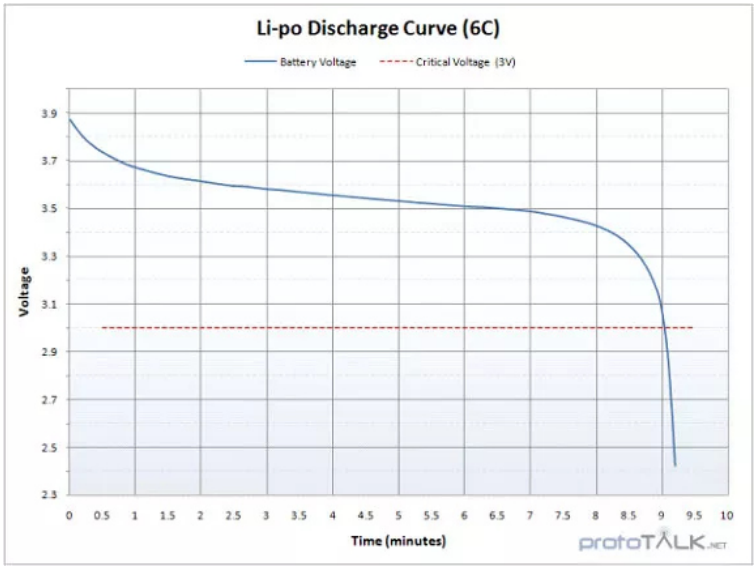


Abbildung 54: Entladekurve Lithium Akku
https://www.exp-tech.de

Dass der Akku in diesem Diagramm innerhalb von zehn Minuten entladen wird, liegt an dem hohen C-Koeffizienten von 6.
In diesem Fall braucht es 6 Minuten und 30 Sekunden, bis die Spannung auf 3,5 V abfällt. Dann würde es weitere 3 Minuten und 30 Sekunden dauern, bis der Akku komplett leer ist. Um nun auf die Kapazität zu schließen, die der Akku bei 3,5 V besitzt, wird die Zeit ab 3,5 V bis 0 V durch die Gesamtzeit geteilt.

Kapazität ab 3,5V = $\frac{2{,}5min}{10min} * 100 = 25\%$

Formel 6: Verlorene Akkulaufzeit

Somit sind 25% der 11600mAh für das Projekt nicht nutzbar.

Nutzbare Kapazität= $11600mAh * 0{,}75 = 8700mAh$

Formel 7: Akkulaufzeit

Durch den gemessenen Stromverbrauch im Normalbetrieb kann nun errechnet werden, wie lange der Akku hält.

Abbildung 55: Stromverbrauch im Normalbetrieb

Akkulaufzeit :

$$\frac{8700mAh}{35{,}1ma} = 247{,}86\ h$$

Formel 8: Berechnung Akkulaufzeit

Somit läge die Akkulaufzeit im Normalbetrieb bei circa 10 Tagen. Jedoch wird für jede geschickte SMS und Ermittlung des GPS-Standortes deutlich mehr Strom benötigt, sodass bei häufiger Benutzung die Akkulaufzeit sinkt.

11.1.3 Mögliche Maßnahmen zu Senkung des Stromverbrauchs

1. Eine Möglichkeit, um den Stromverbrauch deutlich zu senken, wäre die Verwendung eines Arduino Pro Minis, der Variante 8Mhz 3,3V. Dieser kann direkt an dem Lithium-Akku gespeist werden und benötigt so keinen Boost Converter. Der Converter hat einen Energieverlust von 7%, die einem so erhalten bleiben.
2. Des Weiteren wäre es möglich, das GSM-Modul in den Sleep-Mode zu versetzen. Im Punkt 11.2 wird näher auf diesen Versuch eingegangen.
3. Nach dem Sicherstellen der vollen Funktion der Schaltung wäre eine weitere Möglichkeit das Herauslöten der Leuchtdioden, die sich auf den Modulen befinden und Auskunft über das Einwählen in das GSM-Netz oder den Empfang von GPS-Signalen geben.
4. Durch die Anschlussmöglichkeit eines Fahrraddynamos kann beim Fahren der Akku wieder aufgeladen werden. Je nach Länge der Strecke und der Benutzungsdauer kann die Akkulaufzeit verlängert werden oder sogar dauerhaft bestehen bleiben.

11.2 Sleep-Mode Sim800l GSM-Modul

Ein weiteres Problem war es, das GSM-Modul wieder aus dem Sleep-Mode zu bekommen. Mithilfe von AT-Befehlen wurde eine Funktion für den Sleep-Mode 2 der vorhandenen Bibliothek hinzugefügt. Das Sim800l Modul besitzt zwei Sleep-Modes. Sleep-Mode 1: Hier wird das Modul nur geweckt, wenn der DTR-Pin des Moduls ein High Pegel bekommt. Beim Sleep-Mode 2 wird das Modul beim Erhalten einer SMS geweckt. Ziel war es, das Modul in den Sleep-Mode 2 zu versetzen und mit einer SMS zu wecken, welche den Befehl, den Sleep-Mode zu verlassen, beinhaltet. Beim Erhalten dieser SMS ist das SIM800l Modul auch kurz „wach“ geworden, jedoch reichte die Zeit für den Arduino nicht aus, den Befehl zu lesen, bevor das Modul wieder den Strombedarf auf 2mA runterschraubte.
Nur das Verschicken mehrere SMS hintereinander brachten den gewünschten Erfolg. Leider war das nicht im Sinne des Projektes.

12 Kosten

Bauteil	EP	GP (Anzahl)
Arduino Nano	20,00€	-
MOSFET- Transistor	1,50€	10,99€ (10)
GSM Modul (SIM800L)	7,49€	14,98€ (2)
GPS Modul (Neo-6M)	4,99€	-
GPS Modul (MTK3339)	44,58€	-
ELEEGOO Nano V3	3,33€	9,99€ (3)
Akkupack 18650	3.98€	15,08€ (4)
Akkupack	20,49€	-
OPV	2,05€	7,26€ (5)
Platinen	10,69€	-
Breadboards	8,99€	-
Lademodul (DEBO2)	1,37€	-
Boost Converter Modul MT3608	1,26€	6,29€
Keramikantenne GPS	9,99€	-
Gesamt:	=185,69€	
im Projekt verbaut:	=98,93€	

Tabelle 6: Kosten

13 Fazit

Nach vorangegangenen Ausführungen ist es gelungen, einen voll funktionsfähigen GPS-Tracker zusammenzustellen und zu programmieren. Deutlich ausbaufähig wäre der Stromverbrauch des Gerätes. Hier sollten noch Optionen in Erwägung gezogen werden, die diesen auf ein Minimum senken. Des Weiteren wäre die Erstellung einer Leiterplatine sinnvoll. So könnte Platz gespart werden und die unschönen Kabel würden verschwinden. Um eine genauere Standortbestimmung zu erreichen und auch Bereiche abzudecken, in denen GPS Empfang nicht möglich ist, könnte eine Standortermittlung über das GSM-Netz sinnvoll sein. So wäre eine Positionsbestimmung in Kellern und Garagen kein Problem mehr. Mit dem Einhalten dieser Verbesserungsaspekte und einem passenden Gehäuse, wäre dieser GPS-Tracker eine gute Wahl für Fahrradbesitzer aller Art. Erweiterbar wäre dieses Produkt für jegliche andere Verkehrsmittel, die es zu schützen lohnt. Bei motorisierten Fahrzeugen könnte dabei auf eine zusätzliche Akkumulator verzichtet werden und das Produkt könnte nur über die eingebaute Batterie betrieben werden. Im Vergleich zu den meisten anderen GPS-Trackern auf dem Markt, besitzt das beschriebene Projekt den Vorteil, dass es über SMS angesteuert wird und keine laufenden Kosten durch eine zahlungspflichtige Ortungs-App verursacht. Letztlich kann resümiert werden, dass dieses Projekt ein voller Erfolg war und hat den Verfassern viel über die Programmierung eines Arduinos, das Einbinden von GPS- und GSM Modulen und die Gesamtheit einer Produktenwicklung näher gelegt hat.

14 Literaturverzeichnis

Bastelgarage.ch. 2021. TP4056 Lithium LiPo Akku Batterie Lademodul Micro USB 5V 1A. https://www.bastelgarage.ch/tp4056-lithium-lipo-akku-batterie-lademodul-micro-usb-5v-1a. Zugegriffen: 14. Februar 2021.

Bernhard Grotz. 2018. Der Einweg-Gleichrichter. https://www.grund-wissen.de/elektronik/schaltungen/gleichrichter-und-wechselrichter.html. Zugegriffen: 14. Februar 2021.

chemie-schule.de. 2021. Piezoelement. https://www.chemie-schule.de/KnowHow/Piezoelement. Zugegriffen: 14. Februar 2021.

Chr. Caspari. 2018. Akkumulatoren. http://www.elektronikinfo.de/strom/akkus.htm. Zugegriffen: 14. Februar 2021.

Corinne Meunier Jonas Stoll Laura Schoen. 2021. Radverkehr. Vorteile des Fahrradfahrens. https://www.umweltbundesamt.de/themen/verkehr-laerm/nachhaltige-mobilitaet/radverkehr#vorteile-des-fahrradfahrens. Zugegriffen: 14. Februar 2021.

Danny Schreiter. 2019. *Arduino Kompendium. Elektronik,Programmierung und Projekte*. Landshut: bmu-verlag.

Duden Learnattack GmbH. 2021. Kondensatoren. https://www.lernhelfer.de/schuelerlexikon/physik-abitur/artikel/kondensatoren#. Zugegriffen: 14. Februar 2021.

Electronics-tutorials. 2019. Kondensator-Typen. https://www.electronics-tutorials.ws/de/kondensatoren/kondensator-typen.html. Zugegriffen: 14. Februar 2021.

Electronics-tutorials. 2020. Der MOSFET. https://www.electronics-tutorials.ws/de/transistoren/mosfet.html.

Elektronik Kompendium. 2021. Integrierte Festspannungsregler (78xx/79xx). https://www.elektronik-kompendium.de/sites/bau/0204301.htm. Zugegriffen: 14. Februar 2021.

Hannes Ühss. 2020. GSM Ein aussterbender Mobilfunkstandard? https://www.wlansignalverstaerken.de/gsm. Zugegriffen: 14. Februar 2021.

Kirstin Welther. 2020. Flottenüberwachung: Wie funktioniert die GPS-Ortung. https://www.verizonconnect.com/de/ressourcen/artikel/wie-funktioniert-gps-ortung/. Zugegriffen: 14. Februar 2021.

MDR Sachsen. 2020. Leipzig ist Deutschlands Hauptstadt der Fahrraddiebe. https://www.mdr.de/sachsen/leipzig/leipzig-leipzig-land/fahrrad-diebstahl-hoch-burg-klau-100.html. Zugegriffen: 14. Februar 2021.

Mirko Wenig. 2020. Fahrraddiebstahl - 2019 wurden 155.000 versicherte Räder gestohlen. https://www.versicherungsbote.de/id/4892876/Fahrraddiebstahl-2019-versicherte-Raeder-gestohlen/. Zugegriffen: 14. Februar 2021.

SprinterSB BMS. 2015. Lineare Spannungsregler. https://rn-wissen.de/wiki/index.php/Spannungsregler. Zugegriffen: 14. Februar 2021.

Thomas Schaerer. 2021. Operationsverstärker (OP / OV / OPV / OpAmp). https://www.elektronik-kompendium.de/sites/bau/0209092.htm. Zugegriffen: 14. Februar 2021.

15 Anlagen

A1 - Selbstständigkeitserklärung

A2 - Programm

A1
Selbstständigkeitserklärung

Wir versichern wahrheitsgemäß, dass wir die vorliegende Arbeit selbstständig verfasst, alle benutzten Hilfsmittel vollständig und genau angegeben und alles kenntlich gemacht haben, was aus Arbeiten anderer unverändert oder mit Abänderungen entnommen wurde.

Leipzig, den 15.02.2021

.....

Unterschrift

A2

Programm

```
#include <Sim800l.h>
#include <SoftwareSerial.h>
#include <TinyGPS++.h>
Sim800l GSM;
SoftwareSerial SerialAT (10, 11);
TinyGPSPlus gps;
SoftwareSerial ss(4, 3);

String smslesen,handynummer;
char Handynummer1[20];

char sms_Gps[160];
float WertLaenge;
float WertBreite;
float WertHoehe;
String TextGps ="";
boolean timerGps = 0;
unsigned long timerEmpfang= 7200000;

unsigned int Pinlesen;
char sms_Akku[20];
unsigned int umwandelProzent;
String TextAkku ="";

boolean ersteAusloesung;
boolean Alarm = 0;
unsigned long Alarmintervall= 1800000;
unsigned long AlarmBisher = 0;


void setup() {
Serial.begin(9600);
pinMode(6,OUTPUT);
pinMode(7,INPUT);
pinMode(12,OUTPUT);
GSM.begin();
GSM.delAllSms();
}


void loop() {
     Serial.println("Hauptschleife");
     smslesen=GSM.readSms(1);
          if (smslesen.length() > 4)
               {
                timerGps=0;
                handynummer=GSM.getNumberSms(1);
                Serial.println(handynummer);
                smslesen.toUpperCase();
                handynummer.toCharArray(Handynummer1,20);
```

```
                if (smslesen.indexOf("AKKUSTAND")!=-1){
                GSM.delAllSms();
                analogReference(INTERNAL);
                Pinlesen=analogRead(A0);
                Serial.println(Pinlesen);
                umwandelProzent= map(Pinlesen,907,1023,0,100);
                Serial.println(umwandelProzent);
                TextAkku="Akkustand="+String(umwandelProzent)+"%";
                TextAkku.toCharArray(sms_Akku,20);
                Serial.println(sms_Akku);
                GSM.sendSms(Handynummer1,sms_Akku);
                Serial.println("Akkustand geschickt");

                }

                if (smslesen.indexOf("ALARMAN")!=-1){
                  GSM.delAllSms();
                  Serial.println("Alarm ist an");
                  digitalWrite(6,HIGH);
                  Alarm = 1;


                   }
                 if (smslesen.indexOf("ALARMAUS")!=-1){
                   GSM.delAllSms();
                   AlarmBisher = 0;
                   ersteAusloesung = 0;
                   Serial.println("Alarm ist aus");
                   digitalWrite(6,LOW);
                   Alarm = 0;
                 }

                 if (smslesen.indexOf("GPSDATEN")!=-1){
                    GSM.delAllSms();
                     Serial.println("GPS Ermittlung eingeleitet");

                   while(timerGps==0){

                    digitalWrite(12,HIGH);
                    GetGps();
                     }
                     }
                     GSM.delAllSms();
                     }


                   if(digitalRead(7)==HIGH && Alarm ==1 ){
                   unsigned long AlarmMillis= millis();

                   if(digitalRead(7)==HIGH && Alarm ==1 && ersteAusloesung ==0){
                   ersteAusloesung=1;
                   AlarmBisher=AlarmMillis;
                   GSM.sendSms(Handynummer1,"Alarm,Fahrrad wird bewegt!");
                   Serial.println("Alarm ausgelöst");
                   }

while(digitalRead(7)==HIGH && Alarm ==1 && AlarmMillis - AlarmBisher >= Alarmintervall){
AlarmBisher=AlarmMillis;
unsigned long AlarmMillis= millis();
GSM.sendSms(Handynummer1,"Alarm,Fahrrad wird bewegt!");
Serial.println("Alarm ausgelöst");
  }
  }
  }
```

```
void GetGps() {
 unsigned long currentMillis= millis();
 ss.begin(9600);
  while (ss.available() > 0)
  gps.encode(ss.read());
  if (gps.altitude.isUpdated()){
Serial.print("Breitengrad=");  Serial.println(gps.location.lat(), 6);
Serial.print("Längengrad="); Serial.println(gps.location.lng(), 6);
Serial.print("Höhe=");  Serial.println(gps.altitude.meters(), 6);
WertLaenge=(gps.location.lng());
WertBreite=(gps.location.lat());
WertHoehe=(gps.altitude.meters());
TextGps = "Link: https://www.google.com/maps?q="+String(WertBreite,6)+","
+String(WertLaenge,6)+"height:"+String(WertHoehe,6);
TextGps.toCharArray(sms_Gps,160);
GSM.sendSms(Handynummer1,sms_Gps);
digitalWrite(12,LOW);
timerGps=1;
GSM.delAllSms();
}
else if (currentMillis >= timerEmpfang){
GSM.sendSms(Handynummer1,"Kein GPS Empfang");
timerGps=1;
digitalWrite(12,LOW);
GSM.delAllSms();
}
}
```