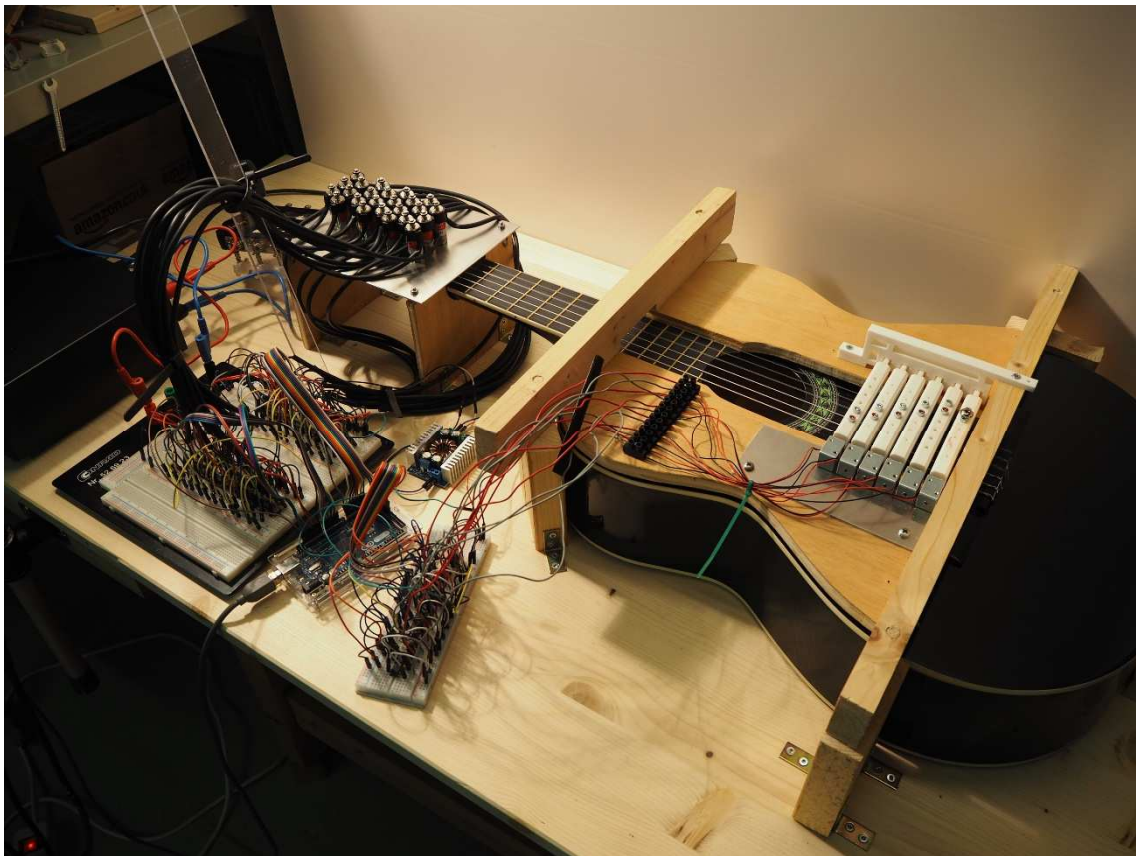


Autoplay – Konstruktion einer selbstspielenden Gitarre



Raphael Bordt, 4e

Kantonsschule Im Lee

Maturitätsarbeit HS 2018/19

betreut von Hannes Toggenburger

Winterthur, 7.1.2019

Vorwort

Schon lange versucht der Mensch, sich den Alltag zu erleichtern, und setzt dazu Maschinen ein, welche seine Arbeit machen. Warum also nicht das Gitarre-Spielen einer Maschine überlassen? Dieser Gedanke, der mir während des Gitarre-Spielens kam, brachte mich auf die Idee einer selbstspielenden Gitarre.

Neben der Musik, speziell dem Gitarre-Spielen, gehören handwerkliches Arbeiten und Konstruieren sowie die Informatik zu meinen grössten Leidenschaften. Meine Maturarbeit sollte in einem dieser Bereiche angesiedelt sein. Auch wollte ich, dass es sich um eine praktische Arbeit handelt. Das Thema «Autoplay – Konstruktion einer selbstspielenden Gitarre» verbindet nun alle Themenbereiche, was mich sehr faszinierte. Zwar wurde ich von verschiedener Seite darauf hingewiesen, dass das geplante Projekt eine hohe Komplexität aufweise und es unklar sei, ob es zu realisieren ist. Der Herausforderung wollte ich mich aber dennoch stellen.

Ein Grund dafür war die bei mir zu Hause vorhandene Infrastruktur, welche unter anderem einen Bastelkeller mit entsprechendem Werkzeug beinhaltet. Auch haben Personen aus meinem Umfeld entsprechendes Knowhow, auf das ich zurückgreifen konnte.

So danke ich an dieser Stelle herzlich meiner Familie für die moralische und finanzielle Unterstützung, speziell meinem Vater, mit dem ich mich über alle technischen Fragen austauschen konnte, der mitdachte und mir auch in schwierigen Situationen gute Ratschläge gab. Peter Rutschmann danke ich für die zwei intensiven Gespräche, in denen er mir die Möglichkeiten, vorwiegend im Informatikbereich, aufzeigte. Hannes Toggenburger danke ich für die sehr angenehme, sowohl fordernde wie teilnehmende Betreuung, in welcher er sich jederzeit meinen Fragen stellte.

Im Nachhinein bereue ich es nicht, dieses Thema gewählt zu haben. Ich habe mir in unterschiedlichsten Bereichen viel neues Wissen angeeignet, zum Beispiel den Umgang mit CAD-Programmen, und ich freue mich über das Endprodukt.

Inhaltsverzeichnis

| | |
|--|----|
| Vorwort..... | 1 |
| 1 Einleitung | 3 |
| 2 Hauptteil | 4 |
| 2.1 Selbstspielende Musikinstrumente | 4 |
| 2.1.1 Begrifflichkeiten | 4 |
| 2.1.2 Geschichtlicher Abriss..... | 4 |
| 2.1.3 Wozu ein mechanisches Musikinstrument? | 5 |
| 2.2 Arbeitsprozess..... | 6 |
| 2.2.1 Allgemeine Überlegungen..... | 6 |
| 2.2.1.1 Klärung des Projekts | 6 |
| 2.2.1.2 Möglichkeiten der Umsetzung | 6 |
| 2.2.1.3 Grundsatzentscheid Arduino | 6 |
| 2.2.2 Mechanische Konstruktionen | 7 |
| 2.2.2.1 Zupfautomat | 7 |
| 2.2.2.2 Drückmechanismus | 11 |
| 2.2.3 Software..... | 13 |
| 2.2.3.1 Programmierung..... | 13 |
| 2.2.3.2 Serielle Schnittstelle..... | 16 |
| 2.2.3.3 Python..... | 17 |
| 2.2.4 Ansteuerung der Magnete..... | 18 |
| 2.2.5 Endprodukt | 20 |
| 2.2.6 Fazit | 21 |
| 2.3 Weiterführende Überlegungen..... | 22 |
| 3 Schlusswort..... | 23 |
| 4 Quellenverzeichnis | 24 |
| Anhang | 26 |

1 Einleitung

Ziel dieser Maturarbeit ist eine selbstspielende Gitarre: Über den Computer soll es möglich sein, dass eine über einer Gitarre angebrachte Vorrichtung, welche einerseits die Töne auf den Gitarrensaiten drückt und andererseits die Gitarrensaiten zupft, verschiedene Lieder wiedergeben kann.

Es handelt sich dabei im Grund um ein Experiment. Für das Gelingen des Projekts müssen sehr unterschiedliche Fragestellungen angegangen, ausprobiert und geklärt werden und deren Ergebnisse schliesslich ineinanderfliessen. Sie sind zunächst einzeln anzupacken, um auf jeden Fall die Resultate einzelner Schritte festhalten zu können. Denn erst sehr spät im Verlauf des Prozesses stellt sich heraus, ob die Gesamtidee schlussendlich realisierbar ist.

Die hier vorliegende schriftliche Arbeit folgt nach einem eher grundsätzlichen, historischen Teil über selbstspielende Musikinstrumente (2.1) den verschiedenen Fragestellungen. Sie stellt dabei den konkreten Arbeitsprozess weniger chronologisch als thematisch dar (2.2). Dabei wird Wert darauf gelegt, die Überlegungen während des Experimentierens, auftretende Probleme und deren Lösungen jeweils nachvollziehbar zu präsentieren und gleichzeitig auch den technisch-physikalischen Hintergrund zu beleuchten.

Nach allgemeinen Überlegungen und der Darlegung von Grundsatzentscheidungen (2.2.1) wird zunächst die Erarbeitung der mechanischen Konstruktionen dargestellt (2.2.2), die über die Gitarre gebaut wurden: einerseits der Zupfautomat, andererseits der Drückmechanismus. Danach wird erklärt, was die Software können muss und wie dies in einem selbst erstellten Programm realisiert wurde (2.2.3). Die Zusammenführung der verschiedenen Teile geschieht durch die Verkabelung der Einzelteile beziehungsweise die Ansteuerung der Magnete (2.2.4). Nach der Präsentation des Endprodukts (2.2.5) und einem Fazit zum gesamten Arbeitsprozess (2.2.6) lege ich einige weiterführende Überlegungen dar (2.3).

2 Hauptteil

2.1 Selbstspielende Musikinstrumente

2.1.1 Begrifflichkeiten

Zu Beginn müssen zwei Begriffe eingeführt werden: das mechanische und das selbstspielende Musikinstrument.

Das mechanische Musikinstrument wird in der Zeitschrift «Das mechanische Musikinstrument» [5, S. 44-45] von Herbert Jüttemann und Jürgen Hocker folgendermassen definiert: «Das mechanische Musikinstrument ist ein Musikinstrument, bei dem in einem Steuerteil die Tonfolge von einem Toninformationsträger abgelesen und zur Tonauslösung an einen Schallquellenteil (eigentliches Musikinstrument) mit einer festen Tonskala beziehungsweise ansteuerbaren festen Frequenzen abgegeben wird.»

Da auch handgespielte Instrumente mechanische Teile enthalten können, fällt schnell der Begriff «selbstspielend» anstelle des «mechanisch». Dieses «selbstspielend» kann allerdings nur auf das Ansteuern der Instrumente bezogen werden, da, wie im Beispiel einer Drehorgel, der Antrieb von menschlicher Seite kommt [3, S. 6-7].

In dieser Arbeit verwende ich die beiden Begriffe synonym.

2.1.2 Geschichtlicher Abriss

Die ältesten selbstspielenden Musikinstrumente, die uns teilweise noch heute erhalten sind, sind Glockenspiele, welche über Stiftwalzen gesteuert werden. Stiftwalzen sind Zylinder, welche auf verschiedenen Linien Löcher enthalten, in welche Stifte gesteckt werden können. Diese Zylinder werden gedreht. Über der Walze sind Hebel angebaut, welche von den Stiften der Walze weggedrückt und damit angeschlagen werden. Auf diese Art kann der Zeitpunkt des Spielens und die Tonhöhe festgelegt werden. Solche Stiftwalzen wurden etwa im zwölften Jahrhundert gefertigt. Rund 150 Jahre später entstanden die ersten Turmglockenspiele, von welchen heute noch zahlreiche existieren. Damals galten sie als Wahrzeichen für die jeweilige Städte. In der Spätrenaissance wurden erste mechanische Instrumente in Auftrag gegeben, allerdings nur von reichen

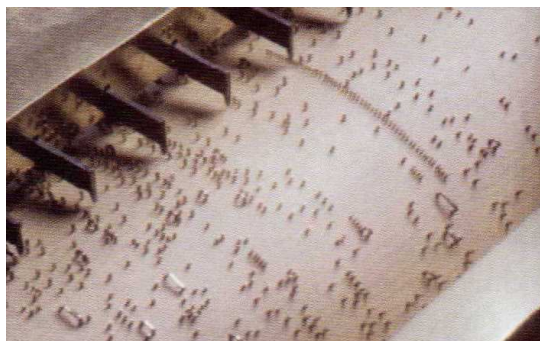


Abbildung 1: Stiftwalze der Haydn-Flötenuhr [2, S. 18]

Fürsten. Erst mit den Flötenuhren im 18. Jahrhundert, kamen die mechanischen Musikinstrumente auch in bürgerliche Häuser. Für diese komponierten unter anderem Berühmtheiten wie Mozart und Beethoven. Auch die Flötenuhren wurden durch

Stiftwalzen gesteuert, mittlerweile enthielten diese aber schon mehrere Lieder, da die Anordnung der Stifte auf den Walzen deutlich kleinräumiger und präziser geworden war [2, S.11-20].

Im 19. Jahrhundert entstanden die ersten Orchestrions, also Musikmaschinen, welche in der Lage waren, ganze Orchester zu imitieren. Die Stiftwalzen wurden immer kleiner und wurden später aus Messing gefertigt. Es entstanden Musikdosen oder auch Spieluhren, die bis heute nach demselben Prinzip funktionieren. Der Nachteil, dass die Stiftwalzen nur wenige Lieder enthalten und nicht austauschbar sind, wurde Ende des 19. Jahrhunderts behoben, indem die Stiftwalzen durch Metallplatten, die mit Noppen versehen sind, ersetzt wurden.

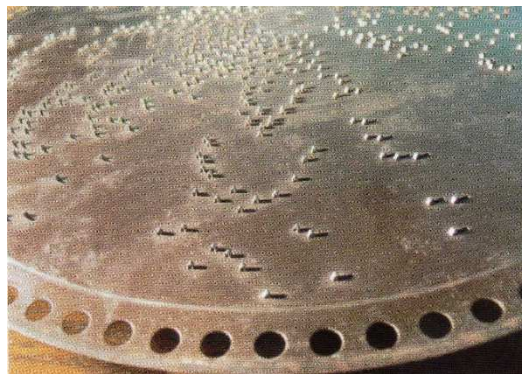


Abbildung 2: Metallplatte [2, S. 30]

Diese Metallplatten sind austauschbar. Dadurch konnte man nun verschiedene Lieder sammeln. Eine Weiterentwicklung der Metallplatten mit Noppen ist die CD, welche heute noch nach fast demselben Prinzip läuft, nur dass die Noppen so klein geworden sind, dass sie von Auge nicht mehr zu erkennen sind, und dass die Daten nicht mehr direkt mechanisch vertont, sondern von einem Laser abgelesen werden [2, S. 20-30].

Auch in meinem Projekt ist das Ziel, Lieder immer nach derselben Methode erstellen und auswechseln zu können, ohne das Instrument und die dazugehörige Software austauschen zu müssen.

2.1.3 Wozu ein mechanisches Musikinstrument?

Musik ist nicht lebensnotwendig, doch kann Musik Gefühle vermitteln und Menschen beeinflussen und so eine Bereicherung fürs Leben sein. Der Wunsch, ein mechanisches Musikinstrument zu besitzen, ist dadurch aber noch nicht besprochen. Ein mechanisches Musikinstrument hat gewisse Stärken gegenüber dem vom Menschen gespielten Instrument: Es ist jederzeit verfügbar und kann von unmusikalischen Menschen bedient werden. Die Maschine kann unter Umständen mehr, als ein Mensch kann. Zum Beispiel kann sie auf einer Gitarre sechs weit auseinanderliegende Bündel gleichzeitig drücken, was dem Mensch mit fünf Fingern und einer begrenzten Spannbreite der Hand nicht möglich ist. Dem Komponisten sind so keine Einschränkungen durch menschlich begrenzte Spielmöglichkeiten gesetzt. Dazu kann eine Maschine das Tempo problemlos steigern. Auch können einige mechanische Musikinstrumente Klänge erzeugen, welche auch im Zeitalter von Internet und Streaming-Diensten wie Spotify oder YouTube sehr aussergewöhnlich sind, weil sie einen ganz eigenen Charakter haben. Zum Beispiel ist die Jahrmarktsmusik im Radio oder auf YouTube wenig verbreitet und damit kaum zu hören [3, S. 4-5].

2.2 Arbeitsprozess

2.2.1 Allgemeine Überlegungen

2.2.1.1 Klärung des Projekts

Um das Projekt in Angriff zu nehmen, bedarf es vorgängig einer soliden Ideenausarbeitung. Das gängige Gitarrenspiel sieht bei erstmaligem Betrachten nicht nach einem komplizierten Prozess aus. Bereits kleinen Kindern gelingt es, ganze Lieder auf einer Gitarre zu spielen. Sobald dieser Ablauf jedoch mechanisiert und automatisiert werden soll, wird der Ablauf schnell kompliziert. Darum ist zu Beginn wichtig, das Gesamtprojekt in angemessene Teilschritte zu unterteilen und Arbeitsweise und Vorgehen der einzelnen Phasen zu klären, dies auch, um eine Absicherung der verschiedenen Schritte zu garantieren.

Einfach erklärt lässt sich das Gitarrenspiel in die Aufgaben der linken und der rechten Hand differenzieren. Die linke Hand drückt dabei die Saiten in bestimmten Gitarrenbünden, um die Tonhöhe zu regulieren. Die rechte Hand ist dafür verantwortlich, eine oder mehrere Saiten anzuschlagen oder zu zupfen. Es wird schnell klar, dass diese zwei Schritte für die Mechanisierung einzeln umgesetzt werden müssen. Als Vereinfachung ist es naheliegend, sich bezüglich beider Aufgaben zuerst auf eine der sechs Saiten zu konzentrieren, und erst wenn eine Lösung gefunden wird, diese auf die anderen fünf Saiten zu übertragen.

Damit diese Hardware angesteuert werden kann, ist zusätzlich notwendig, eine entsprechende Software zu erarbeiten und sie mit der Mechanik zu verknüpfen, damit das Endprodukt über einen Computer gesteuert beziehungsweise gespielt werden kann.

2.2.1.2 Möglichkeiten der Umsetzung

Für alle Teilbereiche gibt es grundsätzlich verschiedene Umsetzungsmöglichkeiten. Dabei ist anfangs nicht abzusehen, welche die beste und überhaupt realisierbare Variante ist. Mehrere Faktoren spielen eine Rolle. Neben finanziellen Aspekten geht es vor allem um Fragen der technischen Realisierbarkeit, der Präzision und des Platzes, zumal auf einer Gitarre die Platzverhältnisse eng sind. Um diese essentielle Probleme zu lösen, ist am Anfang eine Experimentierphase zu absolvieren, in der es hauptsächlich darum geht, verschiedene Methoden auszuprobieren, zu testen und allfällige Fehlerquellen zu sehen und wenn möglich zu eliminieren. Dieser Prozess des Fehler-Findens und -Behebens ist im vorliegenden Projekt grundlegend und allgegenwärtig bis zum Schluss der Arbeit.

2.2.1.3 Grundsatzentscheid Arduino

Um erstes Material anzuschaffen, war ein Gespräch mit einem Experten wegweisend.¹ In diesem konnten grundsätzliche Fragen zum Material besprochen werden und es wurden Ansatzmöglichkeiten diskutiert. Dabei fiel als wesentlicher Entscheid, dass ein Arduino die richtige Schnittstelle zwischen Computer und Mechanik sein soll, weil er einfach zu bedienen ist und kein eigenes Betriebssystem braucht, wie es beispielsweise bei einem Raspberry Pi der Fall gewesen wäre. Dazu kann der Arduino über die zum

¹ Peter Rutschmann, Softwareentwickler, Informatiklehrer an der Berufsbildungsschule Winterthur

Mikrocontroller gleichnamige Open Source Software benutzerfreundlich angesteuert werden.

Arduino bezeichnet zweierlei: Einerseits das Arduino-Board, dessen wesentliche Teile auf einer Platine befestigte Mikrocontroller sowie verschiedenen Pins sind, in welche Kabel eingesteckt werden können. Andererseits wird als Arduino die dazugehörige Entwicklungsumgebung bezeichnet, also die Software, auf welcher Programme geschrieben werden können. Der Programmiersprache des Arduinos ist C++ sehr ähnlich [1, S. 1-10]. In diesem Projekt verwende ich einen Arduino Mega, da dieser mehr Pins hat als ein Arduino Uno.

2.2.2 Mechanische Konstruktionen

2.2.2.1 Zupfautomat

2.2.2.1.1 Erste Überlegungen zum Zupfautomat

Die erste Idee war, die Saite mit einem an einem Drehmotor angeschlossenen Plektrum anzuschlagen. Das Problem dabei ist, dass ein normaler Drehmotor, wie er bei einem Radantrieb vorkommt, nicht fein genug justiert werden kann, um einen Ton auf einer Gitarre in einem präzisen Rhythmus zu spielen, weil der Start-Stopp-Mechanismus nicht auf Exaktheit ausgerichtet ist.

Um diese zu erreichen, war die weitere Idee, einen Schrittmotor zu verwenden. Ein Schrittmotor kann durch einen elektromagnetischen Vorgang immer um den gleichen Winkel beziehungsweise Schritt rotieren. Das Problem hierbei ist seine langsame Geschwindigkeit, da eine Gitarrensaite nur durch eine schnelle Bewegung angeschlagen werden kann. Zudem ist der zeitliche Abstand zwischen den einzelnen Tönen zu gross, wenn sich der Motor mit nur einem befestigten Plektrum bei jedem Ton um 360 Grad drehen muss. Bei der Verwendung mehrerer Plektren, die an dem Rotor des Schrittmotors befestigt sind, leidet die Präzision stark, da es nicht möglich ist, mehrere Plektren derart präzise an demselben Rotor zu montieren, dass ein gleichmässiger Anschlag zwischen den Tönen gewährleistet ist.

Es hat sich damit herauskristallisiert, dass ein drehendes Modul nicht das richtige ist, um eine Saite anzuschlagen. Als geeigneter erwiesen sich lineare Magnete. Linearmagnete oder auch Hubmagnete können nur Translationsbewegungen ausführen. Sie bewegen sich also um eine gewisse Strecke nach vorne. Auch hier werden die Bewegungen durch ein Magnetfeld herbeigeführt, welche in diesem Fall aber das Metallstäbchen im Innern der Spule nach vorne schnellen lassen. Es entsteht eine schnelle, ruckartige Bewegung. Bei kleinen Modellen enthält der Magnet oft nur eine Spule, welche ein Magnetfeld erzeugt. Und der Anker wird nach Wegfall der Spannung mit Hilfe einer Feder zurück in ihre Ausgangsposition geschoben [6]. Zudem beträgt bei kleinen Modellen der zurückgelegte Hubweg nur wenige Millimeter bis Zentimeter. Solche Linearmagnete sollen nun über dem Gitarrenkorpus rechtwinklig zur Saite befestigte Holzstäbe, an welchen jeweils ein Plektrum befestigt ist, um diesen kleinen Weg verschieben. Der Vorteil hierbei liegt auch darin, dass die Konstruktion nicht direkt über den Saiten gebaut werden muss und sehr schmal ist, weshalb sie gut mehrere Male nebeneinander gebaut werden kann.

2.2.2.1.2 Erste Experimente zum Zupfautomat

Nachdem die Gitarre mit Filzpolsterung auf ein Brett montiert worden war, wurden verschiedene Linearmagnete evaluiert, indem sie provisorisch neben der Gitarre befestigt wurden. Über den erwähnten Stab mit dem Plektrum schlagen sie die Saite an. Dabei zeigt sich, dass mancher Magnet zu schwach ist und das Plektrum an der Saite hängen bleibt. Zudem erweist sich als grosses Problem, dass die Magnete bei längerem Verharren unter Strom sehr heiss werden. Als Lösung bietet sich die Verwendung eines bistabilen Magneten an, der in zwei unterschiedlichen Positionen stabil ist und nur mit einem kurzen Stromimpuls von 50 Millisekunden von der einen in die andere Position wechselt [7]. Dazu muss jeweils die Stromrichtung umgepolt werden, was eine aufwendige elektronische Schaltung bedingt, welche ich später noch erläutern werde.

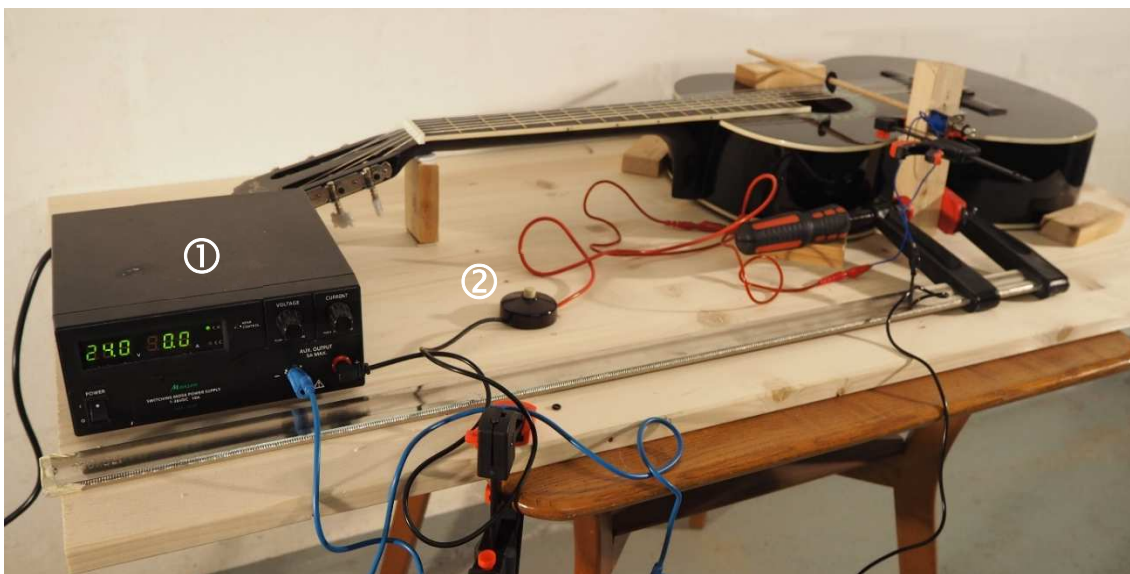


Abbildung 3: Experiment-Aufbau zum Testen der Linearmagnete, eigene Abbildung

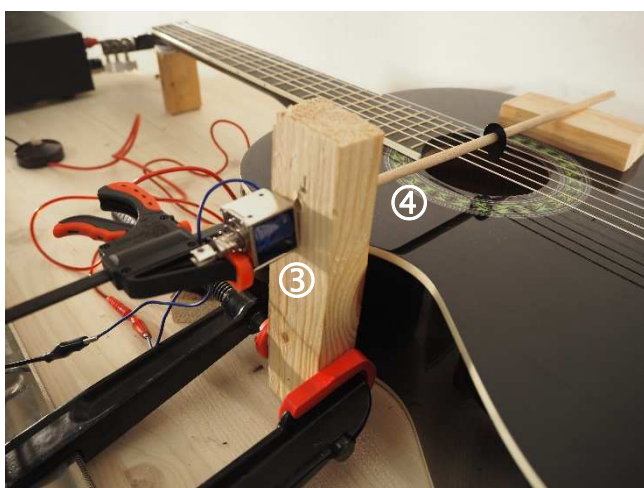


Abbildung 4: Nahaufnahme Linearmagnet, eigene Abbildung

1. Spannungsquelle
2. Schalter
3. Linearmagnet
4. Holzstab mit Plektrum

Das Problem mit der Hitze war durch den neuen Magneten allerdings nicht verschwunden, da der bistabile Magnet bei vielen, schnellen Betätigungen auch heiss wird. Um diesem Problem entgegen zu wirken, wurden die Magnete mit Wärmeleitpaste

auf eine Aluminiumplatte montiert. Dadurch kann sich die Wärme besser verteilen und es ist einfacher, gegebenenfalls die Platte extern zu kühlen.

Weitere Probleme tauchten auf:

Der Stab, der das Plektrum festhält und vom bistabilen Linearmagneten hin und her geschoben wird, muss befestigt werden, darf aber zugleich durch die Halterung nicht zu viel Reibung abbekommen, da sonst zu viel Kraft verloren geht und der Magnet die Saite nicht mehr zupfen kann.

Da der Hubweg nur sechs Millimeter beträgt, hat das Plektrum die Möglichkeit, sich nur an die Saite zu drücken und sich zu verbiegen, anstatt die Saite anzuschlagen und auf die andere Seite der Saite zu gehen (siehe Abb. 5).

Auch die Härte des Plektrums hat auf dieses Verhalten einen Einfluss. Ein sehr hartes Plektrum verbiegt sich kaum, ein sehr weiches verbiegt sich sehr leicht. Hierbei ist auch die Höhe des Plektrums entscheidend. Ist das Plektrum zu tief befestigt, drückt das Plektrum die Saite weg (siehe Abb. 6).

Da der Stab drehbar ist, kann er sich in die für ihn kraftsparendste Lage bringen, was, wenn das Plektrum herkömmlich mit der Spitze benutzt wird, dazu führt, dass sich der Stab mit dem Plektrum so wendet, dass das Plektrum die Saite nicht mehr erreicht (siehe Abb. 7).



Abbildung 5: Das Plektrum biegt sich, eigene Abbildung



Abbildung 6: Das Plektrum drückt die Saite weg, eigene Abbildung



Abbildung 7: Das Plektrum dreht sich von der Saite weg, eigene Abbildung

2.2.2.1.3 Verbesserung des Zupfautomaten

Bei den Experimenten zum Zupfautomaten wurde klar, dass die Justierung eine sehr grosse Rolle spielt. Um diese zu ermöglichen, wurde eine Holzkonstruktion wenige Millimeter über dem Gitarrenkorpus befestigt. Den Abstand braucht es, um den Klang der Gitarre nicht negativ zu beeinflussen. Mit den in Abbildung 3-5 verwendeten Holzstäbchen und einer Führung auf der anderen Seite wurden dieselben Experimente noch einmal durchgeführt. Es zeigte sich, dass so nicht die nötige Präzision erreicht werden konnte, geschweige denn eine funktionsfähige Justierung möglich ist. Deshalb musste eine ganz neue Variante gesucht werden.



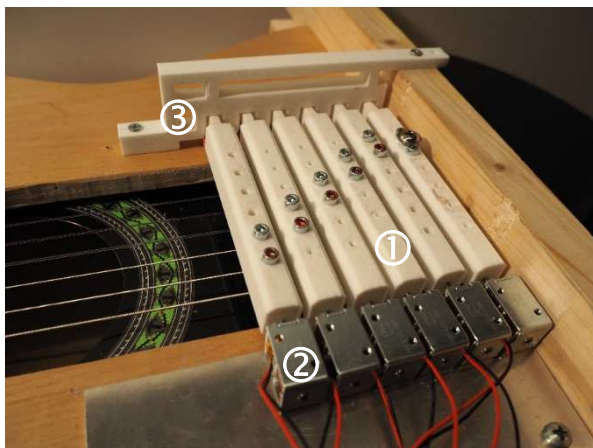
Abbildung 9: Konstruktion über dem Gitarrenkorpus, eigene Abbildung

Die Lösung war der 3D-Druck. Nach vielen Versuchen kam ein Plastikstäbchen, hier Stängel genannt, zustande, in welches ein ebenfalls selbst ausgedrucktes Plektrum eingesetzt werden kann. Der Hauptteil des Plektrums ist ein Prisma, welches ein gleichseitiges Dreieck mit einem Zwischenwinkel von 80 Grad als Grundfläche hat. Das Plektrum wird am langen Befestigungsteil in die Aussparung im Stängel eingesetzt und mit zwei Schrauben höhenverstellbar befestigt.



Abbildung 8: Erster Prototyp des 3D-Plektrums, eigene Aufnahme

Durch ein in den Stängel hineingebohrtes Gewinde kann dieser an den bistabilen Linearmagneten angeschraubt werden und durch leichtes Drehen nochmals mehr oder weniger weit hineingeschraubt und damit quer zu den Saiten justiert werden. Auf der anderen Seite werden die Stängel durch ein weiteres gedrucktes Teil, die Führungsleiste, geführt, um Auslenkungen zu verhindern. Durch diese Vorrichtung können die Saiten in beiden Ausgangslagen des Magneten frei schwingen.



1. Stängel
2. Bistabiler Linearmagnet
3. Führungsleiste

Abbildung 10: Zupfkonstruktion, eigene Aufnahme

2.2.2.2 Drückmechanismus

2.2.2.2.1 Erste Experimente zum Drückmechanismus

Um ein Lied spielen zu können, braucht es nicht nur einen Zupfautomaten, sondern es müssen auch die verschiedenen Bünde gedrückt werden. Um alle Töne spielen zu können, sind mindestens die ersten vier Bünde mit einem Mechanismus zu versehen, welcher die Saiten nach unten drückt und somit verkürzt. Die grösste Schwierigkeit ist, dass zwischen den verschiedenen Saiten sehr wenig Platz ist. Dies erschwert die Aufgabe sehr. Für die ersten Tests wurde der gleiche Linearmagnet benutzt, wie auch beim Zupfautomaten (siehe Abb. 4). Dieser hat zwar genug Kraft, doch erstreckt sich seine Breite über fast drei Saiten, was ihn für diese Funktion unbrauchbar macht. Stattdessen wurden kleinere Magnete gleicher Bauart getestet. Doch es erwies sich schnell als Problem, dass diese zu wenig Kraft haben, um die Saite nach unten zu drücken. Um den Druck, welchen die Magnete benötigen, um die Saite zu drücken, zu minimieren, benutzte ich von da an nur noch super weiche Saiten, die mit weniger Kraft gedrückt werden können.

Gesucht war also weiterhin ein Magnet, welcher eine kurze, etwa fünf Millimeter grosse translatorische Bewegung ausführt, genügend Kraft hat, eine Saite zu drücken und doch nicht breiter als ein Saitenabstand ist, was an dieser Stelle etwa ein Zentimeter ist. Solche Magnete liessen sich als Einzelanfertigung besorgen, allerdings zu horrenden Preisen. In den Tiefen des Internets waren dann aber dünne Magnete zu finden, welche ursprünglich für Stickmaschinen gedacht waren. Egal ob Überproduktion oder Ausschussware: Sie waren sehr billig zu haben. Dazu sind sie genug stark, um eine Saite zu drücken, allerdings nur, wenn sie anfangs einen kurzen Weg ohne Belastung haben. Sie erreichen ihre maximale Kraft also in der Mitte ihres Hubes. Es handelt sich um die dünnsten Magnete, welche zu finden waren, jedoch sind auch sie breiter als der Saitenabstand. Dies bedingt, dass die Linearmagnete in zwei versetzten Reihen platziert werden müssen, was zwar die Klangqualität verschlechtert, doch in diesem Fall die einzig mögliche Lösung ist. Um die Klangqualität möglichst aufrecht zu erhalten, ist an dem Teil des Magnetes, welcher auf die Saite kommt, ein gummiartiger Leim aufgetragen worden. Dieser dämpft sehr feine Vibrationen ab und verbessert somit unsauber klingende Töne.



Abbildung 11: Linearmagnet für den Drückautomaten, eigene Abbildung

2.2.2.2 Verbesserung des Drückmechanismus

Auch dieser Magnet wird bei längerer Betätigung sehr heiss. Deshalb wird auch hier eine Aluminiumplatte als Wärmeleiter verwendet, sodass sich die Wärme verteilen kann und durch die grössere Oberfläche sich die Abkühlung beschleunigt. Dieser Entscheid wurde dadurch gestützt, dass das Aluminium durch die zwei Muttern, welche am Gewinde des Magnets vorhanden sind, befestigt werden kann (siehe Abb. 12 und 13).

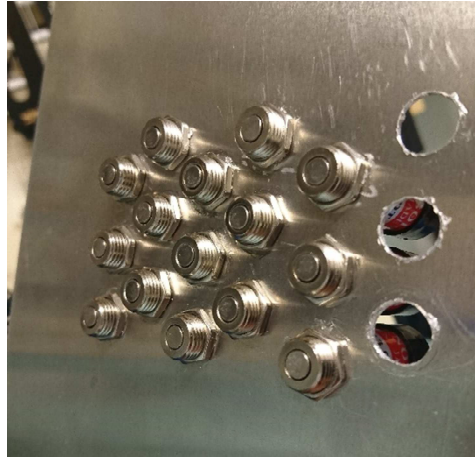


Abbildung 12: Lochplatte mit Linearmagneten von unten, eigene Abbildung

Über eine transparente Folie konnte der Ort, bei dem der Magnet drücken sollte, genau bestimmt werden und auf das Aluminium abgetragen werden. Die fertige Lochplatte wurde probeweise mit Hölzern auf die richtige Höhe über das Griffbrett montiert.



Abbildung 13: Mit Hölzern höhenangepasste Lochplatte, eigene Abbildung

Schnell wurde klar, dass die Höhe dieser Platte sehr entscheidend ist und im Millimeterbereich einzustellen sein muss. Erschwert wird dies durch die Saiten, welche nicht exakt parallel zum Griffbrett verlaufen. Der Abstand zwischen Saiten und Griffbrett nimmt zum Gitarrenkorpus hin zu. Das bedeutet, die Linearmagnete müssen einen unterschiedlich langen Weg zurücklegen, und, fast entscheidender, sie treffen in Bezug auf den ganzen zurückzulegenden Weg schon vor der Hälfte auf die Saite, sie haben also

beim Auftreffen auf die Saite noch nicht ihre maximale Kraft. Zur Bestimmung der Höhe der Magnete sind folgende drei Dinge zu beachten:

1. Das Verhältnis von Griffbrett/Saite und Saite/Magnet muss etwa gleich gross sein, da der Magnet sonst nicht genug Kraft hat, die Saite zu drücken.
2. Der Magnet darf nicht zu weit oben sein, da sonst der Hub zu klein ist und er nicht bis auf das Griffbrett kommt.
3. Er darf aber auch nicht zu tief sein, da sonst beim Spielen der Leersaite die Saite an die Magnete kommt und abgedämpft wird.



Abbildung 14: Lochplatte für Linearmagnete, eigene Abbildung

Um die Höhenverstellbarkeit zu gewährleisten, wurden Gewinde in die Gitarrenhalterung gemacht, wo geköpfte Schrauben eingeführt wurden. Mit zwei Muttern kann man so die Aluminiumplatte in der Höhe verstellen. Die Mutter unterhalb der Platte dient zum Justieren der Höhe, die obere wird zur Befestigung angezogen, damit die Magnete nicht das Aluminium nach oben stossen.

2.2.3 Software

2.2.3.1 Programmierung

Das Ziel, ein Lied vom Computer aus zu spielen, benötigt ein entsprechendes Programm, welches dies ermöglicht. Diese muss nun programmiert werden. Die Magnete des Drückautomaten habe ich dafür von eins bis 24 durchnummeriert, angefangen bei der tiefen E-Saite der Tonhöhe nach aufwärts.

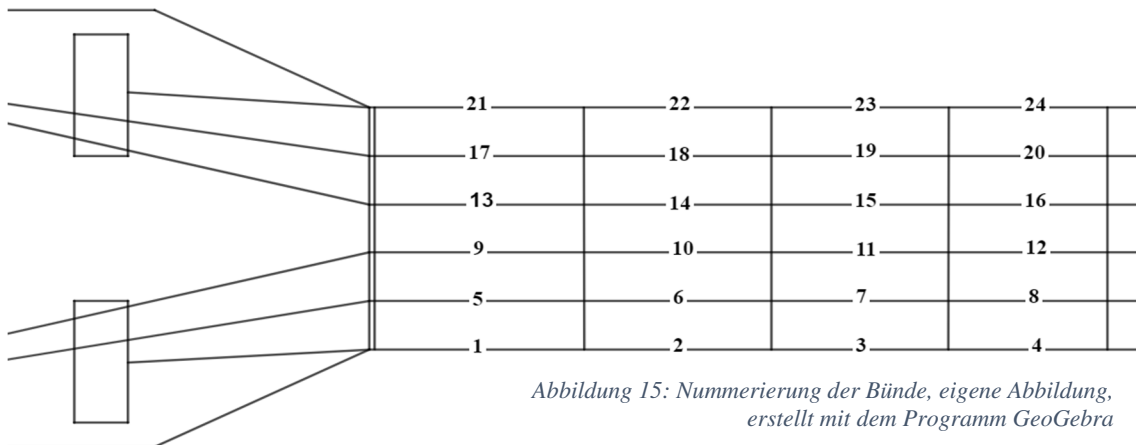


Abbildung 15: Nummerierung der Bünde, eigene Abbildung, erstellt mit dem Programm GeoGebra

Am naheliegendsten wäre es, in einem Arduino-Programm jeden einzelnen Magnet zur richtigen Zeit anzuschalten, eine Verzögerung einzubauen, und den Magnet wieder abzustellen. Um dies zu veranschaulichen, wählte ich das Beispiel der berühmten Melodie des «Big Ben», welche auch von vielen Schulglocken bekannt sein dürfte. Da ich hier nur den Drückmechanismus berücksichtige, habe ich für die Melodie die E-Tonlage gewählt, um Leersaiten zu vermeiden.

```
void loop() {
  digitalWrite(13, HIGH);
  delay(30);
  digitalWrite(13, LOW);
  digitalWrite(10, HIGH);
  delay(30);
  digitalWrite(10, LOW);
  digitalWrite(12, HIGH);
  delay(50);
  digitalWrite(12, LOW);
  digitalWrite(6, HIGH);
  delay(50);
  digitalWrite(6, LOW);
  delay(10);
  digitalWrite(6, HIGH);
  delay(30);
  digitalWrite(6, LOW);
  digitalWrite(12, HIGH);
  delay(30);
  digitalWrite(12, LOW);
  digitalWrite(13, HIGH);
  delay(30);
  digitalWrite(13, LOW);
  digitalWrite(10, HIGH);
  delay(50);
  digitalWrite(10, LOW);
}
```

Abbildung 16: Arduino «Big Ben»-Melodie, eigene Abbildung, Screenshot aus Arduino

Unpraktisch daran ist, dass das Einspeichern einer Melodie auf diese Weise sehr mühsam und aufwendig ist. Mit dem Willen, mehrere Lieder einfacher spielen zu können, musste also eine neue Lösung gefunden werden. Es entstand die Idee, Lieder in Zahlen umzuschreiben und sie dann mit dem Arduino abspielen zu lassen. Dabei benutzte ich ein sehr altes Format zum Notieren von Liedern für die Gitarre: die Tabulatur. Eine Tabulatur besitzt nicht die herkömmlichen fünf Notenlinien, sie besitzt sechs Linien, wovon jede für eine Saite der Gitarre steht. Was schon manch einem Gitarrenschüler zu Nutze kam, konnte hier verwendet werden: Die zu spielenden Töne werden nicht als Noten dargestellt, sondern als Zahl, welche den Bund auf der entsprechenden Saite angibt. Zur Veranschaulichung die Melodie des «Big Bens» in Noten und Tabulatur in Abbildung 17.

Abbildung 17: «Big Ben»-Melodie in Notenschrift und Tabulatur, eigene Darstellung, erstellt mit dem Programm «Power Tab Editor 1.7»

Aus dieser Tabulatur kann nun jeweils eine sechsstellige Zahl abgelesen werden, nämlich vertikal von unten nach oben, also von der tiefen zur hohen E-Saite. Jede Ziffer der Zahl steht also für eine Saite und sagt aus, welcher Bund auf ihr gedrückt werden soll. Um auch die Tonlängen zu berücksichtigen, wird das oben erklärte Verfahren im Metrum des kleinsten Notenwerts durchgeführt. Im Falle von punktierten Noten handelt es sich um den Wert, um den die Note durch den Punkt verlängert wird. Im Beispiel des «Big Ben» ist dies ein Viertel. Damit auch Leersaiten direkt mit der Ziffer zur Saite signalisiert werden können, braucht es pro Saite einen Zustand mehr. Die Zuteilung der Ziffern lautet wie folgt:

- 0 → die Saite wird nicht gespielt
- 1 → es wird die Leersaite gespielt, also die Saite, ohne dass ein Bund gedrückt wird
- 2 → die Saite wird mit dem 1. Bund gedrückt gespielt
- 3 → die Saite wird mit dem 2. Bund gedrückt gespielt
- 4 → die Saite wird mit dem 3. Bund gedrückt gespielt
- 5 → die Saite wird mit dem 4. Bund gedrückt gespielt
- 6 → der Bund wird gelöst, es wird keine Saite gezupft

Damit führende Nullen beim Rechnen und Schicken der Daten nicht wegfallen, wird jeder Zahl eine Eins vorangestellt. Alle Zahlen werden in eine einzelne Zeile geschrieben und in einer Textdatei gespeichert.

Im Beispiel der «Big Ben»-Melodie sehen die Zahlen dann wie folgt aus:

- | | |
|---|--|
| <ul style="list-style-type: none"> 1000200 1003600 1005000 1036000 1000000 1000000 1030000 1065000 1006200 1003600 1000000 1000000 1006000 | <p>Eine Sechs wird dabei nur benötigt, falls während des Anhaltens des Tones oder direkt nach dem Ton kein anderer Ton auf derselben Saite gespielt wird. Ansonsten wird der Bund des zuletzt gespielten Tones automatisch gelöst.</p> <p>Bei den ersten Experimenten, bei denen im Arduino-Programm selbst ein paar Griffe definiert wurden, war das ganze Rechnen dem Arduino überlassen. Aus den siebenstelligen Zahlen wird für jeden Bund der entsprechende Pin berechnet, welcher angesteuert werden soll. Es war schnell offensichtlich, dass der interne Speicher des Arduinos für ganze Lieder zu klein ist. Deshalb war schon früh klar, dass die Zahlen extern gespeichert werden müssen und dem Arduino geschickt werden müssen, beziehungsweise der Arduino sie sich holt. Dafür gab es zwei verschiedene Ansätze: Der eine, bei dem sich der Arduino von einer SD-Karte, auf der</p> |
|---|--|

die Zahlen gespeichert sind, die Zahlen holt. Der Vorteil dabei ist, dass es sehr platzsparend ist, allerdings muss die SD-Karte bei neuen Liedern an einen Computer angeschlossen werden und der Arduino hat sehr viel zu rechnen. Die andere Variante ist, die Zahlen auf einem Computer zu speichern, der sie an den Arduino schickt. Dabei kann viel Rechenleistung auf den Computer verschoben werden, setzt aber voraus, dass immer ein Computer angeschlossen ist. Die letztere Variante gefällt mir besser, da durch die Möglichkeit, Zahlen auf dem PC-Monitor auszugeben, viel mehr Kontrolle über den Ablauf des Programms vorhanden ist und Fehler schneller behoben werden können.

Dazu ist es allerdings nötig, dass auf dem Computer parallel zum Arduino ein Programm laufen muss, welches dem Arduino die Daten schickt, da dieser sich nicht selbst in den Tiefen des Computers zurechtfindet.

Diese Aufgabe sollte Python übernehmen, wie ich in Absprache mit Peter Rutschmann beschlossen hatte. Zum einen ist Python das Programm, welches wir im Informatikunterricht schon einmal behandelt haben, zum anderen bietet es, zumindest auf den ersten Blick, einfache Möglichkeiten, mit Arduino zu kommunizieren, indem man ein Modul zur Kommunikation mit Arduino importiert [8]. Python ist eine Programmiersprache, welche durch eine klare Syntax einfach zu erlernen ist [9]. Dazu ist Python wie Arduino gratis erhältlich. Da nun sowieso ein Programm auf dem Computer laufen muss, verschob ich die ganze Rechenarbeit auf den PC und schrieb das Arduino-Programm so um, dass es nun in Python lief. Dies dauerte eine Weile, brachte aber keine grossen Schwierigkeiten.

2.2.3.2 Serielle Schnittstelle

Die grösste Herausforderung war die serielle Schnittstelle, also die Kommunikation zwischen Python und Arduino. Anhand von ein paar Beispielen im Internet war es schnell möglich, Zahlen an den Arduino zu schicken und dort mit einer if-Abfrage zu erkennen. Allerdings liess sich mit ihnen nicht rechnen und sie waren nicht weiter zu verarbeiten. Da der serielle Anschluss des Arduinos während des Abspielens des Python-Programms von diesem beansprucht wird, kann der serielle Monitor, auf dem man direkt via Arduino Zeichen auf dem Computer ausgeben kann, nicht benutzt werden. Somit können nur Zeichen eingesehen werden, welche vom Arduino wieder an den PC zurückgeschickt und dort ausgegeben werden. Dies machte die Programmierung sehr aufwendig.

Mehrfache Datentypkonvertierung war notwendig, um die siebenstelligen Zahlen (in einzelnen Ziffern) über die serielle Schnittstelle zu übertragen und auf dem Arduino wieder zusammensetzen [10].

```
while (Serial.available() > 0)
{
  int inChar = Serial.read();
  if (isDigit(inChar))
  {
    // Konvertiert das empfangene Byte in ein char und fügt es dem String hinzu
    inString += (char)inChar;
  }
  // Bei einer neuen Zeile wird der String geschrieben, dann der Wert des Strings:
  if (inChar == '\n')
  {
    int pin = inString.toInt()+1; //erster Pin dient zur Kommunikation, deshalb +1
```

Abbildung 18: Arduino Kommunikation, eigene Abbildung, Bildschirmfoto aus Arduino

2.2.3.3 Python

```
47 | if bund1 != bund2 and bund2 > 0 and bund2 !=5 and akг[saitе] >=0: #gelöst, gedrückt
48 |     print ('Pinschickenloesen: %r' %((saitе-1)*4 + akг[saitе]))
49 |     print "Bund wird geloest, neuer Bund wird gedrueckt" #Saite wird im 1. bis 4. Bund gedrückt
50 |     print "Akg: %r" %(akг[saitе])
51 |     pinschicken((saitе-1)*4 + akг[saitе]) #bund1 lösen, auf akг zurückgreifen
52 |
53 |     print ('Pinschickendruecken: %r' %(((saitе-1)*4 + bund2)+100))
54 |     pinschicken(((saitе-1)*4 + bund2)+100) #bund2 drücken
55 |     akг[saitе] = bund2 #akг neu setzen
56 |
57 | elif bund1 != bund2 and bund2 > 0 and bund2 !=5: #gedrückt
58 |     print "Bund wird neu gedrueckt"
59 |     print ('Pinschickendruecken: %r' %(((saitе-1)*4 + bund2)+100))
60 |     pinschicken(((saitе-1)*4 + bund2)+100) #bund2 drücken
61 |     akг[saitе] = bund2 #akг neu setzen
62 |
63 | elif bund2 == 0 and akг[saitе] > 0 or bund2 == 5: #gelöst
64 |     print "Bund wird geloest, evtl eine Leersaite gezupft"
65 |     print ('Pinschickenloesen: %r' %((saitе-1)*4 + akг[saitе]))
66 |     pinschicken((saitе-1)*4 + akг[saitе]) #Bund loesen, es wird neu kein Bund gedrückt
67 |     akг[saitе] = 0 #akг wird auf 0 gesetzt
68 |
69 | elif bund2 == -1 and akг[saitе] >= 1: #gedrückt lassen
70 |     print "Bund bleibt gedrueckt"
71 |
72 | else: #weiterhin nichts drücken
73 |     print "Bund wird weiterhin nicht gedrueckt"
```

Abbildung 19: If-Schleife in Python zum Lösen und Drücken der Bünde, eigene Abbildung, Screenshot aus dem Programm Notepad ++

Durch das Verschieben des Programms in Python ist es nun möglich, die nächste Zahl mit der vorherigen abzugleichen. Dazu werden die vier Fälle

- Bund lösen und neuen Bund drücken,
- Bund lösen,
- Bund drücken und
- nichts tun

unterschieden, wobei der unterste Fall zur Kontrolle in «den vorherigen Bund gedrückt lassen» und «es wird weiterhin nichts oder eine Leersaite gespielt» unterteilt wird. Durch das Abspeichern des aktuell gedrückten Bundes im Array «Akg» (Akg steht für «Aktuell gedrückt») ist es auch möglich, einen Bund, der lange gespielt wurde, zu lösen, ohne auf die Zahl zurückgreifen zu müssen, welche den Bund gedrückt hat. Analog wird auch beim Zupfautomaten die aktuelle Position im Array «soz» (soz steht für «stossen oder ziehen») abgespeichert, damit klar ist, ob der bistabile Magnet als nächstes nach vorne stossen oder nach hinten ziehen muss.

2.2.4 Ansteuerung der Magnete

Das Arduino-Board läuft mit einer Spannung von fünf Volt, was für alle Magnete, damit sie genügend Kraft für ihre jeweilige Aufgabe haben, zu wenig ist. Deshalb suchte ich nach einer Lösung, um aus den anfänglichen fünf Volt, zwölf oder gar 24 Volt zu machen. Hierfür führte ich erste Experimente mit einem mechanischen Relais-Modul durch. Bei diesem wird ein Laststromkreis durch einen Elektromagneten geschlossen, welcher mit einem Steuerstromkreis gesteuert wird [11]. Die beiden Stromkreise können dadurch unabhängig voneinander verschiedene Spannungen haben. Ein Dorn im Auge war jedoch, dass diese Relais bei jeder Betätigung ein lautes Klacken verursachten.

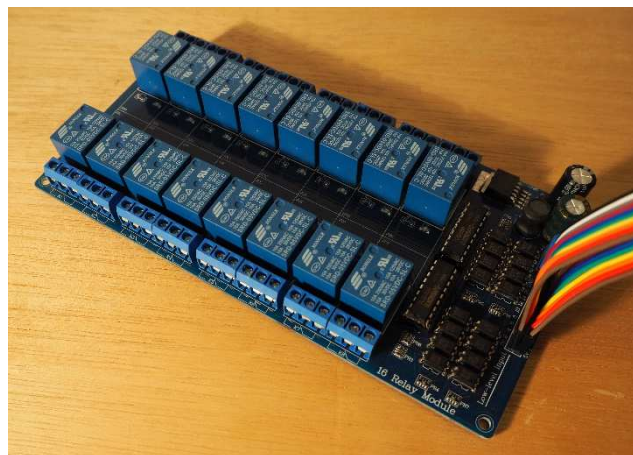


Abbildung 20: 16-Kanal Relais-Modul, eigene Abbildung

Die beiden Stromkreise können dadurch unabhängig voneinander verschiedene Spannungen haben. Ein Dorn im Auge war jedoch, dass diese Relais bei jeder Betätigung ein lautes Klacken verursachten.

Mit dem kurzen Signal und dem Umpolen des Stroms zum Betrieb der bistabilen Magneten wurde der Lärm der Relais endgültig zu laut. Anstelle von diesen verwendete ich nun die wesentlich kleineren ICs. Die englische Abkürzung IC steht im Deutschen für Integrierter Schaltkreis. Die ICs ermöglichen es, eine Umpolung des Stroms sehr platzsparend und leise vorzunehmen. Dabei wird der Stromkreis des Arduinos (5 Volt) und jener der Magnete (24 Volt) über Optokoppler getrennt.

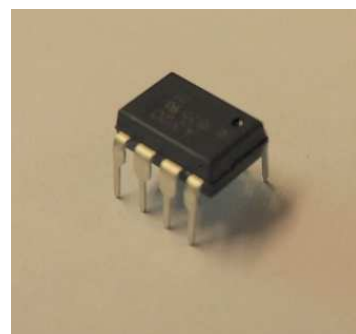


Abbildung 21: IC, eigene Abbildung

Die folgende Abbildung zeigt den Aufbau der verwendeten ICs:

FUNCTIONAL BLOCK DIAGRAM

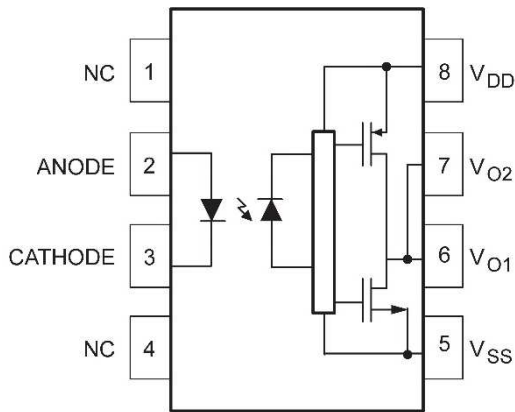
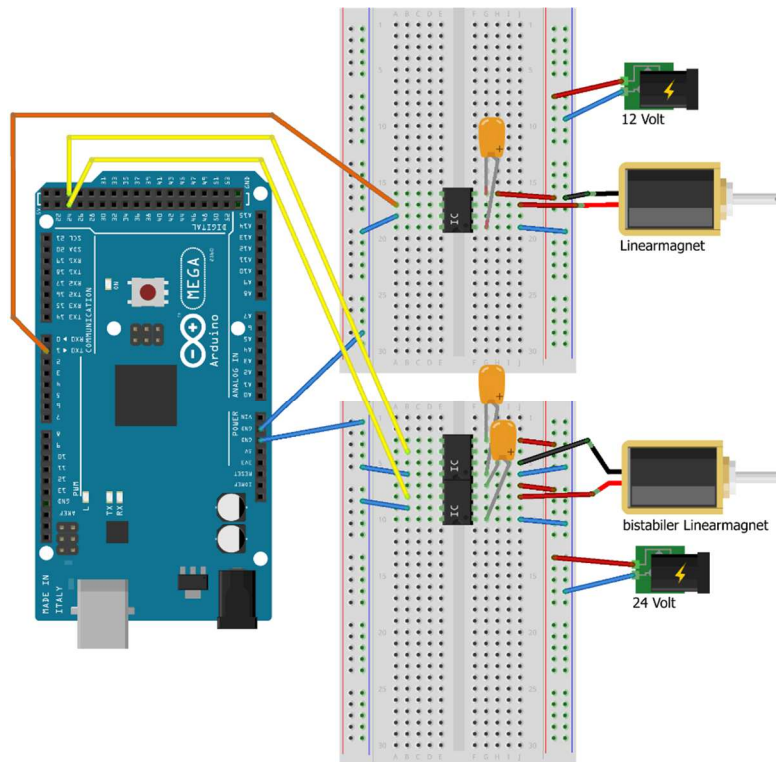


Abbildung 22: Schaltung de ICs [12]

Die Verkabelung für die drückenden Magnete folgt nach diesem Schema:

| | |
|---|--|
| 1 | |
| 2 | Pluspol des Arduinos |
| 3 | Ground des Arduinos |
| 4 | |
| 5 | Pluspol der 12 Volt/24 Volt Spannung |
| 6 | An einen der beiden Pins kommt jeweils das eine Kabel des Magneten |
| 7 | Ground der 12 Volt/24 Volt Spannung |
| 8 | |

Für die zupfenden bistabilen Magnete, bei welchen die Stromrichtung ändern muss, werden jeweils zwei ICs benötigt. Der Unterschied zu den drückenden Linearmagneten ist, dass hier beide Kabel mit je einem IC verbunden sein müssen. Bei den Magneten des Drückmechanismus ist das eine Kabel mit dem Ground der 24 beziehungsweise 12 Volt Seite verbunden. Sichtbar wird dies im Schaltplan in Abbildung 23, welcher einen Druckmagneten und einen Zupfmagneten und deren Ansteuerung darstellt:



fritzing

Abbildung 23: Schaltplan, eigene Abbildung, erstellt mit dem Programm Fritzing

2.2.5 Endprodukt

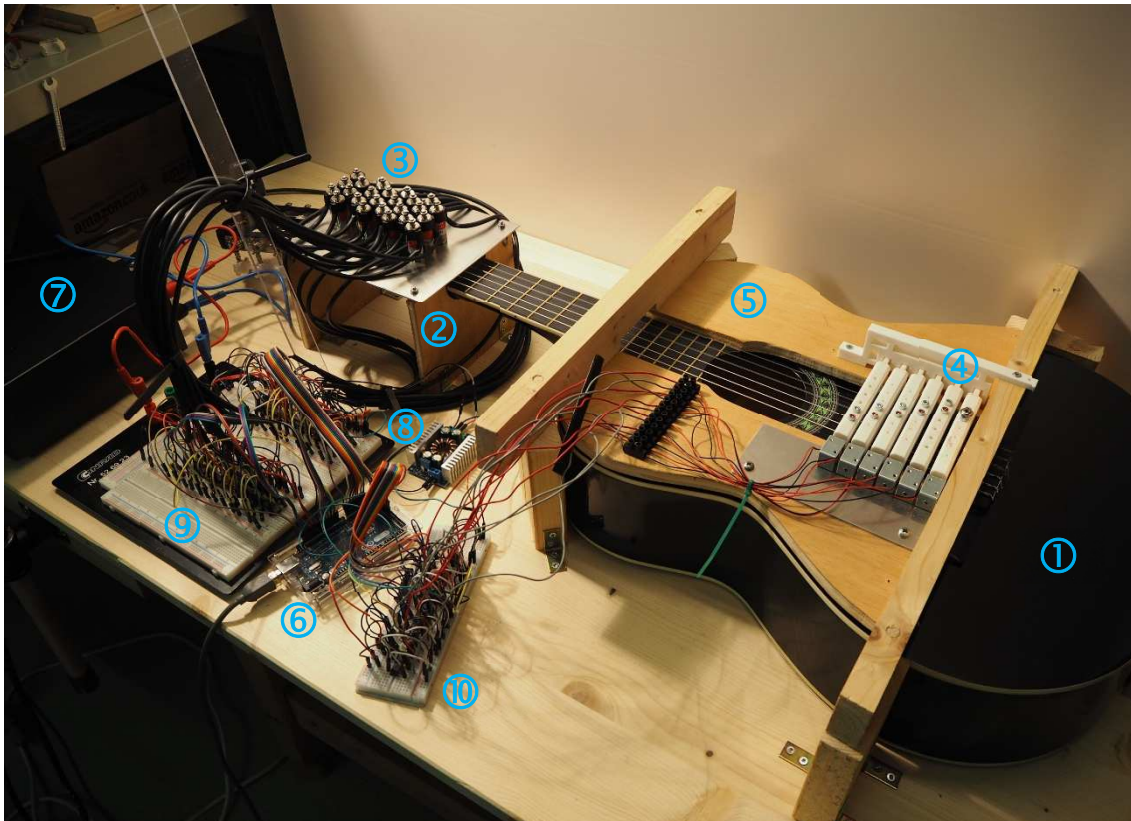


Abbildung 24: Das Endprodukt, eigene Abbildung

1. Gitarre
2. Gitarrenhalterung
3. Drückmechanismus
4. Zupfautomat
5. Holzkonstruktion zur Befestigung des Zupfautomaten
6. Arduino Mega mit Kabel zum Computer
7. Spannungsquelle
8. Spannungsumwandler von 12 auf 24 Volt
9. Verkabelung in Steckbrett für Drückmechanismus
10. Verkabelung in Steckbrett für Zupfautomat

Die fertige Gitarre kann komplizierte Lieder in einem sehr hohen Tempo wiedergeben. Auch bei sehr schnellem Spielen sind die Melodien erkennbar und werden fehlerfrei gespielt. Das Einspeichern neuer Lieder ist zwar aufwendig, doch mit ein wenig Übung immer schneller zu bewerkstelligen. Dazu sind Fehler leicht zu finden, da durch die Zeilenanzahl leicht herausgefunden werden kann, wo man sich im Lied befindet.

Bei hohen Tempi wird jedoch bei Tönen, welche gleichzeitig spielen sollten, eine kleine Verzögerung bemerkbar. Dies ist der Fall, weil der bistabile Linearmagnet 50 Millisekunden Strom braucht, um die Position zu wechseln. Der Arduino bekommt das nächste Signal erst, wenn der letzte Zupfmagnet betätigt wurde. Bei langsamen Liedern fällt dies aber kaum auf.

Des Weiteren verursachen die bistabilen Linearmagnete bei jeder Betätigung einen lauten Ton. Dieser verstärkt sich, da die Magnete mit der Konstruktion über der Gitarre verbunden sind und diese somit als Klangkörper agiert. Auch der Drückmechanismus scheppert ab und zu, da hier nur die ganze Platte und nicht die einzelnen Magnete höhenverstellbar sind.

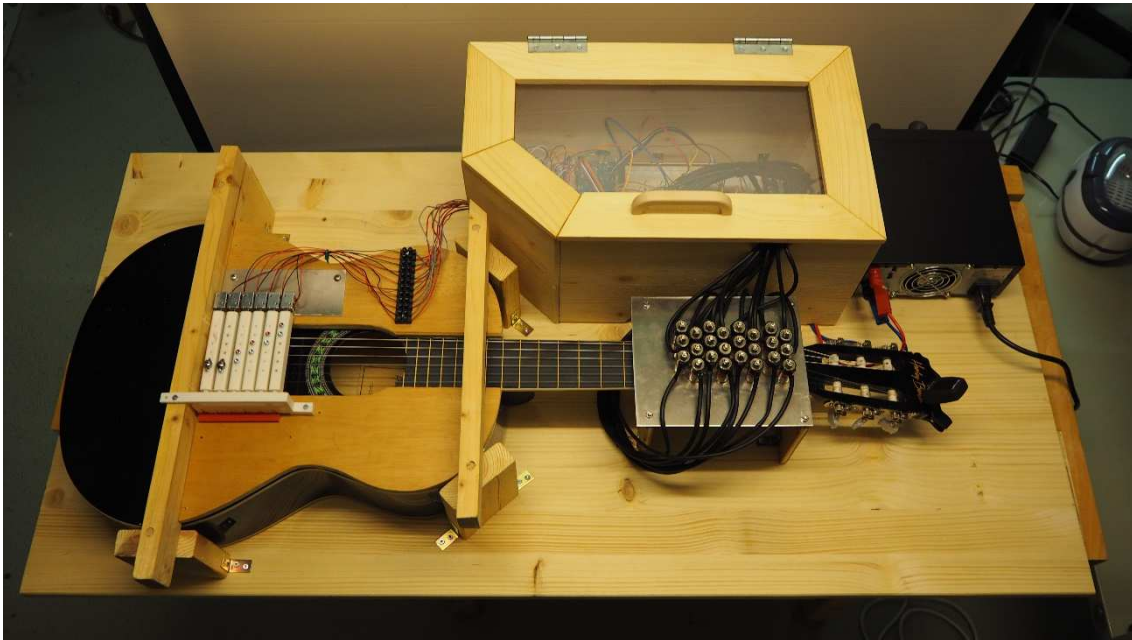


Abbildung 25: Endprodukt mit Schutz um Elektronik, eigene Abbildung

Damit die Elektronik, vor allem die in die Steckbretter gesteckten Kabel, nicht beschädigt werden, wird sie von einer Holzgehäuse umbaut. Sein Deckel, welcher ein Fenster enthält, kann geöffnet werden.

2.2.6 Fazit

Im Rückblick war die Konstruktion der selbstspielenden Gitarre ein extrem spannendes und vielseitiges, aber auch aufwendiges Projekt. Dass die Realisierung am Schluss tatsächlich funktioniert hat, freut mich sehr.

Das Vorgehen, die unterschiedlichen Arbeitsschritte je einzeln anzugehen, war sinnvoll. Allerdings mussten im Prozess alle Schritte parallel vorangetrieben werden, da in jedem Bereich sehr viel Experimentieren notwendig war, was unerwartet viel Zeit erforderte. Zudem war es anfangs nicht klar, welche Schritte viel Zeit benötigen würden und welche schnell erledigt werden können. Beispielsweise war zuerst der Drückmechanismus das grosse Sorgenkind. Er konnte dann aber deutlich schneller und einfacher als der Zupfmechanismus realisiert werden.

Das Arbeiten mit den verschiedenen Materialien, zuerst Holz, dann Metall, und zum Schluss der unvorhersehbare Gebrauch des Filaments, des Grundbaustoffes des 3D-Druckers, war abwechslungsreich, da sie unterschiedliche Behandlung erforderten. Es bereitete mir viel Freude, unterschiedliche Erfahrungen zu machen. Dazu gehört auch diejenige des Programmierens.

Bei Entscheidungen, welches nun der nächste Schritt oder das nächste Experiment sein soll, sowie bei Fragen zur besten Lösung eines anstehenden Problems, wäre oft ein ganzes

Team hilfreich gewesen, auch um mehrere Experimente gleichzeitig durchzuführen, und damit mehr Wissen in kürzerer Zeit zu generieren. Es ist also nachvollziehbar, dass heutzutage in Teams geforscht wird.

2.3 Weiterführende Überlegungen

Nachdem das Grundmodell der selbstspielenden Gitarre funktioniert, zeigen sich verschiedene Punkte, die noch perfektioniert werden könnten. Eine Möglichkeit, die Hardware für das Instrument zu verbessern, wäre das Bauen eines Dämpfers für jede Saite. Um einiges aufwendiger wäre, musikalische Feinheiten des klassischen Gitarrespielens nachzuahmen, zum Beispiel ein «Pull-off», also das Spielen eines Tones nur durch Wegziehen eines zuvor gedrückten Bundes, ein Vibrato, also das leichte Vibrieren bei der drückenden Hand, oder ein Beding, also das Verzerren eines Tones durch Hoch- und Runter-Schieben der Saite am Bundstäbchen mit dem drückenden Finger.

Im Bereich des Programmierens liesse sich am einfachsten wohl noch ein «Hammer-On», das Spielen einer Saite nur durch das Drücken des Bundes, realisieren. Dies wäre möglich, da die Magnete des Drückautomaten schon jetzt beim Aufprall die Saite ein wenig in Schwingung versetzen. Weiter wäre es natürlich gediegen, direkt aus Noten die Zahlen zu generieren. Ausserdem könnten musikalische Feinheiten wie Fermaten markiert werden. Diese Perfektionierungen liessen sich aber in der vorgegebenen Zeit und im Rahmen meiner Möglichkeiten nicht realisieren.

3 Schlusswort

«Autoplay – Konstruktion einer selbstspielenden Gitarre» war ein erfolgreiches Projekt. Das anfangs gesetzte Ziel, die Gitarre verschiedene Lieder über einen Computer spielen zu lassen, wurde erreicht. Durch eine selbstgeschriebene Software in Python gelang es, verschiedene Linearmagnete anzusteuern. Diese Magnete drücken die entsprechenden Bünde und zupfen die entsprechenden Saiten.

Der grösste Nachteil liegt darin, dass die bistabilen Linearmagnete einen sehr grossen Lärm verursachen. Dies konnte allerdings nicht vermieden werden, da ich mich im Prozess nach einer gewissen Zeit auf einen Magneten konzentrieren musste, und dieser in den Tests nicht die gleiche Lautstärke erreicht hatte.

Die selbstspielende Gitarre erfüllt also ihren Zweck, kann aber einen guten Gitarrenspieler nicht ersetzen. Das computergesteuerte Spielen eines Instrumentes wird für mich nie das gleiche Niveau bekommen wie das manuelle Spielen. Interpretationen, Feingefühl, aber auch Fehler gehören für mich zum Musizieren dazu. Das manuelle Spielen eines Instruments ist für mich unantastbar und es gilt jedem, der sein Instrument beherrscht, mein höchster Respekt.

4 Quellenverzeichnis

Bücher:

- [1] Bartmann, Erik: *Die elektronische Welt mit Arduino entdecken*, O'Reilly Verlag, Köln, 3. korrigierte Auflage 2012.
- [2] Hocker, Jürgen: *Faszination Player Piano: das selbstspielende Klavier von den Anfängen bis zur Gegenwart*, Serie: Edition Bochinsky, PPV Medien, Bergkirchen, 2009.
- [3] Jüttemann, Herbert: *Mechanische Musikinstrumente: Einführung in Technik und Geschichte*, Serie: Fachbuchreihe Das Musikinstrument; Band 45, Bochinsky cop., Frankfurt am Main, 1987.
- [4] Saxer, Marion: *Spiel (mit) der Maschine: Musikalische Medienpraxis in der Frühzeit von Phonographie, Reproduktionsklavier, Film und Radio*, Serie: Musik und Klangkultur, Transcript, Bielefeld, 2016.

Zeitschriften:

- [5] Das mechanische Musikinstrument, 11. Jahrgang, Nr. 37, Dezember 1985.

Internetseiten:

- [6] Technische Erläuterungen zur Auswahl und Anwendung von Gleichstrommagneten, http://www.islikermagnete.ch/download/technische_erlaeuterungen.pdf, 5.12.2018.
- [7] Doppelspulmagnete, <http://www.magnetbasics.de/hubmagnete/doppelspulen.htm>, 1.10.2018.
- [8] Python Serial Port Extention, <https://sourceforge.net/projects/pyserial/>, 17.10.2018.
- [9] <https://www.bigdata-insider.de/was-ist-python-a-730480/>, 22.11.2018.
- [10] Arduino Python Communication Via USB 4 Steps, <https://arduinobasics.blogspot.com/2012/05/reading-from-text-file-and-sending-to.html>, 17.10.2018.
- [11] <https://de.wikipedia.org/wiki/Relais>, 31.11.2018.
- [12] Fairchild's high speed octocouplers, <https://www.fairchildsemi.com/collateral/FOD3120-and-FOD3150-High-Speed-MOSFET-IGBT-Gate-Drive-Optocouplers.pdf>, 24.11.2018.

Internetseiten zur Materialbeschaffung:

Bistabile Linearmagnete: <https://www.conrad.com/ce/de/product/507766/Hubmagnet-drueckend-ziehend-2-N-8-N-24-VDC-35-W-Tremba-HMB-1513001-24VDC>, 29.8.2018.

Drückmagnete: <https://de.aliexpress.com/item/Computer-embroidery-machine-parts-Clutch-electromagnet-install-equipment-electromagnet/32827037127.html?spm=a2g0s.9042311.0.0.27424c4dCtr5yM>, 27.8.2018.

ICs: <https://de.aliexpress.com/item/Free-shipping-10pcs-lot-Optocouplers-HCPL-3120-HCPL3120-3120-A3120-DIP8-new-original/32827069005.html?spm=a2g0s.9042311.0.0.27424c4dZ6aOky>, 29.9.2018.

Abbildungen:

Titelbild: siehe Abbildung 24

Abbildung 1: Stiftwalze der Haydn-Flötenuhr [2, S. 18]

Abbildung 2: Metallplatte [2, S. 30]

Abbildung 3: Experiment-Aufbau zum Testen der Linearmagnete, eigene Abbildung

Abbildung 4: Nahaufnahme Linearmagnet, eigene Abbildung

Abbildung 5: Das Plektrum biegt sich, eigene Abbildung

Abbildung 6: Das Plektrum drückt die Saite weg, eigene Abbildung

Abbildung 7: Das Plektrum dreht sich von der Saite weg, eigene Abbildung

Abbildung 8: Erster Prototyp des 3D-Plektrums, eigene Aufnahme

Abbildung 9: Konstruktion über dem Gitarrenkorpus, eigene Abbildung

Abbildung 10: Zupfkonstruktion, eigene Aufnahme

Abbildung 11: Linearmagnet für den Drückautomaten, eigene Abbildung

Abbildung 12: Lochplatte mit Linearmagneten von unten, eigene Abbildung

Abbildung 13: Mit Hölzern höhenangepasste Lochplatte, eigene Abbildung

Abbildung 14: Lochplatte für Linearmagnete, eigene Abbildung

Abbildung 15: Nummerierung der Bündel, eigene Abbildung, erstellt mit dem Programm GeoGebra

Abbildung 16: Arduino «Big Ben»-Melodie, eigene Abbildung, Screenshot aus Arduino

Abbildung 17: «Big Ben»-Melodie in Notenschrift und Tabulatur, eigene Darstellung, erstellt mit dem Programm «Power Tab Editor 1.7»

Abbildung 18: Arduino Kommunikation, eigene Abbildung, Bildschirmfoto aus Arduino

Abbildung 19: If-Schleife in Python zum Lösen und Drücken der Bündel, eigene Abbildung, Screenshot aus dem Programm Notepad ++

Abbildung 20: 16-Kanal Relais-Modul, eigene Abbildung

Abbildung 21: IC, eigene Abbildung

Abbildung 22: Schaltung der ICs [12]

Abbildung 23: Schaltplan, eigene Abbildung, erstellt mit dem Programm Fritzing

Abbildung 24: Das Endprodukt, eigene Abbildung

Abbildung 25: Endprodukt mit Schutz um Elektronik, eigene Abbildung

Anhang

Arduino-Code:

```
1. String inString = "";
2. void setup()
3. {
4.   Serial.begin(9600);           //Uebertragungsrate 9600
5.   for(int p=0; p<=37; p++) {
6.     pinMode(p, OUTPUT);        //initialisiert Pin 1-36 als Output
7.     digitalWrite (p, LOW);     //Stellt alle Pins auf AUS
8.   }
9.   Serial.println("Hallo Python, ich bin bereit!");
10. }
11.
12. void loop()
13. {
14.   // Liesst den seriellen Input
15.   while (Serial.available() > 0)
16.   {
17.     int inChar = Serial.read();
18.     if (isDigit(inChar))
19.     {
20.       // Konvertiert das empfangene Byte in ein char und fügt es dem String hinzu
21.       inString += (char)inChar;
22.     }
23.     // Bei einer neuen Zeile wird der String geschrieben, dann der Wert des Strings:
24.     if (inChar == '\n')
25.     {
26.       int pin = inString.toInt()+1; //erster Pin dient zur Kommunikation, deshalb +1
27.       if (pin > 100)
28.       {
29.         Serial.print("Pin druecken: ");
30.         Serial.println(pin);
31.         digitalWrite(pin - 100, HIGH);
32.       }
33.       else if (pin >=26 and pin < 100 )
34.       {
35.         Serial.print("Pin zupfen: ");
```

```

36.     Serial.println(pin);
37.     delay(5);
38.     digitalWrite(pin, HIGH);
39.     delay(50);
40.     digitalWrite(pin, LOW);
41. }
42. else {
43.     Serial.print("Pin loesen: ");
44.     Serial.println(pin);
45.     digitalWrite(pin, LOW);
46. }
47. //String für neuen Inhalt leeren
48. inString = "";
49. }
50. }
51. }

```

Python-Code:

```

1. #-*-coding:utf-8-*-
2. import serial
3. import sys
4. import time
5.
6. comport = int(sys.argv[1]) #comport = erstes Argument nach Aufruf in CMD
7. lied = str(sys.argv[2]) #Datei = zweites Argument nach Aufruf in CMD
8. tempo = float(sys.argv[3]) #Tempo = drittes Argument nach Aufruf in CMD
9.
10.
11. ser = serial.Serial('COM%r' %(comport), 9600)
12.
13. gitarrenbund = []
14. akg = [-1,-1,-1,-1,-1,-1,-1] #aktuell gedrückt
15. soz = [0,0,0,0,0,0,0] #Stossen oder ziehen
16.
17. def pinschicken(pin):
18.     ser.write('%r\n' %(pin)) #Erster Pin wird nicht besetzt, da er der Seriellen Übertragung dient -> in Arduino
19.     print ser.readline() #liest serielle Daten und gibt sie aus
20.
21. print ser.readline()

```

```

22.
23. lieddatei = open(lied, 'r')
24. for line in lieddatei:
25.     neues_element = (line.rstrip()) #Trennt einzelne Zeilen
26.     gitarrenbund.append(neues_element) #Fügt diese in die Liste gitarrenbund[]
27. lieddatei.close()
28.
29.
30. for akkord in range(0, len(gitarrenbund)): #von 0 - Anzahl Einträge im Lied
31.     print ""
32.     print ""
33.     print ""
34.     print "Akkord: %d" %(akkord+1) #Akkorde ab 1 Nummeriert
35.
36.     for saite in range(1,7):
37.         print ("Saite: %r" %(saite))
38.         gitbundtext1 = str(gitarrenbund[akkord-1]) #Umwandlung nach String
39.         gitbundtext2 = str(gitarrenbund[akkord])
40.
41.         bund1 = int(gitbundtext1[saite])-1 #zu drückender Bund, einzelne Saite
42.         bund2 = int(gitbundtext2[saite])-1
43.         ""
44.         EINSTELLIGE Zahl, Korrektur der Aufaddierung, Bund = 0 bedeutet leere Saite spielen,
45.         Bund = -1 bedeutet, Saite wird nicht gezupft, tiefe E Saite = Pin 1-4, A Saite Pin 5-
8 etc., ausrechnen von Pinnummer
46.         Ansteuerung Druckautomat:
47.         ""
48.         if bund1 != bund2 and bund2 > 0 and bund2 !=5 and akg[saite] >=0: #gelöst, gedrückt
49.             print ('Pinschickenloesen: %r' %((saite-1)*4 + akg[saite]))
50.             print "Bund wird geloest, neuer Bund wird gedrueckt" #Saite wird im 1. bis 4. Bund gedrückt
51.             print "Akg: %r" %(akg[saite])
52.             pinschicken((saite-1)*4 + akg[saite]) #bund1 lösen, auf akg zurückgreifen
53.
54.             print ('Pinschickendruecken: %r' %(((saite-1)*4 + bund2)+100))
55.             pinschicken(((saite-1)*4 + bund2)+100) #bund2 drücken
56.             akg[saite] = bund2 #akg neu setzen
57.
58.         elif bund1 != bund2 and bund2 > 0 and bund2 !=5: #gedrückt
59.             print "Bund wird neu gedrueckt"
60.             print ('Pinschickendruecken: %r' %(((saite-1)*4 + bund2)+100))

```

```

61.         pinschicken(((saite-1)*4 + bund2)+100) #bund2 drücken
62.         akg[saite] = bund2 #akg neu setzen
63.
64.         elif bund2 == 0 and akg[saite] > 0 or bund2 == 5: #gelöst
65.             print "Bund wird geloest, evtl eine Leersaite gezupft"
66.             print ('Pinschickenloesen: %r' %((saite-1)*4 + akg[saite]))
67.             pinschicken((saite-1)*4 + akg[saite]) #Bund loesen, es wird neu kein Bund gedrückt
68.             akg[saite] = 0 #akg wird auf 0 gesetzt
69.
70.         elif bund2 == -1 and akg[saite] >= 1: #gedrückt lassen
71.             print "Bund bleibt gedrueckt"
72.
73.         else: #weiterhin nichts drücken
74.             print "Bund wird weiterhin nicht gedrueckt"
75.
76.         if bund2 > -1 and bund2 != 5: #Ansteuerung Zupfautomat Saite wird leer oder mit gedrücktem Bund gezupft.
77.             if soz[saite] == 1:
78.                 if bund2 == 0:
79.                     print "Leersaite, Zupfen Hin, Pin: %r" %(2 * (saite-1) + 25)
80.                     pinschicken(2 * (saite-1) + 25)
81.                 else:
82.                     print "Bund: %r, Relais: %r, Zupfen Hin: Pin: %r" %(bund2,(((saite-1)*4 + bund2)),(2 * (saite-1) + 25))
83.                     pinschicken(2 * (saite-1) + 25)
84.                     soz[saite] = 0
85.                 else:
86.                     if bund2 == 0:
87.                         print "Leersaite, Zupfen Her, Pin: %r" %(2 * (saite-1) + 26)
88.                         pinschicken(2 * (saite-1) + 26)
89.                     else:
90.                         print "Bund: %r, Relais: %r, Zupfen Her: Pin: %r" %(bund2,(((saite-1)*4 + bund2)),(2 * (saite-1) + 26))
91.                         pinschicken(2 * (saite-1) + 26)
92.                     soz[saite] = 1
93.             time.sleep(tempo)
94.         print ("Lied Ende!")
95.         time.sleep(2)
96.         for aus in range(0,25): #Schaltet alle Drückemagneten aus
97.             pinschicken(aus)
98.         for aus in range(25, 37, 2): #Setzt alle Zupfmagneten in Ausgangsposition
99.             pinschicken(aus)
100.        print ("Magnete in Ausgangslage!")

```