Nicolas Lugauer
Herr Faßbender
Informatik LK
SGR Q1 2019

Bau und Programmierung eines mikrocontrollergesteuerten Fallgerätes mit einem Arduino und eigener Testmessung

Inhaltsverzeichnis

1 Einleitung

1.1 Historisierung

Galileo Galilei gilt als Erster, der die Erdbeschleunigung g erforschte. Er stellte wichtige physikalische Eigenschaften dieser fest, wie zum Beispiel, dass die Erdanziehungskraft immer senkrecht nach unten wirkt und proportional zu der Masse des beschleunigten Körpers ist. Gemessen hatte Galileo die Konstante im freien Fall jedoch nicht. Dazu waren zu seinen Lebzeiten (*1564, ✞1642) die Uhren einfach noch zu ungenau. Daher war er gezwungen den Vorgang eines freien Falles zu verlangsamen und die wirkenden Kräfte durch eine schiefe Ebene aufzuteilen. Diese birgt aber Messfehler wie Reibung und Rotation.[1] 1651, kurz nach Galileos Tod, führte Giovanni Battista Riccioli Experimente auf dem ca. 100m hohen Asinelli-Turm in Bologna über die Erdbeschleunigung durch und bestimmte diese als 9,36$\frac{m}{s^2}$, was den heutigen 9,81$\frac{m}{s^2}$ erstaunlich nahe ist.[2]

1.2 Vorhaben und Grundkonzept

Rund 350 Jahre später sind Mikrocontroller in der Lage Zeiten auf ein Millionstel einer Sekunde genau zu bestimmen und sich somit der Konstanten weiter zu nähern. Meine Aufgabe war es ein Fallgerät zu bauen und mit solch einem Mikrocontroller (Arduino Mega mit einer Taktrate von 16MHz) Werte für die Berechnung der Erdbeschleunigung zu messen.[3] Das Grundkonzept war von Anfang an klar, da mir zwei PMMA-Röhren gegeben wurden, durch die ich ein Testkörper, ausgelöst von einem Elektromagneten, fallen lassen und diesen in regelmäßigen Abständen durch Lichtschranken messen sollte. Die Programmierung und der Bau dieser Konstruktion war mir überlassen.

2 Die Lichtschranken

2.1 Hardware

Es gibt drei Möglichkeiten bei der Wahl einer Lichtschranke, die such durch die Wellenlänge des zu durchbrechenden Strahles unterscheiden. Entweder nimmt man eine im ultravioletten (>400nm – 10^{-8}m), im sichtbaren (400nm – 780nm) oder im infraroten

[1] Vgl. https://wiki.zum.de/wiki/Galileo_Galileis_schiefe_Ebene, Zugriff am 19.01.2019
[2] Vgl. https://www.welt.de/print/die_welt/wissen/article112337878/Galilei-hatte-recht.html, Zugriff am 19.01.2019
[3] Vgl. Quelle zu „16MHz“, https://store.arduino.cc/mega-2560-r3, Zugriff am 21.01.2019

Bereich (780nm – 10^{-5}).[4] Mit dem sichtbaren Licht erwies es sich als schwierig, dieses präzise genug zu bündeln, sodass die Intensität auf dem Receiver so hoch ist, damit man die Sonneneinstrahlung vernachlässigen kann. Auch das ultraviolette Licht ist ein großer Bestandteil des Spektrums der Sonne[5], somit bestand auch hier das gleiche Problem. Bei meinen Recherchen stieß ich auf einen Infrarotsensor, welcher alle bisherigen Probleme lösen sollte. Der TSOP4836 ist ein Infrarotsensor zur digitalen Ansteuerung von Endgeräten, was bedeutet, dass er eigentlich dazu gemacht ist, Signale von zum Beispiel Fernbedienungen zu erkennen. Dementsprechend hat er eine relativ hohe Richtwirkung mit 45° und schirmt Umgebungslicht durch eingebaute Filter ab.[6] Jetzt fehlt noch das Gegenstück eines Empfängers, eine Quelle für infrarotes Licht. Da laut dem Datasheet des Sensors er eine maximale Empfindlichkeit bei 950nm erreicht, bestellte ich mir die Leuchtdiode IR383. Diese besitzt handliche Maße einer normalen LED und strahlt Licht mit 940nm in einem Winkel von 20° aus.[7]

2.2 Software

Um einen ersten Überblick von der Funktionsweise des Sensors zu bekommen durchsuchte ich das Datasheet nach elektrischen Spezifikationen und schloss ihn anschließend an den Arduino an. Was jedoch zu sehen war, war nicht sehr befriedigend. Zufällige Zahlen von null bis eins, die zwar irgendwie auf Helligkeit reagierten, ohne aber ein nachvollziehbares Muster erkennen zulassen, befüllten meinen Monitor. Bei weiterer Fehlersuche ist mir die „Carrier frequency“ von 36kHz aufgefallen. Diese ergibt natürlich Sinn, da der Empfänger zu Erkennung digitaler Signale konzipiert ist. Das heißt, dass wie bei dem digitalen Protokoll I2C die SerialClockLine der Master und der Slave (hier: der Sender und Empfänger) aufeinander abgestimmt sein müssen, damit der Master ein sinnvolles Signal aus dem Wechsel von Einsen und Nullen erkennen kann. Außerdem dient diese Frequenz als Individualisierung des Signales, da Infrarot auch unter Wärmestrahlung bekannt ist und somit Störungen verursachen kann. Um diese Verbindung also herzustellen, musste ich die LED auf 36kHz takten und ein ständiges Signal, welches beim Empfänger als LOW aufgenommen wird, senden. Dabei

[4] Vgl. Elektromagnetisches Spektrum, https://de.wikipedia.org/wiki/Elektromagnetisches_Spektrum#/media/File:Electromagnetic_spectrum_-de_c.svg, Zugriff am 8.02.2019
[5] Vgl. Spektrum der Sonne, https://www.spektrum.de/lexikon/physik/sonnenspektrum/13460, Zugriff am 26.04.2019
[6] Vgl. TSOP4836 Datasheet, https://www.vishay.com/docs/82459/tsop48.pdf, Zugriff am 26.04.2019
[7] Vgl. IR383 Datasheet, http://www.everlight.com/file/ProductFile/IR383.pdf, Zugriff am 26.04.2019

bediente ich mich der IRremote Libary, die nach ihrem Einbinden den Job mit nur 3 Zeilen Code erledigt.

Der Arduino wurde erneut angeschlossen und lieferte diesmal schöne Werte, welche entweder null oder eins betrugen. Es war also an der Zeit ein Programm zu entwickeln, um die Zeit nach dem Start bis zum Durchbruch der Lichtschranke und die Geschwindigkeit an der Lichtschranke zu messen. Das Erstere war recht einfach, da man durch den Befehl „millis()“ oder „micros()“ die Systemlaufzeit in Millisekunden bzw. in Microsekunden zurückgeliefert bekommt. Man merkt sich, wann die Messung beginnt und wann der entsprechende digitale Eingang erstmalig HIGH, also die Lichtschranke durchbrochen, wird. Danach bildet man die Differenz aus den Werten und erhält die Zeit, die das Objekt vom Anfang bis zur Lichtschranke brauchte. Damit bei jedem Durchlauf der loop-Methode nicht immer ein neuer Wert erzeugt wird, baute ich noch eine Abbruchvariable des Typen Boolean ein, die sich, wenn ein Wert gespeichert wurde, auf false setzt und somit das erneute Speichern verhindert.

Bei der Geschwindigkeit hingegen geht man wie folgt vor:
1. Man misst die Zeit, die das Objekt braucht die Lichtschranke zu durchqueren;
2. Man misst den Querschnitt des Objektes (vorausgesetzt es fliegt mittig);
3. Man berechnet die Geschwindigkeit mit der Formel $v = \frac{s}{t}$.

Schritt eins war hierbei jedoch nicht so trivial wie Schritt zwei und drei. Um die Zeit zu messen, in der sich das Objekt in der Lichtschranke befindet, muss man nicht nur die Zeit des erstmaligen Durchbruches, sondern auch die des Verlassens der Schranke ermitteln. Auf Abbildung 1 bezogen, bedeutet das, dass die erste Kante, die von null nach eins verläuft und die zweite Kante, die von eins nach null verläuft, erkannt werden muss.

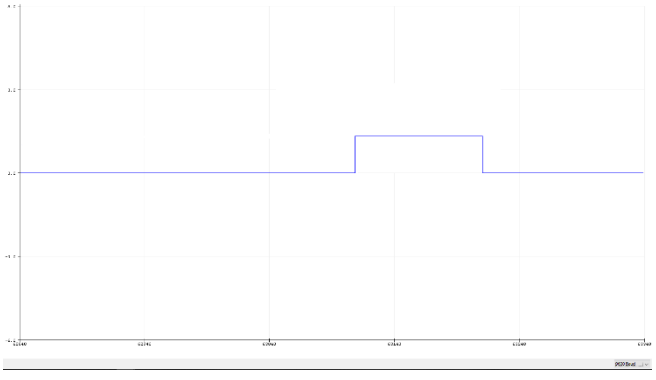


Abbildung 1

Für die erste Kante gilt die gleiche Strategie wie oben beschrieben, zur Messung der Zeit vom Start zur jeweiligen Lichtschranke. Bei der zweiten Kante aber reicht das einfache HIGH oder LOW als Input nicht aus, da in diesen Fall das letzte HIGH Signal bevor einem LOW, somit der Wechsel dieser, gesucht wird. Man merkt sich also das

Signal, welches man davor bekommen hat und wenn dieses ungleich des gewünschten Signales ist, hier LOW, bestimmt man die Zeit, wann das Objekt die Lichtschranke verlassen hat. Als ich diese Idee nun implementierte und an einer Testvorrichtung ausprobierte, kamen zu meiner Überraschung sehr viele und sehr kleine Werte zum Vorschein, vor allem wenn ich das Objekt etwas langsamer hindurch zog. Bei erneuter detaillierter Analyse der Daten vom seriellen Plotter, welcher mir auch schon Graphen ähnlich wie Ab1. lieferte, entdeckte ich, dass bei manchen Testversuchen der Übergang von HIGH auf LOW nicht so glatt wie in Ab1 verlief, sondern mit Störungen behaftet war (siehe Ab2). Zur Erkennung der ersten Kante war dies nicht schlimm, da man ja nach dem ersten Signal, welches eins beträgt, sucht. Bei der fallenden Kante hingegen wurden diese Störungen als Abschluss der Messung interpretiert. Um dieses Schwanken zwischen null und eins nicht jeweils als einzelne Durchquerung der Lichtschranke zu interpretieren, muss man diese ignorieren. Was dazu noch auffällig war, ist das die jeweiligen Längen der Schwankungen extrem gering waren. Vielleicht waren sie ja so gering, dass sie physikalisch ausgeschlossen sind und somit keine Verwechslungen mit dem richtigen Durchflug eines Objektes entstehen können. Durch das Umformen von Formeln zur beschleunigten Bewegung kam ich auf die Formel

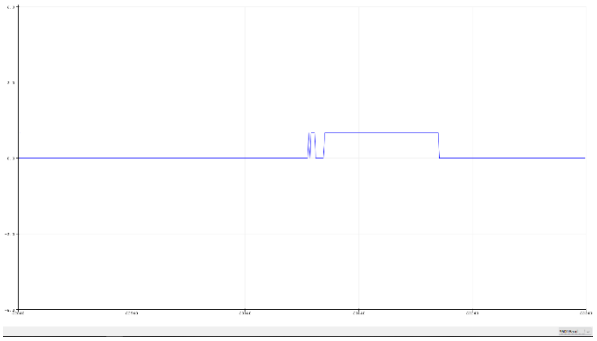


Abbildung 2

$t = \frac{s1}{g * \sqrt{\frac{2*s2}{g}}}$, welche die physikalisch korrekte Zeit, einer Durchquerung einer Lichtschranke, eines Objektes mit dem Durchmesser s1 liefert, welches von einer Höhe s2, mit g beschleunigt wird. Bei meinem Versuchsaufbau mit der Höhe von 1,80m und einem Golfball mit dem Durchmesser von 4,37cm, ergab sich eine maximale Geschwindigkeit von 5,942m/s und eine minimale Zeit von 7,354ms. Durch diese Rechnung konnte ich in der Software alle Werte unter dieser Grenze ignorieren. Also sobald die Differenz der Systemlaufzeit und der Zeit der steigenden Kante größer als 8ms und die vorher genannten Bedingungen erfüllt sind, kann man die Zeit der sinkenden Kante nehmen und somit auf die Geschwindigkeit des Objektes schließen.

3 Elektronik

Nachdem alle Teile einzeln funktioniert hatten, war der nächste Schritt, diese zusammenzuführen. Ich fing mit den TSOPs an, welche genau drei Anschlüsse besaßen: OUT, Ground, Vin.[8] Um wenig Kabel zu verwenden, entschied ich mich für eine Parallelschaltung für die Stromversorgung der Sensoren. Somit hatte ich insgesamt acht Kabel auf dieser Seite, die sich aus zwei für plus und minus und sechs für das Signal an den Arduino zusammensetzen. Als Stromquelle nahm ich den Mikrocontroller selber, da sich zwar durch die Parallelschaltung der Strom addiert, jedoch dieser mit 6* 0.7mA immer noch deutlich unter dem Powerlimit von 20mA bei digitalen Pins liegt.[9] Als Nächstes waren die LEDs dran, die ich takten und mit Strom versorgen musste. Auch hier entschied ich mich, um Kabel zu sparen, für eine Parallelschaltung, was allerdings auch ein Problem mit sich führte. Wie gesagt, schafft es ein digitaler Pin um die 20mA an Strom zu liefern. Wenn, man nun aber den nötigen Strom der LEDs summiert, kommt man auf 120mA. Dennoch müssen die LEDs an einen PWM-Pin, um auf 36kHz gebracht zu werden. Um trotzdem die Dioden zum Leuchten zu bringen, verwendete ich einen Transistor (IRFZ34N), welcher mein Signal auf einen anderen Stromkreis überträgt. Ich baute also ein Power Distribution Board, welches meine LEDs und meinen Elektromagneten die korrekte Apannung und genügend Ampere zur Verfügung stellte. Da die IR383 keine 12V aushielten, sondern 1,2V und 20mA pro Stück zogen, brauchte ich noch einen Vorwiderstand. Dieser sollte 100 Ohm groß sein, welches sich durch die Rechnung $R = \frac{U}{I} \rightarrow 100\Omega = \frac{12V}{0.12A}$ ergab. Für den Elektromagneten benutze ich einen zweiten Transistor, da diese einen wesentlich schnelleren Schaltvorgang als ein Relais besitzen und außerdem handlicher zum Löten war. In Abbildung 3 ist der gesamte Schaltplan, der mit dem Programm Fritzig erstellt wurde, mit exemplarisch vier Lichtschranken zu sehen. Die gelb dargestellten Leiter sind für die Übertragung von Signalen zuständig.

[8] Vgl. TSOP4836 Datasheet, https://www.vishay.com/docs/82459/tsop48.pdf, letzter Zugriff am 26.04.2019
[9] Vgl. „20mA“, https://store.arduino.cc/mega-2560-r3, Zugriff am 21.01.2019

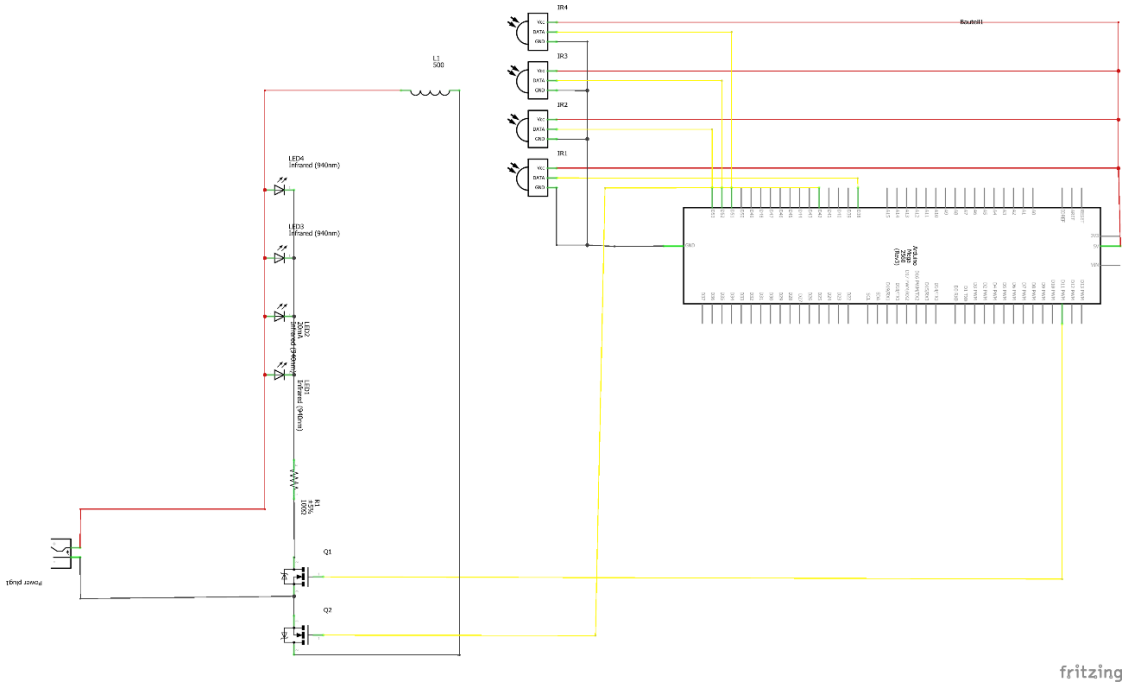


Abbildung 3

4 Zusammenbau der Komponenten

Als Grundgerüst der Konstruktion war Holz eine gute Wahl, da es relativ leicht zu bearbeiten und schwer genug ist, um eine stabile Basis für ein 1,80m hohes Rohr zu bilden. Die Maße des Kastens sind 40cm*40cm*24cm, wobei die Fallröhre nicht dezentral positioniert wurde, um Platz für den Touchscreen zu lassen. Die beiden mir zur Verfügung gestellten Röhren wurden von mir mit einem Plexiglaskleber namens Acrifix unlösbar verklebt.

Beginnend am Elektromagneten, welchen ich zentral durch eine Plexiglas Vorrichtung über der Röhre platziert habe, findet man jede ca. 25cm eine Lichtschranke, deren Befestigung der wohl zeitaufwendigste Teil des Bauens war. Die LEDs haben einen präzisen Durchmesser von 5mm, welchen man mit einem Bohrer recht leicht in den PVC Kabelkanal und die Plexiglasröhre gebohrt bekommt. Danach konnte ich sie problemlos in das passende Loch stecken und mit einem Tropfen Sekundenkleber fixieren. Die Sensoren hingegen waren viereckig und hatten ihre Anschlüsse nicht nach hinten, sondern nach unten abstehend. Tagelang probierte ich alle Kleber aus, die mir zur Verfügung standen, um diese viereckigen Sensoren möglichst mittig in eine 6mm Bohrung zu bekommen. Sekundenkleber klebte kein PVC, Plexiglaskleber klebt nur Plexiglas und muss zum Härten zwölf Stunden mit UV Licht bestrahlt werden, Heißkleber war zu dickflüssig und Epoxidkleber zu dünnflüssig. Die endgültige Lösung

war dann die Hohlräume zwischen den Sensor und dem Kabelkanal mit PVC-Kleber aufzufüllen, zwei Minuten warten, bis der Kleber anzieht und dabei den Sensor in der richtigen Position gedrückt halten und dann mit Heißkleber den Rest auffüllen. Also Testobjekt kam ein Golfball zur Verwendung, welcher eine Schraube beinhaltet, um von dem Magneten ausgelöst zu werden. Den Golfball habe ich wegen dessen Aerodynamik ausgewählt.

5 Fallversuch und dessen Auswertung

	Messung1	Messung2	Messung3	Messung4	Messung5	Messung6	Messung7	Messung8	Messung9
Sensor1	231404	233060	232984	231392	230532	231472	231728	231312	229984
Sensor2	326380	328096	326488	325104	324104	326620	327684	324696	324856
Sensor3	401684	399526	400000	400000	400000	400000	400000	400000	400000
Sensor4	461036	462740	461136	459736	458680	459472	459608	459244	458012
Sensor5	514216	515904	514312	513048	512072	512924	513220	512772	511528
Sensor5	563756	565364	563860	562508	561460	562228	562452	562140	560744
Sensor1	3,27	3,24	3,64	3,51	3,56	3,55	3,62	3,45	3,4
Sensor2	4,77	4,82	4,89	4,86	5,04	5,84	7,25	4,82	5,04
Sensor3	6,15	6,06	6	6	6	6	6	6	6
Sensor4	7,49	7,33	7,49	7,18	7,17	6,79	6,8	6,66	6,59
Sensor5	6,41	6,41	6,47	6,35	6,4	6,35	6,41	6,29	6,28
Sensor6	8,72	8,66	8,78	8,56	8,45	8,14	8,24	8,24	7,85

Die horizontale Achse sind die neun verschiedenen Fallversuche und die vertikale die jeweiligen Messdaten. Von Zeile eins bis sechs sind die Mikrosekunden von dem Start der Messung zu einer Lichtschranke und von Zeile 8 bis 13 die Geschwindigkeiten des Balles an einer Lichtschranke in m/s. Die dritte Lichtschranke hat leider nach der zweiten Messung nur noch zufällige Zahlen zurückgegeben, weshalb ich sie in der Software fürs Erste deaktivierte. Auffällig ist auch noch, dass die Werte der Zeitmessung erstaunlich nah aneinander sind und sich nur in wenigen Millisekunden unterscheiden. Anhand der Abstände der Lichtschranken und der gemessenen Zeiten habe ich ein s-t-Diagramm erstellt und die Werte durch Regression in eine Funktion überführt. Die zweite Ableitung dieser ergab $\frac{d^2}{dx^2}(4{,}8068 * x^2) = 9{,}6136\,\frac{m}{s^2}$.
Da aber noch mehr Kräfte auf den Golfball wirken als die Schwerkraft, wie zum Beispiel die Auftriebskraft und der Luftwiderstand, sind $9{,}6136\frac{m}{s^2}$ ein durchaus realistischer Wert, für die Erdbeschleunigung. Die gemessenen Geschwindigkeiten streuen etwas stärker als die diskutierten Zeiten. Da man jetzt die Funktion hat, die die Bewegung des Balles beschreibt und deren erste Ableitung ein v-t-Diagramm ist, kann man die gemessenen Daten rechnerisch vergleichen.

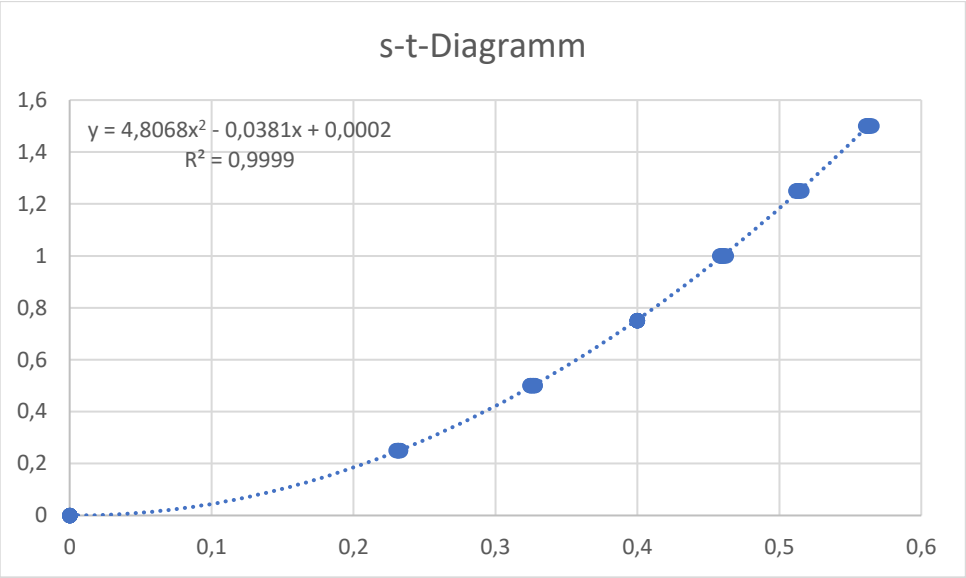
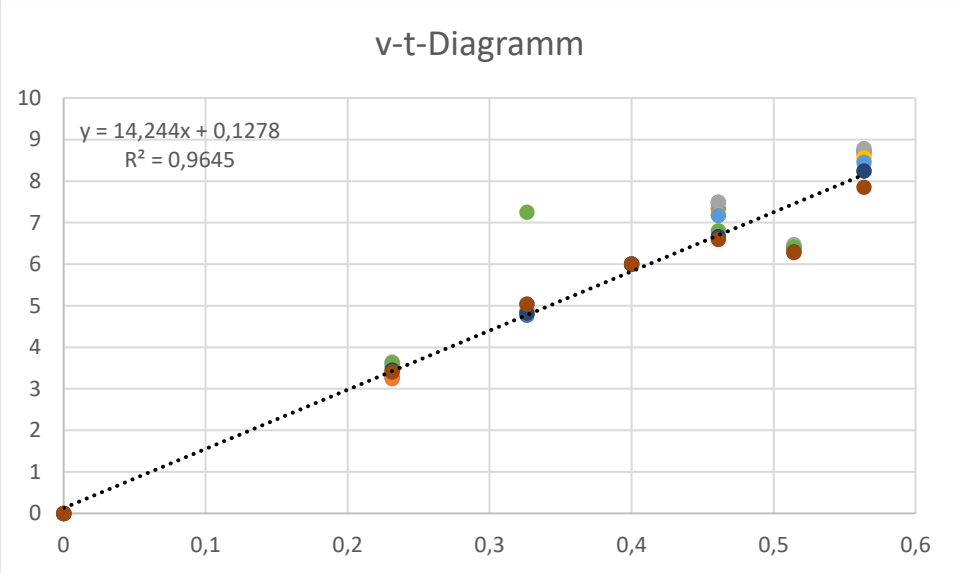


Abbildung 4: s-t-Diagramm



1Abbildung 5: v-t-Diagramm

In Abbildung 5 sind zwei Merkmale sehr prägnant: Die Steigung, welche mit 14,244 67% über den eigentlichen 9,61 liegt und der fünfte Messwert, welcher immer unter dem Wert von Sensor vier zu finden ist. Diese großen Messfehler könnte auf die verschiedenen Messwinkel der TSOP-Sensoren zurückführen. Da diese nicht exakt gleich eingeklebt sind und somit mechanische Bauteile das Sichtfeld beeinträchtigen, ist auch die zu durchfliegende Strecke meist unterschiedlich.

6 Herausforderungen

Der Mikrocontroller Atmega2560 hat fünf kleine Hardwarestrukturen, welche direkt mit dem Prozessortakt verknüpft sind. Diese Strukturen nennt man Timer und regeln

eigentlich alles was mit Zeiten zu tun hat, wie zum Beispiel delay(), millis(), oder PWM-Signale. Ein Timerregister hat entweder die Größe von 8bit oder 16bit und wird bei jedem Takt um eins erhöht, bis es zu einem Overflow kommt. Damit beginnt das Ganze wieder von vorne. Da man auch mit 16bit, welches einen Wertebereich von 2^{16} = 65536 abdeckt, bei einer Taktrate von 16MHz nur ca. 4 Millisekunden hat, bis das Register voll ist, gibt es einen Prescaler, der den Zusammenhang zwischen der Taktrate und dem Hochzählen des Timers verlangsamt. Der Arduino selbst benutzt für die standardmäßigen Befehle den Timer 0.[10]

Doch was genau hat das mit meinem Projekt zu tun?

Nun, um die LED auf 36kHz zu takten, ist eine hohe Präzision vonnöten und der geschriebene Code darf diesen Prozess nicht beeinflussen. Ein Timer erfüllt diese Forderungen, da durch ein Interrupt, welches zum Beispiel einen Overflow des Registers erkennt, diese Priorität gegeben wird. Die Library „IRremote“, verwendet standardmäßig den Timer 2, dessen Ausgabevergleichsregister (OCR2B (Assembler)) bei dem Atmega2560 an den physischen Pin 9 verbaut wurde.[11,12] Das Problem bestand jetzt darin, dass auch der Touchscreen in seiner Library „Elegoo GFX“ den Timer 2 beanspruchte und ich somit keinen digitalen Ausgang mehr für die LEDs frei hatte.[13] Durch ein Blick in die IRremote Library, fiel mir die Datei Boarddefs.h auf, welche alle Voreinstellungen von bekannten Mikrocontrollern beinhaltete. In Zeile 82 konnte man letztendlich einen gewünschten Timer für den Atmega2560 auswählen. Ich entschied mich für den Fünften, da sein Ausgangspin auf den digitalen Pin 46 lag, der nicht mit dem Touchscreen interferierte.

Eine zweite Herausforderung verbarg sich in den Sensoren. Nach dem Zusammenbau der Konstruktion testete ich die Sensoren, in dem Glauben, dass ich mich danach direkt an die Software setzen konnte. Leider lieferten nur genau 5 von den 12 Sensoren, die mir zugesandt wurden, ein defektfreies Signal. Da alles schon verklebt war, musste ich neue Sensoren bestellen, von vorne anfangen und die Vorrichtung erneut bauen. Dieses Mal reagierten sie zuverlässig, hatten jedoch wesentlich mehr Störungen als die Funktionierenden der ersten Lieferung. Zufällig stoß ich auf die Verpackung, welche mir die Information lieferte, dass es sich bei den Sensoren nicht um TSOP4836

[10] Vgl. https://www.exp-tech.de/blog/arduino-tutorial-timer, Zugriff am 26.04.2019
[11] Vgl. https://github.com/z3t0/Arduino-IRremote, Zugriff am 26.04.2019
[12] Vgl. https://www.arduino.cc/en/Hacking/PinMapping2560, Zugriff am 26.04.2019
[13] Vgl. Elegoo Libraries, als Datei in Literaturverzeichnis

handelte, sondern um TSOP4838. Dieser hatte eine Carrier frequency von 38kHz. Nach dem Umstellen der Software liefen sie einwandfrei.[14]

7 Fazit

Es ist möglich sich der Erdbeschleunigung mit Mikrocontrollern zu nähern und wie damals Galileo Galilei wichtige physikalische Eigenschaften dieser zu erforschen. Das Projekt war eine erfrischende Kombination aus praktischen Bauen, Gestaltung der Oberfläche und dem tiefen Auseinandersetzen mit den Strukturen eines Mikrocontrollers. Auch die Themenkombination aus Physik und Informatik hat mir Freude bereitet, obwohl die wohl eher lästige Fehlersuche diese in Grenzen hielt. Vielleich wir mein Gerät ja als Demonstrierung im Unterricht verwendet. Trotz allen Herausforderungen hat mir das Projekt einen gelungenen Einblick in die Konzipierung von mikrocontrollergesteuerten Geräten gegeben.

[14]Vgl. TSOP4836 Datasheet, https://www.vishay.com/docs/82459/tsop48.pdf, letzter Zugriff am 26.04.2019

8 Anhang

8.1 Literaturverzeichnis

(1) Arduino Mega Spezifikationen, https://store.arduino.cc/mega-2560-r3, Zugriff am 21.01.2019

(2) Arduino Timer und Interrupts, https://www.exp-tech.de/blog/arduino-tutorial-timer, Zugriff am 26.04.2019

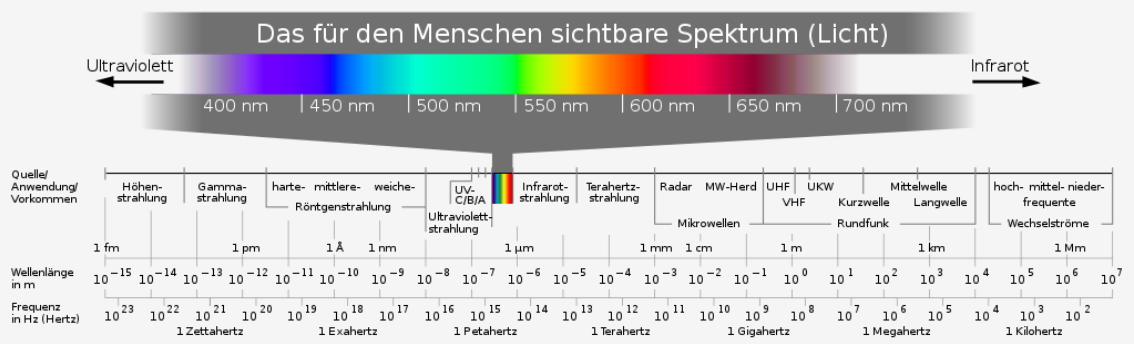
(3) Atmega2560 Pin Layout, https://www.arduino.cc/en/Hacking/PinMapping2560, Zugriff am 26.04.2019

(4) Die Welt, Galilei hatte recht, Norbert Lossau, Veröffentlicht am 02.01.2013, https://www.welt.de/print/die_welt/wissen/article112337878/Galilei-hatte-recht.html, Zugriff am 19.01.2019

(5) Elegoo Libraries

a

(6) Elektromagnetisches Spektrum, https://de.wikipedia.org/wiki/Elektromagnetisches_Spektrum#/media/File:Electromagnetic_spectrum_-de_c.svg, Zugriff am 8.02.2019



Von Horst Frank / Phrood / Anony - Horst Frank, Jailbird and Phrood, CC BY-SA 3.0

(7) Galileo Galileis schiefe Ebene, https://wiki.zum.de/wiki/Galileo_Galileis_schiefe_Ebene, Zugriff am 19.01.2019

(8) IRremote Library von z3t0, https://github.com/z3t0/Arduino-IRremote, Zugriff am 26.04.2019

(9) IR383 Datasheet, http://www.everlight.com/file/ProductFile/IR383.pdf, letzter Zugriff am 26.04.2019

(10) Modifizierte IRremote Library

IRremote mod.zip

(11) Spektrum der Sonne,
https://www.spektrum.de/lexikon/physik/sonnenspektrum/13460, Zugriff 26.03.2019

8.2 Selbstständigkeitserklärung

Ich erkläre, dass ich die Facharbeit ohne fremde Hilfe angefertigt und nur die im Literaturverzeichnis angeführten Quellen und Hilfsmittel benutzt habe.

Nicolas Lugauer