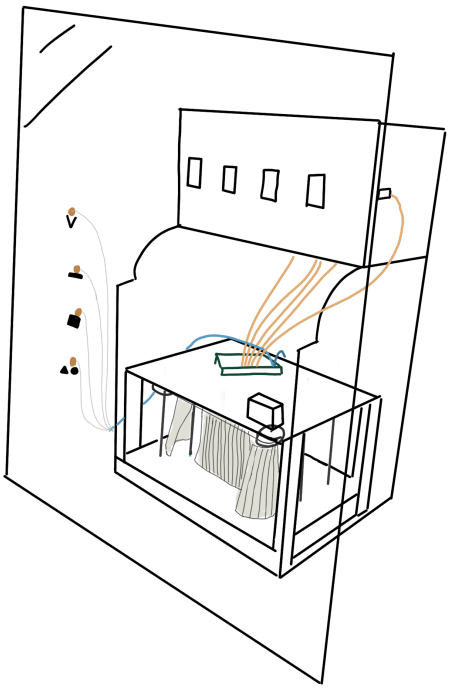
Kantonsschule Im Lee Winterthur

Maturitätsarbeit HS 2020/21

Bau und Programmierung einer interaktiven Schaufensterinstallation



Autorin: Maria De Menech 4e

Betreuerin: Margherita Fierz

Eingereicht: 04.01.2021 Winterthur

Inhaltsverzeichnis

Vorwort

Die Idee für die vorliegende Arbeit entstand bei einem Tischgespräch in der Familie. Meine Mutter besitzt ein Wollgeschäft in der Altstadt Winterthur. Die Frage stellte sich; Wie kann man ein Schaufenster attraktiver gestalten? In der Diskussion kam der Vorschlag, ein bewegtes Bild im Schaufenster zu kreieren, welches die Passanten beeinflussen könnten. Schon seit längerem trage ich mich mit dem Gedanken herum, mich in die Programmierung einzuarbeiten. Ein kleiner Einstieg in die Programmierung erhielten wir in der Schule durch die Anwendung von Tigerjython, eine Entwicklungsumgebung, wo wir eine „Turtle“ durch einfache Codes steuerten. Vielmehr als nur das Erstellen von einfachen Codes interessierte mich die Steuerung von mechanischen Komponenten durch Programmierung. Die Maturarbeit bot sich als ideale Einstiegsmöglichkeit, um sich tiefer in diese Thematik einzuarbeiten. So entschloss ich mich, in meiner Maturitätsarbeit eine interaktive Schaufensterinstallation zu bauen. Mein Ziel ist es, dass Passanten durch das Berühren der Schaufensterscheibe Objekte im Fenster bewegen können.

Abstract

Für die Umsetzung der interaktiven Schaufensterinstallation wurde entschieden einen Strickmusterautomaten zu entwerfen und zu bauen. Die Passanten können durch die Auswahl von Stricksymbolen ein beliebiges Strickmuster zusammenstellen, dass den Passanten angezeigt wird.
Das Vorgehen war wie folgt: Zuerst wurde schematisch der Aufbau des Automaten skizziert, dabei wurde definiert, wie der Passant Einfluss auf die Steuerung des Musterautomaten nehmen kann. Nach der Ideenentwicklung kam die Frage auf, mit welchen Materialien und mechanischen Komponenten die Installation gebaut werden soll. Es wurde entschieden die interaktive Bedienung des Automaten via Sensoren zu lösen und die Bewegung der Objekte durch Motoren abzubilden.
Im nächsten Punkt stellte sich die Frage, wie man die einzelnen Komponenten ansteuert und den benötigten Code für die Steuerung des Automaten programmiert.
Für die Steuerung wurde ein Raspberry Pi (ein Mikrocomputer) eingesetzt und als Programmiersprache wurde Python verwendet.
Nachdem die Ansteuerung der einzelnen mechanischen Komponenten gelöst werden konnte, folgte der effektive Bau des Musterautomaten. Um Problemstellungen und Fehlkonstruktionen zu erkennen und zu verbessern, wurden mehrere Prototypen gebaut. Während des Arbeitsprozesses sind drei Prototypen erstellt worden. Dabei stellte sich heraus, dass nur schon der Bau dieser Modelle sehr zeitaufwendig war und daher wurde entschieden, den letzten Prototyp als Ergebnis der Arbeit zu belassen. Die Mechanismen des Prototyps funktionierten wie geplant, auch die Interaktivität mit Touchsensoren durch eine Scheibe konnte umgesetzt werden. Der finale Schritt, den Prototyp in die effektive Schaufensterinstallation umzusetzen ist nicht mehr Bestandteil dieser Arbeit.

Einleitung

In der vorliegenden Arbeit wird der Weg und die Erkenntnisse zum Bau der interaktiven Schaufensterinstallation aufgezeigt.
Im ersten Kapitel wird der kreative Prozess beschrieben, nämlich die Ideenentwicklung der Installation. Im Zentrum stand die Überlegung, wie die Installation zum Schluss aussehen soll. Dies wird im Kapitel 3 gezeigt. Aufgrund dieser Idee wurde entschieden, mithilfe des Raspberry Pi und der Programmiersprache Python die Objekte anzusteuern. Im Kapitel 4 wird der Raspberry Pi, wie auch meine Entwicklungsumgebungen genauer beschrieben. Als nächster Punkt folgt der Umgang und die Steuerung der mechanischen Komponenten (Motoren und Sensoren) durch Python.
Zum Schluss kommt der effektive Bau meiner Installation. In diesem Abschnitt wird der Entwicklungsprozess beschrieben, wie man durch mehrere Prototypen Problemstellen ausfindig machen und lösen konnte. Auf Basis dieser Erkenntnis fand der Bau des finalen Musterautomaten statt.

Material und Methode

1 Material

Für diese Arbeit wurden folgende Materialien verwendet:

- Kapa-Platten
- Plexiglas
- Karton
- Kupferplatte
- Holzrollen
- Hozstäbchen
- Röhrchen
- Gummifaden
- Kabel
- Draht
- Breadboard
- RaspberryPi 4 1 GB
- Servo-Motoren
  - Continuous-Servo
  - Analog-Servo
- RPi Servo Driver HAT 16-Channel
- Adafruit CAP1188 Capacitive Sensor
- Sparkfun Gesturesensor APDS-9960 RGB
- MCP3008-Reader
- Widerstände
- Magnete
- Stepper Motor 28bYJ-48 5V DC
- Adafruit Motor HAT
- Hallsensor
- Spannungsregler

2 Entwicklung der Idee

2.1 Erste Überlegungen

Die erste Idee war es, ein Schattentheater in einem Schaufenster interaktiv zu gestalten. Die Passanten könnten von aussen die Figuren bewegen und so Einfluss auf die Geschichte des Schattentheaters nehmen. Doch der Bau eines solchen Schattentheaters wäre ohne jegliche Vorkenntnisse in Programmierung und mechanischer Steuerung viel zu kompliziert und zeitaufwendig gewesen.

2.2 Finale Idee

Als neue Idee für die Installation sollte etwas gebaut werden, was im Zusammenhang steht mit dem Geschäft, indem die Installation gezeigt werden sollte. Das ausgewählte Geschäft verkauft alles rund um Wolle. Gleichzeitig sollte die Installation einen technischen Aspekt aufzeigen. So fiel die Entscheidung einen Strickautomaten nach dem Modell einer alten Registrierkasse zu bauen. Mithilfe von Stricksymbolen sollte man mit diesem Automaten sein eigenes Strickmuster anzeigen lassen können.
Diese Stricksymbole sind wie eine eigene Sprache, die definiert, wie gestrickt werden soll. Entsprechend wie beim Programmieren, wo man eine Programmiersprache lernen muss. Diese Analogie war passend zu dem Projekt und so entstand die Idee dieser Kasse, welche ausgewählte Maschenproben anzeigen kann. In Abbildung 1 ist ein Beispiel einer sogenannten Strickschrift aufgezeigt. Jedes Strickprojekt startet mit einer Maschenprobe. Das bedeutet, man strickt ein kleines Quadrat, bei welchem man das gewünschte Muster ausprobiert und dadurch bestimmen kann, wie viele Maschen und Reihen man für eine gewisse Breite und Länge stricken muss. (siehe Abbildung 2)
Die mechanischen Komponenten der Kasse sollten mit Servo-Motoren angesteuert werden. Da diese nur relativ einfache Bewegungen ausführen können und nicht in der Lage sind, zu schwere Objekte zu bewegen, mussten alle Objekte aus Karton und Papier gebaut werden.
Die Interaktivität sollte so umgesetzt werden, dass die Passanten die gewünschten Stricksymbole auswählen können und der Automat danach das passende Muster anzeigt.

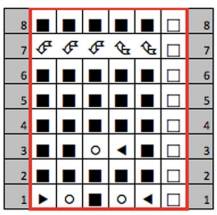


Abbildung 1: Beispiel einer Strickschrift[1]

Abbildung 2: Beispiel einer gestrickten Maschenprobe[2]

3 Schlussergebniss

Da die Erstellung des geplanten Prototyps den zeitlichen Rahmen einer Maturitätsarbeit bereits vollumfänglich ausgefüllt hatte, wurde entschieden, die Arbeit bei einem funktionstüchtigen Prototyp zu belassen. Dieser Prototyp wird in Abbildung 3 und 4 gezeigt. Der Weg zum Prototyp, wie auch Ergebnisse und Schwierigkeiten zum Bau wird in den folgenden Abschnitten beschreiben.

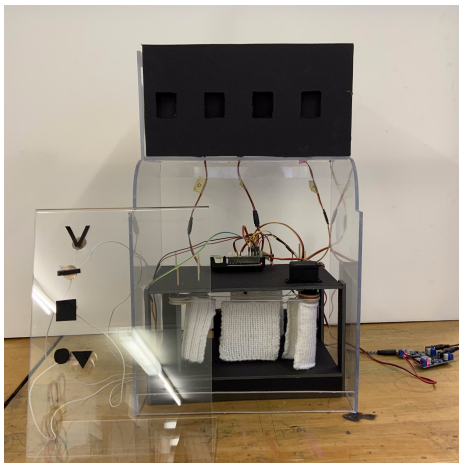


Abbildung 3: Fertiger Prototyp in der Ansicht von Vorne

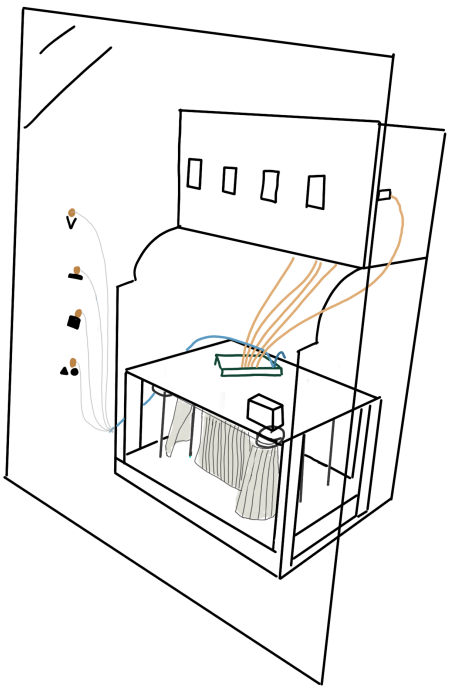


Abbildung 4: Skizze des Prototypen mit allen eingebauten Komponenten

4 RaspberryPi

Mithilfe des Mikrocomputers „Raspberry Pi“ werden die Motoren und Sensoren angesteuert. Der RaspberryPi ist ein Einplatinen-Computer. Dies bedeutet, dass alle notwendige Komponente auf einer Leiterplatte untergebracht sind (siehe Abbildung 5). Dadurch ist der Raspberry Pi ein sehr kompakter Mini-Computer, welcher trotzdem eine beachtenswerte Rechenleistung hat. Dies und die anwenderfreundliche Benutzeroberfläche führte zur Entscheidung, den Raspberry Pi für das Projekt zu benutzen.
Auch ohne grosse Vorkenntnisse im Programmieren lassen sich schnell einfache Projekte verwirklichen. Im Internet gibt es viele Foren und Beispielprojekte, in denen man Hilfestellungen zu Problemen und zur Benutzung findet.
In der Abbildung 6 wird schematisch der Raspberry Pi aufgezeigt mit seiner Anschlussbelegung (Pinouts). Diese Anschlüsse (Pins) ermöglichen dem Raspberry Pi die Kommunikation nach aussen.

Abbildung 5: Raspberry Pi

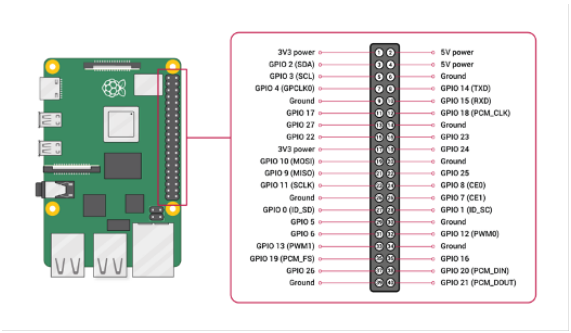


Abbildung 6: Schematische Darstellung des RaspberryPi's mit Pinout[3]

4.1 Zugriff auf den Raspberry Pi

Der RaspberryPi wird mit einem vorinstallierten Betriebssystem[1] ausgeliefert. Dieses Betriebssystem bietet sogar eine graphische Oberfläche. Um diese zu benutzen, kann direkt am Raspberry Pi ein Monitor und eine Tastatur angeschlossen werden (siehe Abbildung 7). Der Monitor wird über ein HDMI-Kabel und die Maus und Tastatur über ein USB-Kabel verbunden.
Jedes Mal den RaspberryPi so aufwendig zu starten, wurde mit der Zeit recht umständlich, deshalb wurde als Verbindung zum Bildschirm eine andere Lösung gesucht, nämlich der Einsatz des VNC-Viewers. VNC bedeutet Virtual Network Computing. Dies ist eine Software, um vom lokalen PC aus auf den Raspberry Pi zuzugreifen, als wären Tastatur und Bildschirm am Raspberry Pi angeschlossen.
Durch das bereits eingebaute WLAN-Modul im Raspberry Pi, muss sich der PC nur im selben lokalen Netzwerk befinden. Danach reicht es den Raspberry Pi an die

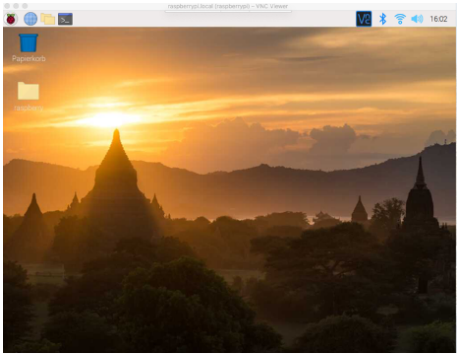


Abbildung 7: Vorinstallierte graphische Oberfläche des Raspberry Pi

[1] RaspberryPi OS, dies ist eine Linux-Distribution[4]

Stromversorgung anzuschliessen und via VNC-Viewer auf den RaspberryPi zuzugreifen.

4.2 Python

Mit Python wurden die Sensoren und Servo-Motoren programmiert. Python ist die primäre Programmiersprache des Raspberry Pi's und ist bereits vorinstalliert. Es stellte sich heraus, dass es viele Foren und Beispielprojekte als Hilfestellungen im Internet vorhanden sind.

4.2.1 Entwicklungsumgebung Thonny

Zu Beginn wurde für die Programmierung Thonny verwendet. Thonny ist ein integrated development environment (IDE). Also eine Entwicklungsumgebung für Python, welche auf dem RaspberryPi schon integriert ist (siehe Abbildung 8). Diese ist sehr einfach aufgebaut und nur auf das Notwendigste reduziert, was für einfache Codes auch genügt. Um später komplexere Codes zu schreiben, erwies sich Thonny als unpraktisch. Da diese Entwicklungsumgebung recht reduziert ist, wurden die Codes schnell unübersichtlich und das «Debuggen» stellte sich ebenfalls schwierig heraus. Auch das Programmieren über VNC erwies sich mit der Zeit als umständlich.

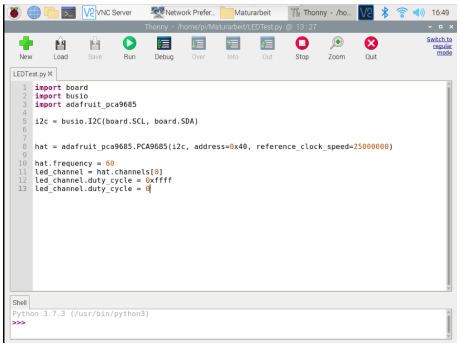


Abbildung 8: Grafische Oberfläche von Thonny IDE

4.2.2 Entwicklungsumgebung Visual Studio Code

Die im vorherigen Kapitel beschriebene Probleme lassen sich durch den Einsatz von Visual Studio Code[5] umgehen. Visual Studio Code ist eine äusserst umfangreiche Entwicklungsumgebung, inder man mit unterschiedlichsten Programmiersprachen entwickeln kann. Die Erweiterung "Remote-SSHërmöglicht einen einfachen Zugang auf den Raspberry Pi. Vor allem diese Funktion führte zum Entscheid Visual Studio Code zu verwenden. Die Secure Shell oder auch Secure Socket Shell (SSH) ist ein Protokoll, mit welchem man verschlüsselt auf einen entfernten Computer zugreifen kann (siehe Abbildung 9). Dadurch ist der Zugriff auf das Dateisystem des Raspberry Pi's möglich, um Dokumente zu ändern oder auch Neue zu erstellen. Auf diese Weise wird der Programmcode direkt auf dem Raspberry Pi erstellt oder editiert, ohne dass eine VNC Verbindung notwendig ist. Folglich lässt sich die vollumfängliche Funktionalität des Visual Studio Code auf dem lokalen PC nutzen.

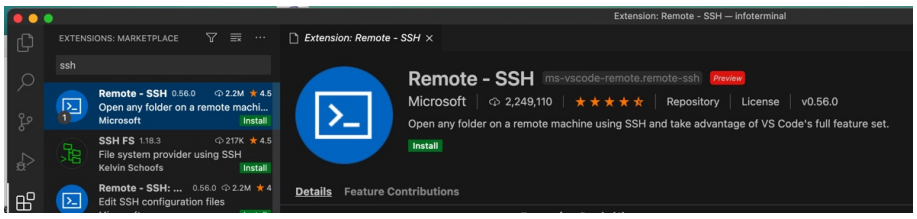


Abbildung 9: Remote SSH Erweiterung für Visual Studio Code

5 Motoren und Sensoren

5.1 Servo-Motoren

Um überhaupt irgendein Objekt zu bewegen benötigt man Motoren. In diesem Projekt wurden Servo-Motoren verwendet (siehe Abbildung 10). Dabei unterscheidet man zwischen Analog- und Continuous-Servos. Die Analog-Servos lassen sich um einen gegebenen Winkel von 0° bis 180° drehen.
Die Continuous-Servos drehen, wie der Name schon sagt, kontinuierlich. Es lassen sich nur die Geschwindigkeit, Drehrichtung und die Zeit, wie lange der Servo drehen soll angeben.

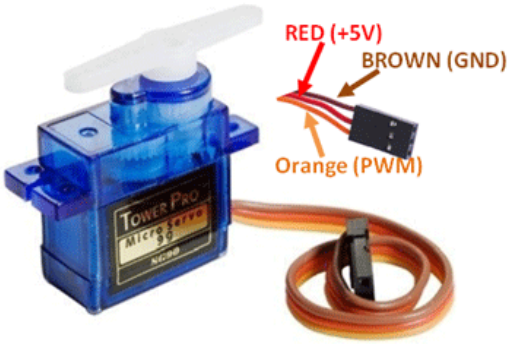


Abbildung 10: Analog-Servo, mit beschrifteten Anschlüsse[6]

Für die einfache Verbindung der Servos an den Raspberry Pi, verwendet man ein RPi Servo Driver HAT 16-Channel (siehe Abbildung 11). Der Vorteil dabei ist, dass man mehrere Servos gleichzeitig anschliessen kann. Bei dem verwendeten Hat können 16 Servos angeschlossen werden. Um die Servos zu Programmieren wurde die ServoKit Library von Adafruit verwendet[7].
Zu Beginn jedes Codes muss man also diese Python Library importieren, wie auch angeben wie viele Anschlüsse der Servo-Hat besitzt. Das Hat, welches verwendet wurde, hat 16 Anschlüsse. Somit können 16 Servos gleichzeitig angesteuert werden.

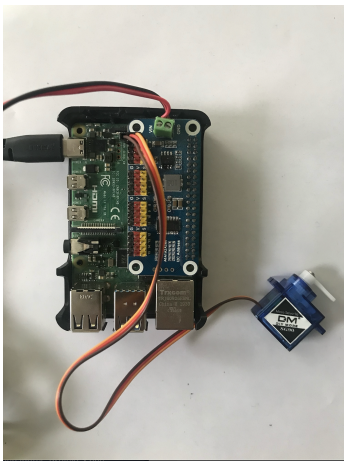


Abbildung 11: Servo Hat mit angeschlossenem Analog Servo

5.2 Steppermotoren

Ein Steppermotor ermöglicht eine Drehbewegung durch Ausführung einzelner Schritte. Durch diese spezielle Ansteuerung ist eine genaue Positionierung möglich. Im Steppermotor selber hat es vier Spulen um den magnetischen Drehkörper und diese können in Einzelschritten angesteuert werden. Um nun einen Schritt

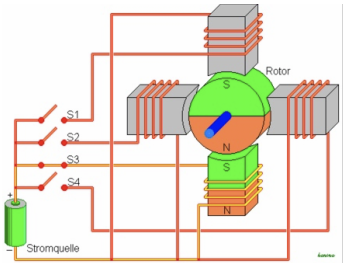


Abbildung 12: Schematische Darstellung der Funktionsweise eines Steppermotors[8]

zu drehen, werden die Spulen abwechslungswei-
se unter Spannung gesetzt. Dadurch wird der Magnet abgestossen oder angezogen. So dreht der Motor Schritt für Schritt. (siehe Abbildung 12)
Mit den bis jetzt verwendeten Servo-Motoren war die Drehbewegung eingeschränkt. Nur bis 180° konnte man einen Winkel definieren. Wünscht man eine exakte Positionierung über 180° benötigt man einen Steppermotor.
Der verwendete Stepper Motor hat die Bezeichnung 28bYJ-48 (siehe Abbildung 13). Um diesen am Raspberry Pi anzuschliessen wird ein "DC und Stepper Motor Hat"vom Hersteller Adafruit verwendet (siehe Abbildung 14).

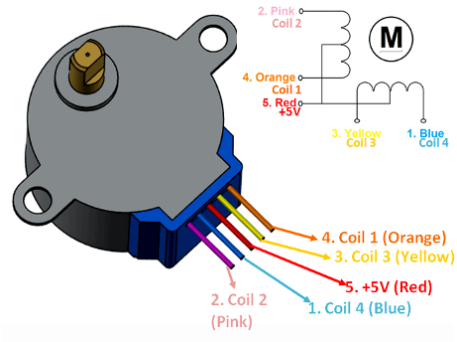


Abbildung 13: Verwendeter Steppermotor mit beschrifteten Anschlüssen[9]

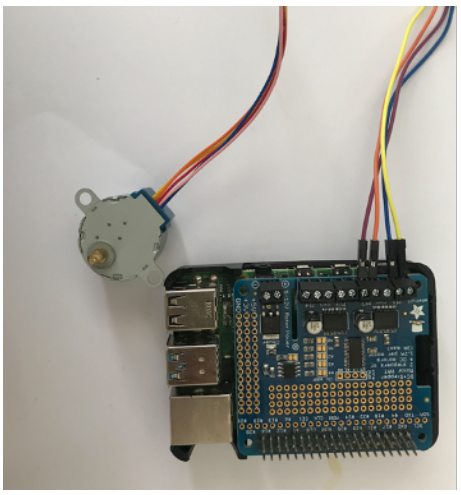


Abbildung 14: Motor-HAT auf RaspberryPi

Dies allein reicht noch nicht, um die Ansteuerung des Motors zu ermöglichen. Dafür benötigt man zusätzlich, die Motorkit Library von Adafruit [10]. Erst dadurch erkennt der Raspberry Pi den angeschlossenen Motor. Den Steppermotor steuert man in Schritten, dabei lässt sich festlegen, in welche Richtung er drehen soll. Da diese Art von Motoren ständig unter Spannung stehen, um ihre genau Position zu fixieren, kann man den Motor auch frei drehen lassen. Somit verhindert man das Überhitzen der Motoren.

5.3 Touchsensoren vs. Bewegungssensoren

Geplant ist es die Installation interaktiv für den Passanten zu gestalten. Der Passant soll durch die Scheibe die Objekte im Schaufenster ansteuern können. Dies sollte über Touch- oder Bewegungssensoren geschehen, welche in diesem Kapitel genauer beschrieben werden.
Der Touchsensor, welcher verwendet wurde, ist der Adafruit CAP1188 Capacitive Sensor (siehe Abbildung 15). Die berührungsempfindlichen Pins können über Kabel und Kupferplättchen verlängert werden. Für die Programmierung wird die cap1188 Library von Adafruit verwendet [11].

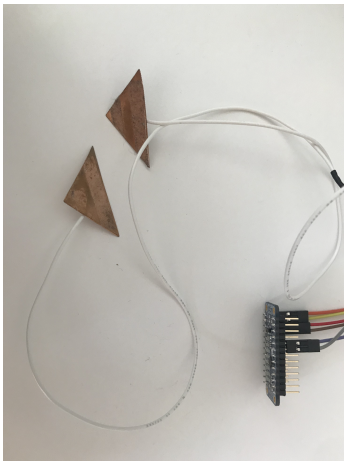


Abbildung 15: Touchsensor verbunden mit Kupferplättchen

Es stellte sich heraus, dass der Touchsensor sehr empfindlich ist und auf jegliche Berührung sofort reagiert. Durch diese hohe Empfindlichkeit ist es wichtig, dass man die Kabel, welche den Sensor mit den Kupferplättchen verbindet, ruhig hält. Sonst erkennt der Sensor die Bewegung der Kabel als Berührung.
Als Bewegungssensor wurde der Sparkfun Gesturesensor APDS-9960 RGB verwendet. Dieser Sensor erkennt Bewegungen und kann unterscheiden zwischen Bewegungen von recht, links, oben und unten.
Auch hier wird für die Ansteuerung eine Library verwendet, die apds9960 Library von Adafruit[12].
Der Bewegungsensor zeigte bei Tests, dass er nicht immer genaue Messresultate zurückgab oder überhaupt keine Bewegung erkannte. Er ist nicht so empfindlich wie der Touchsensor.

5.3.1 Sensoren durch Glasscheibe

Um festzustellen, ob die Sensoren überhaupt durch eine Scheibe die Berührung respektive Bewegung erkennen können, wurden zum Test eine Plexiglasscheibe verwendet (siehe Abbildung 16, 17).

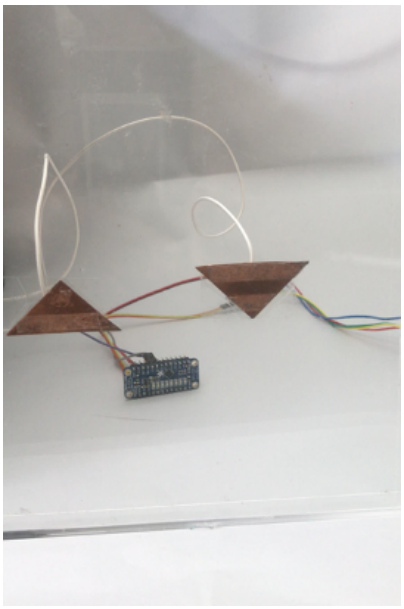


Abbildung 16: Versuchsaufbau der Touchsensoren durch Plexiglas

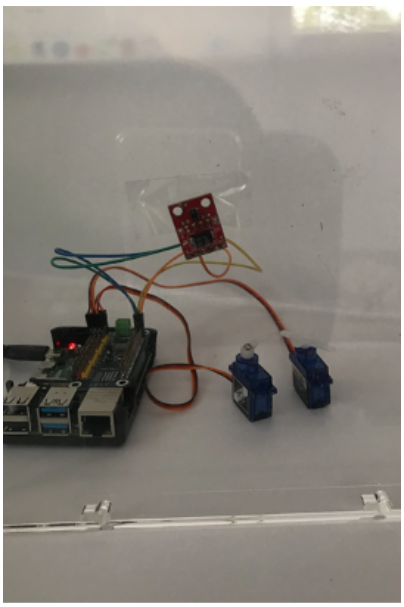


Abbildung 17: Bewegungssensor durch Plexiglas

Der Touchsensor funktionierte einwandfrei. Durch das Befestigen der Kupferplättchen und der Plexiglasscheibe, konnte verhindert werden, dass sich die Kabel gegenseitig berühren und falsche Ergebnisse lieferten.
Beim Bewegungssensor hingegen, konnte die Bewegung durch die Scheibe nicht erkannt werden. Auch mit Ändern der Lichtverhältnisse konnte kein Ergebnis erzielt werden.
Folglich um die Interaktivität der Installation umzusetzen, fiel der Entscheid auf den Touchsensor.

6 Bau des Modells

6.1 Entwicklung des Modells

Der endgültige Prototyp entstand über einen iterativen Entwicklungsprozess. Insgesamt wurden drei Modelle gebaut (siehe Abbildung 18). Jedes dieser Teilmodelle half dabei Probleme ausfindig zu machen und Fehler zu korrigieren. Die Mechanismen für die Steuerung der Komponenten der Installation wurden stetig weiterentwickelt. Das Endresultat ist im finalen Prototyp ersichtlich.

Abbildung 18: Alle drei Prototypen, vom Ältesten (links) zum Neusten (rechts)

6.1.1 Bau der Hülle

In der Abbildung 18 wird ersichtlich, dass zu Beginn die Prototypen aus Karton gebaut wurden. Es stellte sich aber heraus, dass der Karton zu dünn war und die geplanten Mechanismen beeinträchtigt wurden. Deshalb wurde für die Weiterentwicklung mit Kapa-Platten (Hartschaumplatten) gebaut, welche stabiler und dicker sind. Die Hülle des Automaten hat eine Höhe von 50 cm und Breite von 30 cm. Mit einem genügen grossen Cutter oder auch spezifischen Hartschaum-Cutter lassen sich einfach gerade Stücke, wie auch Kurven aus Kapa-Platten herausschneiden. Je grösser die Stücke sind, desto einfacher sind sie zu schneiden. Mithilfe von Kontaktkleber wurden die einzelnen Teile zusammengefügt.

6.2 Symbolanzeige

Die Inspiration der Symbolanzeigen lieferte eine alte Registrierkasse. Anstatt Zahlen sollten die vier verschiedenen Stricksymbole verwendet werden (siehe Tabelle 1). Wie in Abbildung 19 war die Anfangsidee, die Symbole auf Kärtchen abzubilden, welche sich bei Auswahl senkrecht nach oben bewegen. Diese Bewegung sollte über Analog-Servos umgesetzt werden. Jedoch war durch die eingeschränkte Bewegung der Servo-Motoren diese Idee mechanisch umzusetzen viel zu aufwendig.
Bei anderen Registrierkassen wurden die Zahlen auf Rollen angebracht (siehe Abbildung 20), welche je nach Auswahl sich bis zur richtigen Stelle drehten. Diese Idee

für die Umsetzung der Symbolauswahl erschien einfacher. Folglich wurde entschieden eine vierteilige Rolle zu bauen (siehe Abbildung 22), um alle Stricksymbole je nach Eingabe abzubilden.
Da ein Zylinder aus Kapa-Platten zu bauen relativ schwierig ist, wurden fünfeckige Prismen gebaut (siehe Abbildung 21). Dabei werden auf vier Seiten die jeweiligen Symbole angezeigt und eine Seite bleibt leer. Wenn kein Symbol ausgewählt wird, sollte auch kein Symbol angezeigt werden. Die Ansteuerung der Prismen erfolgte über Steppermotoren. Dabei stellte sich heraus, dass auf dem Motor Hat nur zwei Steppermotore angeschlossen werden können und diese auch zu langsam drehen.

Abbildung 19: Mechanik Symbolanzeige[13]

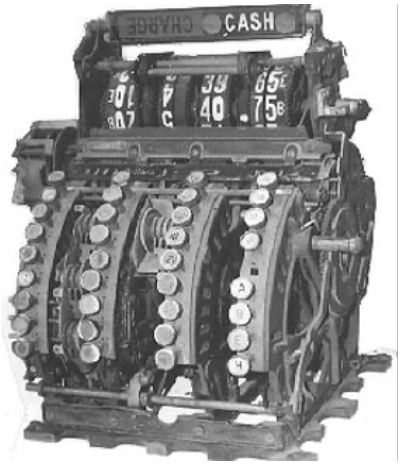


Abbildung 20: Mechanik Registrierkasse mit "Rolle"[14]

Diese Erkenntnis führte dazu bei der Ansteuerung der Symbolauswahl Continous-Servos zu verwenden. Durch bestimmte Zeit- und Geschwindigkeitsangaben wird die Drehbewegung definiert, um das Prisma zu dem gewünschten Symbol zu drehen.
Jedem Symbol wurde eine Nummer zugewiesen. So konnte man später im Code definieren welche Eingabe zu welchem Symbol führt.

Tabelle 1: Beschriftung der Stricksymbole

Nummer	Bedeutung	Symbol
1	Rechte Masche vorne, hinten linke Masche	V
2	Linke Masche vorne, rechte Masche hinten	–
3	Rechte Masche vorne, hinten rechte Masche	□
4	zwei Maschen recht zusammen, Umschlag	△○

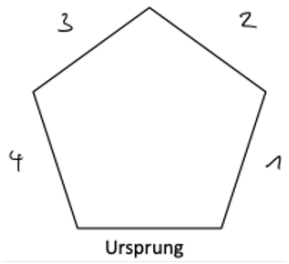


Abbildung 21: Skizze des Prismas mit Beschriftung der Seiten

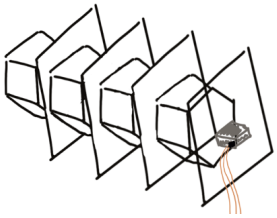


Abbildung 22: Schematische Darstellung der Symbolanzeige

In Abbildung 23 wird die fertige Symbolanzeige von vorne dargestellt. Die einzelnen Prismen wurden in vier abgetrennten Abschnitten platziert (siehe Abbildung 24). In jeder Trennwand ist für die Steuerung ein Continuous-Servo befestigt.

Abbildung 23: Fertige Symbolanzeige von vorne

Abbildung 24: Fertige Symbolanzeige von hinten

6.3 Mechanik der Musteranzeige

Nach der Auswahl der Symbole sollte bei jeder Symbolfolge von vier Zeichen das passende Strickmuster angezeigt werden. Die Anzeige erfolgt im unteren Bereich des Automaten, folglich müssen sämtliche Muster im Automaten einen Platz finden.

In den ersten Modellen wurde überlegt, die Muster auf einem Zylinder aufzuspannen und durch einen Motor bis zu dem gewünschten Muster drehen zu lassen. Durch ein Fenster im Automaten sollte das Muster dann angezeigt werden.

Diese Idee wurde aber schnell wieder verworfen, da sobald man mehrere Muster anzeigen möchte, der Zylinder zu gross würde. Die neue Idee war dann die Muster auf eine Schnur zu spannen und über Rollen, wie über ein Laufband zu führen (siehe Abbildung 25, 26). Auf Holzstäben wurden Holzrollen oben und unten befestigt. Diese Rollen besitzen eine Rille, über die ein Faden geführt werden kann. Der Vorteil bei dieser Konstruktion ist, dass sie sie platzsparend ist. Durch das Bilden von Schlaufen im Laufband kann die Schnur verlängert werden. An diesem Laufband werden die einzelnen Maschenproben aufgespannt. Die Ansteuerung der Konstruktion erfolgt über einen Steppermotor. Erfolgt eine Musterauswahl fährt der Motor das richtige Muster an das Anzeigefenster.

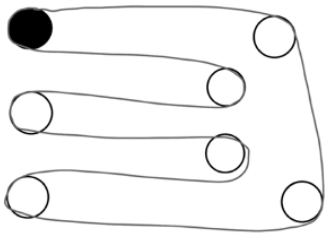


Abbildung 25: "Laufband"von oben, schwarzer Kreis zeigt Rolle welche mit dem Motor verbunden ist

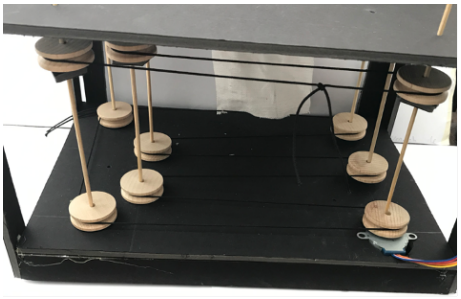


Abbildung 26: Aufbau dieser Musteranzeige mit Steppermotor

Es stellte sich heraus, dass der Stepper Motor nicht geeignet ist für diese Konstruktion. Er drehte viel zu langsam und war zu schwach. Auch zeigte sich, dass das gleichzeitige Ansteuern von zwei parallelen Holzrollen sehr schwierig war. Die unteren und oberen Rollen drehten nicht gleich schnell. Das lag daran, dass nur die untere Rollen direkt mit dem Motor verbunden war.

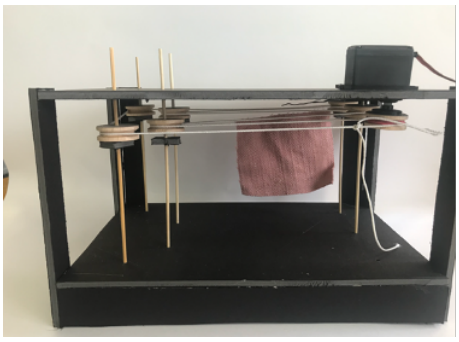


Abbildung 27: Konstruktion der Musteranzeige mit Continuous-Servo

Diese Erkenntniss führte dazu, die Konstruktion neu durch einem stärkeren Continuous-Servo anzusteuern. Ähnlich wie bei den Symbolanzeigen, kann man durch Geschwindigkeit- und Zeitangaben die Muster platzieren. Auch wurde die Schnur durch eine elastische Schnur ersetzt. So kann man sichergehen, dass sie nicht mehr auf den Rollen rutschte. Zusätzlich wurde ein Gummi um die Rolle befestigt, die mit dem Motor verbunden ist. So erhöht sich die Reibung zwischen Rolle und Schnur. Um das Problem der gleichzeitigen Ansteuerung der parallelen Rollen zu vermeiden, wurden die unteren Rollen weggelassen. Somit hängen die Muster nur noch oben an einer Schnur (siehe Abbildung 27). Am Anfang wurden für die Tests leichte Stoffquadrate verwendet. Sobald die gestrickten Muster an die Schnur gehängt wurden, fiel die Schnur häufiger aus den Rollen. Ursache des Problems war, dass die gestrickten Muster schwerer waren und dadurch die Schnur leicht nach unten gezogen wurde. Dies führte dazu, dass die Rollen, die nicht in ihrer Höhe fixiert waren, auch runterrutschten. Um dieses Problem zu beheben, wurden gleichlange Röhrchen unter den Rollen platziert und somit blieben die Rollen immer auf derselben Höhe (siehe Abbildung 28 und 29).

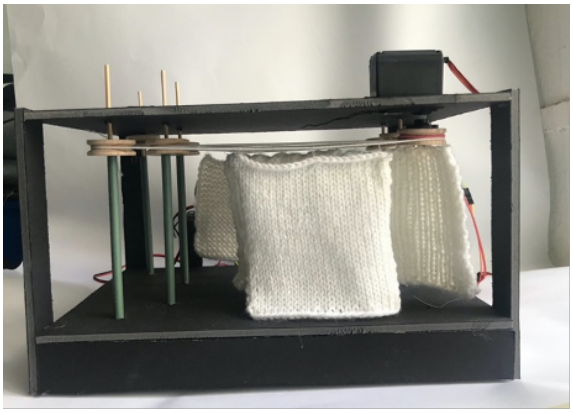


Abbildung 28: Musteranzeige von vorne

Abbildung 29: Musteranzeige Ansicht von links

6.4 Platzierung der Muster

Im vorherigen Kapitel wurde das Problem der Ansteuerung und Bewegung der Muster gelöst. Nun müssen aber die Muster richtig platziert werden. Die Problemstellung, die nun gelöst werden muss, ist wie man das ausgewählte Muster zum Anzeigefenster bewegt. Optimal wäre es, die Position sämtlicher Muster zu kennen. Um dies zu vereinfachen markiert man die Position eines Musters und platziert alle anderen Muster realtiv zum Martkierten. Folglich wird nun ein Mechanismus benötigt, der die Position eines bestimmten Musters sich merken kann.

6.4.1 Platzierung durch Induktion

Die erste Überlegung war, eine Drahtspule in das Modell einzubauen und ein Magnet an einem Muster zu befestigen. Wenn nun das Muster an der Drahtspule vorbeifährt, sollte aufgrund der Induktion[2] eine Spannung gemessen werden können. Somit würde man durch diesen Wert wissen, wo sich das markierte Muster befindet. Damit die gemessene Spannung möglichst gross ist und somit klarere Werte geliefert werden können, wurde eine Spule verwendet anstatt ein einfacher Drahtring (siehe Abbildung 30).
Die Spannung misst man analog. Da der Raspberry Pi kein Analog-Reader hat, musste ein Analog-Digital-Reader angeschlossen wer-

Abbildung 30: Versuchsaufbau zum Messen des Induktionseffekt

[2]Sobald es eine Änderung des magnetischen Flusses gibt, wird eine Spannung induziert. Zur Änderung des magnetischen Flusses kann es auch kommen, wenn sich die magnetische Flussdichte ändert, was hier der Fall ist. Die induzierte Spannung ist folgendermassen definiert:

$$U = -N * \frac{\Delta\Phi}{\Delta t}$$

dabei ist N die Anzahl Windungen der Spule, Φ der magnetische Fluss und Δt die Änderung der Zeit

den. Verwendet wurde der MCP3008-Reader. Die Spule aus Draht wurde mit dem Ground Pinout und über einen Widerstand mit dem Analogwandler verbunden. Um Spannungsunterschiede zu messen, muss man einen gemeinsamen Nullpunkt zwischen dem Analogwandler und dem Raspberry Pi festlegen. Dieser Nullpunkt ist als Ground Pinout definiert. Nun sollte sobald ein Magnet sich an der Spule nähert ein Wert gemessen werden. Mit der Formel ergibt sich die Spannung U aus dem Produkt des Widerstands R und der Stromstärke I, wobei die Stromstärke I konstant ist.

$$U = R * I$$

Daraus folgt, um nun möglichst einen grossen Spannungsunterschied zu messen, muss man einen möglichst grossen Widerstand verwenden. Hier wurde ein Widerstand von 460Ω verwendet.
Es zeigte sich, dass sich dieser Aufbau mit der Spule als ungenau und unpräzise herausstellte. Anstatt noch mehr Zeit in diese Lösung zu investieren, wurde nach einem neuen Ansatz gesucht.

6.4.2 Platzierung der Muster durch Hall-Effekt

Abbildung 31: Grossaufnahme des verwendeten Hall-Sensors

Über einen Hallsensor (siehe Abbildung 31) lässt sich ein Magnetfeld durch den sogenannten Hall-Effekt erkennen. Wie bei der Platzierung durch Induktion, wurde ein Magnet an einem Muster befestigt. Durch den Sensor sollte das Magnetfeld, welches durch das Magnet am Muster erzeugt wird, gemessen werden. So wird gleichzeitig die Position des markierten Musters gemerkt. Beim verwendeten Sensor lässt sich über ein Potentiometer die Empfindlichkeit einstellen. Der Sensor misst ständig Werte. Wird keine Spannung gemessen, wird das als Wert 0 interpretiert. Sobald ein Magnetfeld detektiert wird, heisst dass, es hat eine Spannung und der Wert wird als 1 interpretiert.

Funktionsweise Hallsensor

Der Hallsensor misst im Prinzip keine Magnetfelder, sondern eine Spannung, die durch den sogenannten Hall-Effekt entsteht. Der Sensor wird von Strom durchflossen. Durch das Magnetfeld wirkt eine Kraft auf die Elektronen und verschiebt diese. Dadurch entsteht eine Spannungsverteilung, die ein elektrisches Feld hervorruft, das als Spannung gemessen werden kann. Es sollte also durch diesen Sensor möglich sein, ein Magnetfeld präziser zu erkennen, auch wenn sich das markierte Muster sehr schnell am Sensor vorbeibewegt.

```
#importiere GPIO Modul
import RPi.GPIO as GPIO
#importiere time Modul
import time
#setzte Pins des Raspberry Pi fuer den HallSensor fest
GPIO.setmode(GPIO.BCM)
GPIO.setup(4, GPIO.IN, pull_up_down=GPIO.PUD_DOWN)
#GPIO sollte auf Event reagieren, hier die Aenderung des Pin 4
```

```
GPIO.add_event_detect(4,GPIO.FALLING)
#definiere callback Funktion
def my_callback(n):
#sobald ein Magnetfeld gemessen wird, sollte "pushed" ausgegeben
    werden
    print("pushed")
#Callback Funktion registrieren
GPIO.add_event_callback(4, my_callback)

while True:
    print (GPIO.input(4))
    time.sleep(2)
```

Um den Hall-Sensor zu programmieren, wurde eine Funktion namens `callback` verwendet. Mit dieser Funktion wird der Code erst ausgeführt, sobald der Hallsensor ein Magnetfeld gemessen hat. Falls der Hallsensor kein Magnetfeld misst, wird auch kein Code ausgeführt. Da nun durch den Hall-Sensor die Position eines Musters bestimmt wurde, kann man zählen, wie lange und wie schnell der Continuous-Servo drehen muss, damit die anderen Muster platziert werden können. Da es sich bei diesem Modell des Musterautomaten nur um einen Prototyp handelt, wurden nur fünf verschiedene Muster verwendet, um die grundsätzliche Idee aufzuzeigen. (siehe Abbildung 32,33).

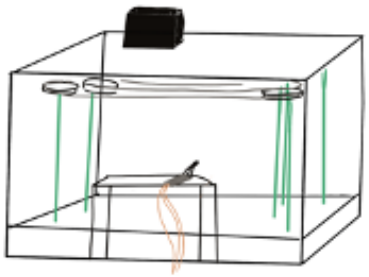


Abbildung 32: Skizze der Musteranzeige mit Hall-Sensor

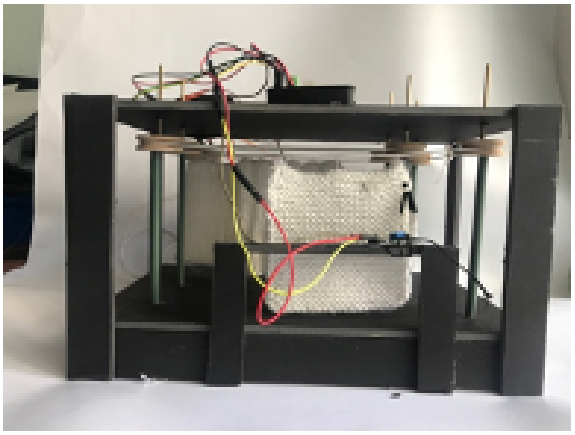


Abbildung 33: fertige Musteranzeige, mit gestricktem Muster und angeschlossenem Hall-Sensor

6.5 Programmierung der einzelnen Komponente

6.5.1 Symbolanzeige

Die Symbolanzeigen sind abhängig von dem Input der Touchsensoren. Im Code wurden zwei Funktionen definiert. Die Funktion `Symbolanzeige`, um das ausgewählte Symbol anzugeigen und die Funktion `zumUrsprung`, um die Anzeigen zum Ursprung zurückzudrehen. Berührt man das erste Symbol, dreht das erste Fenster der Symbolanzeige durch einen Continous-Servo zum ausgewählten Symbol. Automatisch, nachdem der erste Pin berührt wurde, wird nun bei der weiteren Auswahl, das nächste Fenster der Symbolanzeige angesteuert, usw. Ein Problem, das auftauchte

war, dass alle Servo-Motoren unterschiedlich schnell drehten, auch wenn die gleiche Geschwindigkeit für alle eingestellt wurde. Um dies zu umgehen, wurde für jeden Servo-Motor ein eigener Code erstellt, um so sicherzustellen, dass am Schluss das richtige Symbol angezeigt wird. Während der Platzierung der Symbole, wird die Position in einer Variablen festgehalten. Diese Information wurde später genutzt, um die Symbolanzeigen wieder zum Ursprung zurückzudrehen.

```
#globale Variabel choices wird auf 0 gesetzt, in dieser Variabel wird das ausgewaehlte Muster gespeichert
choices = 0
#Funktion Symbolanzeige wird definiert, der Paramenter a gibt an welcher Servo angesteuert wird
def Symbolanzeige(a):
    i = 0
    global choices
#solange i kleiner als 1 ist wird der folgende Code durchgefuehrt
    while i < 1:
#Falls der erste Pin des Touchsensors gedrueckt wird, wird folgender Code ausgefuehrt
        if cap[1].value:

#continous-Servo dreht aufgrund der Geschwindigkeits-und Zeitangabe
            kit.continuous_servo[a].throttle= 0.1
            time.sleep(0.25)
            kit.continuous_servo[a].throttle=0
#zu i wird eins dazugezaehlt, damit der naechste Servo angesteuert wird bei der naechsten Auswahl
            i = i + 1
#bei der Auswahl des ersten Pins ist choices = 1
            choices = 1
#Hier wird der Wert von choices zur ckgegeben
            return choices

                .
                .
                .
#der Code ginge hier noch weiter, ist aber im prinzip dasselbe einfach fuer jeden Touchsensor bewegt sich der Servo an einer anderen Stelle.
# der Parameter a beschreibt wieder, welcher Servo angesteuert wird

def zumUrsprung(a):
  #der Wert von choices wird verglichen, falls der Vergleich True ist wird der nachfolgende Code ausgefuehrt
        if choices == 1:
            kit.continuous_servo[a].throttle= - 0.2
            time.sleep(0.25)
            kit.continuous_servo[a].throttle=0
                .
                .
                .
#der Code ginge hir noch weiter, um von anderen Positionen je nach Auswahl sich zum Ursprung zu bewegen
```

6.6 Auswahl einschränken

Wie bereits erwähnt, wurden nur fünf Muster verwendet. Mit vier unterschiedlichen Symbolen könnte man aber insgesamt 256 Symbolfolgen kombinieren. Man muss also die Auswahl für die Symbolfolgen so einschränken, dass nur fünf Kombinationen richtig sind. Dieses Problem wurde mit Arrays behoben. Alle richtigen Symbolfolgen wurden in vierer Blöcke in einem Array namens `patterns` abgespeichert. Nun kann man bei jedem Input (das heisst Berührung eines Pins des Touchsensors) die nächstmögliche Auswahl bestimmen und falls die Auswahl nicht möglich ist, wird das Programm neu gestartet und die Symbolfolge lässt sich neu wählen.

```
1  #Hier werden die moeglichen Symbolfolgen im Array patterns
       gespeichert
2  patterns = [ [1,1,1,1], [1,2,1,2], [1,1,2,2], [3,3,3,3], [3,4,4,3]]
3  # nach der ersten Eingabe wird getestet, welche Eingaben nun noch
       moeglich sind
4  def validchoices2(first):
5  #Leeres Array choices merkt sich das gewaehlte Muster
6      choices = []
7  #Jedes Element im Array wird nun verglichen mit dem gewaelten
       ersten Muster
8      for i in range (0,len(patterns)):
9          if patterns[i][0] == first:
10 #Falls es moegliche Muster gibt mit dem gewaelten Muster an erster
       Stelle, werden alle naechsten moeglichen Symbole gemerkt und
       ausgegeben
11             nextchoice = patterns[i][1]
12             choices.append(nextchoice)
13     print(choices)
14     return choices
15 # moegliche Eingabe fuer die dritte Wahl
16 def validchoices3(first, second):
17     choices = []
18 #dasselbe Verfahren wie vorher: Die Wahl wird mit den moeglichen
       Muster in Patterns verglichen
19     for i in range (0, len(patterns)):
20 #Nun muss die erste Wahl und die zweite Wahl mit einem Muster in
       Patterns   bereinstimmen  , falls dies der Fall ist wird die
       naechste moegliche Wahl wieder ausgegeben
21         if patterns[i][0] == first and patterns[i][1] == second:
22             nextchoice = patterns[i][2]
23             choices.append(nextchoice)
24     print(choices)
25     return choices
26 # moegliche Eingabe fuer die vierte Wahl
27 def validchoices4(first, second, third):
28     choices =[]
29     for i in range (0, len(patterns)):
30 #Jetzt muessen alle 3 Wahlen mit einem Muster uebereinstimmen. Es
       gibt nun nur noch eine moegliche Symbolfolge, welche waehlbar
       ist. Diese wird ausgegeben.
31         if patterns[i][0] == first and patterns[i][1] == second and
       patterns[i][2] == third:
32             nextchoice = patterns[i][3]
33             choices.append(nextchoice)
34     print(choices)
35     return choices
```

6.7 Fertigstellung des Musterautomaten

Zum Schluss wurden alle Objekte miteinander verbunden und verkabelt (siehe Abbildung 34, 35). Die Kabel der Servo-Motoren mussten verlängert werden, damit diese ordentlich verstaut werden konnten. Um eine genügend grosse Stromversorgung der Motoren und Sensoren zu gewährleisten, wurde der Raspberry Pi und die genannten Komponenten separat angeschlossen. Alle Verkabelungen, bis auf die Touchsensoren wurden im Gehäuse des Automaten verstaut (siehe Abbildung 36, 37).

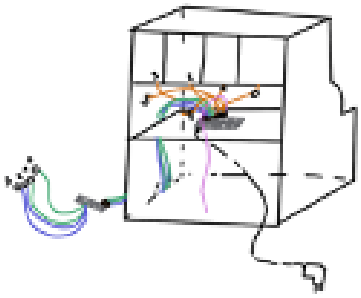


Abbildung 34: Skizze der Verkabelung

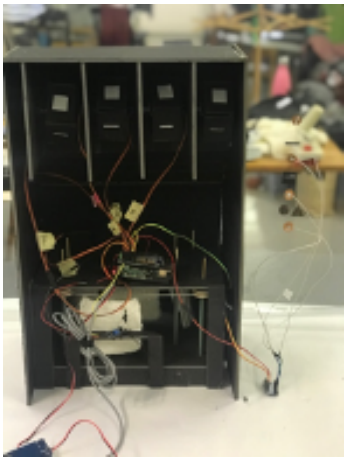


Abbildung 35: fertig angeschlossener Prototyp

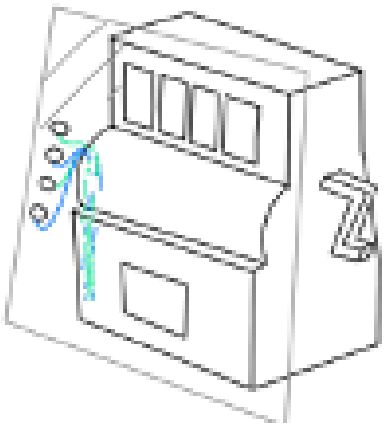


Abbildung 36: Skizze des fertigen Prototypen, verbunden mit Touchsensoren und hinter Scheibe

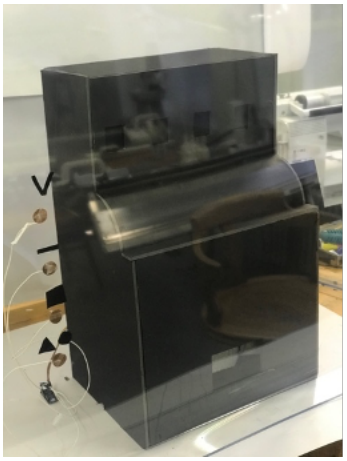


Abbildung 37: fertiger Prototyp, verbunden mit Touchsensoren und hinter Scheibe

6.7.1 Hülle aus Plexiglas

Damit die Mechanik des Musterautomaten besser ersichtlich ist, wurde entschieden die Hülle des Automaten aus Plexiglas zu bauen. Dadurch liegt der Fokus nicht mehr auf dem Modell des Automaten, sondern auf der eingebauten Mechanik.

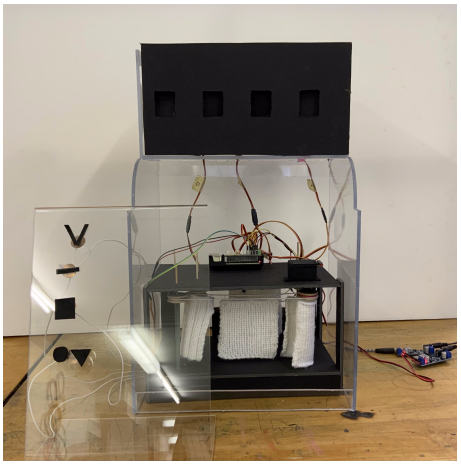


Abbildung 38: Fertiges Modell aus Plexiglas von vorne mit angeschlossenem Touchsensor

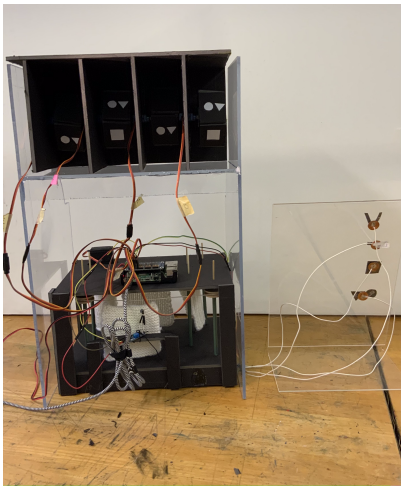


Abbildung 39: Mechanik von hinten gezeigt, mit angeschlossenem Touchsensor

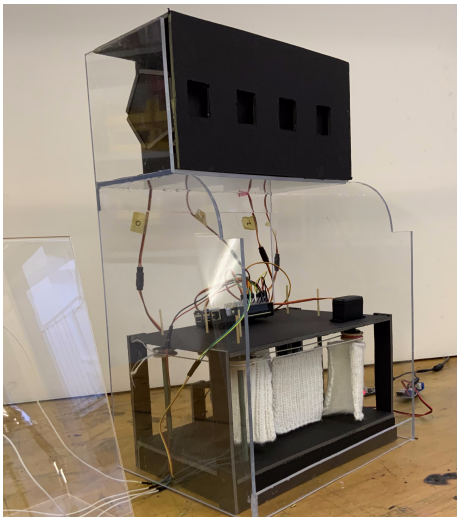


Abbildung 40: Ansicht von der Seite

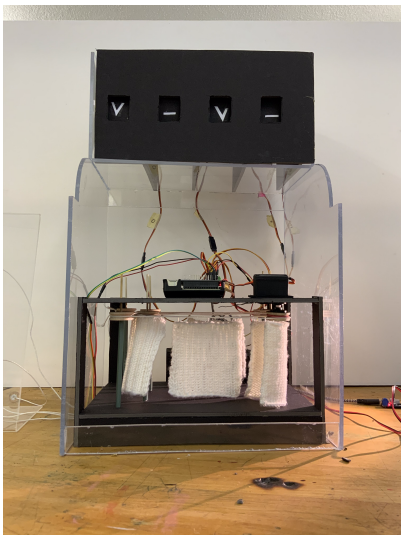


Abbildung 41: Nahaufnahme der mechanischen Komponente

Resultate

In der vorliegenden Arbeit wurde der Weg und die Entwicklung zum Bau einer interaktiven Schaufensterinstallation aufgezeigt. Im Arbeitsprozess stellte sich die Umsetzung und Lösung der gewählten Idee als sehr zeitintensiv dar. Daher ist das Endresultat dieser Arbeit nicht die effektive Umsetzung der Installation in einem Schaufenster, sondern die Erstellung eines voll funktionsfähigen Prototyps, der als Ausgangslage dienen kann, eine Schaufensterintallation auf Basis dieser Idee zu erstellen.
Die meiste Zeit wurde darauf verwendet, die mechanische Umsetzung abzubilden. Die softwaretechnische Ansteuerung der Komponenten stellte sich vergleichsweise einfach dar. Zwar wurde die Arbeit bei einem vollfunktionstüchtigen Prototyp belassen, dennoch beinhaltet dieser die Mechanik und die Idee der grossen Installation. Das heisst folgende Ziele wurden erreicht:

- Bewegung der Komponenten durch Servo-Motoren
- Interaktivität via Touchsensoren
- Ansteuerung durch die Scheibe
- Verbindung von Software und Mechanik

Der erstellte Prototyp funktionierte einwandfrei: Die Objekte wurden über Servo-Motoren bewegt. Die Wahl der Muster konnte über die Touchsensoren durch eine Scheibe erfolgen. Die Mechanik reagierte auf korrekte Weise auf die Ansteuerung durch die Software.

Diskussion

Rückblickend auf meine Arbeit habe ich die Umsetzung der Anfangsidee unterschätzt. Während dem Arbeitsprozess musste ich mich entscheiden den Fokus auf dem Prototyp zu belassen. Nichtsdestotrotz konnten meine gewünschten Erkenntnisse erreicht werden, wie die Einarbeitung und Vertiefung in die Programmierung, sowie die Verbindung der Programmierung mit der Mechanik.
Nutzte man den vorliegenden Prototypen für die effektive Umsetzung einer grösseren Schaufensterinstallation würden wahrscheinlich neue Problemstellungen auftauchen. Beim Übergang von einem verkleinerten Modell zu einer Schaufensterinstallation in Realgrösse stellen sich die folgenden Fragen respektive Probleme: Es müsste abgeklärt werden, ob die Stärke der Servo-Motoren auch für die Bewegung grösserer Objekte ausreichen würde. Müsste man allenfalls grössere Motoren verwenden? Auch könnte die Stromversorgung nicht mehr ausreichen. Durch die Aufskalierung der Objekte würde vielleicht nicht mehr dieselbe Mechanik funktionieren. Würde der Code immer noch funktionieren?
Die Idee der Umsetzung einer interaktiven Schaufenstergestaltung finde ich nach wie vor sehr faszinierend und ich hoffe, dass meine Arbeit als Inspiration dient, die vorliegende Idee in Wirklichkeit umzusetzen.

Glossar

- HDMI
  Unter dem High Definition Multimedia Interface (HDMI) versteht man den Standard zur gleichzeitgen Übertragung von Ton und Bild[15].

- USB
  Der Universal Serial Bus (USB) wird verwendeten um Daten zu übertragen oder für die Stromversorgung[16].

- Linux
  Linux ist ein Open-Source-Betriebssystem. Dieses basiert auf dem Unix Betriebssystem. Weiter gibt es zahlreiche Abänderungen(Linux-Distributionen),wie zum Beispiel das RaspiOS.[17]

- WLAN
  Durch den Begriff Wireless Local Area Network (WLAN) werden alle drahtlosen lokalen Netzwerke defniert.[18]

- Servo
  Kurzform von Servomotor. Ein Servomotor ist ein elektrischer Motor, welcher gut geregelt werden kann.[19]

- Library
  Eine Library auf Deutsch Bibliothek ist in der Programmierung eine Sammlung von Funktionen, welche als Hilffunktionen für den Code verwendet werden können[20].

- Tigerjython
  Tigerjython ist eine Webbasierte Entwicklungsumgebung für Python[21].

Literaturverzeichnis

[1] Bild stricksymbol. URL https://www.sockshype.com/strickschrift-lesen-geht-es/.

[2] Bild maschenprobe. URL https://lisibloggt.com/2015/07/31/lisitipps-maschenprobe-fur-eine-mutze-so-gehts/.

[3] offizielle raspberrypi webseite. URL https://www.raspberrypi.org/. (Zugriff: 14.05.2020).

[4] Mike Thomson and Peter Green. Raspberry pi os, 2020. URL https://www.heise.de/download/product/raspbian-91329. (Zugriff: 29.22.20).

[5] Ofizielle visual studio code webseite. URL https://code.visualstudio.com/.

[6] Bild aufbau eines servos. URL https://components101.com/servo-motor-basics-pinout-datasheet.

[7] adafruit learn adafruit 16-channel pwm/servo hat bonnet for raspberry pi, 2020. URL https://learn.adafruit.com/adafruit-16-channel-pwm-servo-hat-for-raspberry-pi. (Zugriff:24.05.2020).

[8] Schrittmotor, 2020. URL https://de.wikipedia.org/wiki/Schrittmotor. (Zugriff: 10.12.2020).

[9] Bild stepper motor. URL https://forum.arduino.cc/index.php?topic=550199.0.

[10] Adafruit dc and stepper motor hat for raspberry pi, 2015. URL https://learn.adafruit.com/adafruit-dc-and-stepper-motor-hat-for-raspberry-pi/using-stepper-motors. (Zugriff: 8.06.2020).

[11] Adafruit cap1188 breakout, 2014. URL https://learn.adafruit.com/adafruit-cap1188-breakout?view=all. (Zugriff: 24.05.2020).

[12] Hymel Shawn. Sprakfun apds-9960 rgb and gesture sensor hookup guide. URL https://learn.sparkfun.com/tutorials/apds-9960-rgb-and-gesture-sensor-hookup-guide/all. (Zugriff: 4.06.2020).

[13] Abbilding registrierkasse mit täfelchen. URL https://www.google.com/url?sa=i&url=https%3A%2F%2Fwww.ebay.de%2Fitm%2Fantike-Registrierkasse-Ladenkasse-alte-Kasse-National-NCR-Berlin-um-1900-%2F253672816742&psig=AOvVaw2X5tPocZKMDvYcInXat05o&ust=1604319090981000&source=images&cd=vfe&ved=0CAIQjRxqFwoTCLDLl8Co4ewCFQAAAAAdAAAAABAD.

[14] Abbildung registrierkasse mit rolle. URL http://www.moah.org/kaching/kaching.html?keepthis=true.

[15] Ascherman Tim. Hdmi-einfach erklärt, 2018. URL https://praxistipps.chip.de/hdmi-einfach-erklaert_100388. (Zugriff: 16.12.2020).

[16] Usb-wie funktioneirt es?was bedeuted es? URL https://www.conrad.ch/de/ratgeber/technik-einfach-erklaert/usb.html. (Zugriff:16.12.2020).

[17] Was ist und bedeuted linux? URL http://www.softselect.de/business-software-glossar/linux. (Zugriff:16.12.2020).

[18] Radkte Michael. Was ist das wlan?, 2018. URL https://www.ip-insider.de/was-ist-wlan-a-579430/. (Zugriff:16.12.2020).

[19] Definition servomotor. URL https://www.ingenieur.de/fachmedien/konstruktion/antriebstechnik/definition-servomotor/. (Zugriff:16.12.2020).

[20] Bibliothek(programmierung). URL https://deacademic.com/dic.nsf/dewiki/168872. (Zugriff:16.12.2020).

[21] Tygerjython. URL http://www.tigerjython.ch/index.php?inhalt_links=navigation.inc.php&inhalt_mitte=lernumgebung/einrichtung.inc.php.

[22] Adafruit learning system, adafruit 16-channel pwm/servo hat bonnet for raspberry pi. 2019.

[23] Abbildung hallsensor. URL https://www.bastelgarage.ch/hall-sensor-modul-mit-analogausgang.

[24] The official rasberrypi beginner's book, 2017. URL https://magpi.raspberrypi.org/books/beginners-1.

[25] Freeman Eric. *Programmieren lernen von Kopf bis Fuss*. o'reilly/ dpunkt.verlag GmbH, 2018.

[26] Vishal Dhayalan. electrobotify remote development on raspberry pi with vs code, 2019. URL https://electrobotify.wordpress.com/2019/08/16/remote-development-on-raspberry-pi-with-vs-code/. (Zugriff: 24.09.2020).

[27] Roine Jussi. Developing remotely on raspberry pi 4 and linux using visual studio code, 2020. URL https://jussiroine.com/2020/06/developing-remotely-on-raspberry-pi-4-and-linux-using-visual-studio-code/. (Zugriff: 24.09.2020).

[28] realvnc. URL https://www.realvnc.com/de/connect/download/viewer/. (Zugriff: 15.05.2020).

[29] dexter industries. connect from mac. URL https://www.dexterindustries.com/getting-started/using-the-pi/connect-to-your-raspberry-pi-from-a-mac/. (Zugriff: 15.05.2020).

[30] Woodman Justin. Using the apds-9960 rgb, proximity, and gesture sensor with the raspberry pi, 2014. URL `https://justinwoodman.wordpress.com/2014/11/15/using-the-apds-9960-rgb-proximity-and-gesture-sensor-with-the-raspberry-pi-2/` (Zugriff: 04.06.2020).

[31] Liske Thomas. github python-apds9960. URL `https://github.com/adafruit/Adafruit_CircuitPython_APDS9960`. (Zugriff: 05.06.2020).

Abbildungsverzeichnis

Anhang

Hier im Anhang werden alle Codes, die für den finalen Prototyp verwendet wurden aufgelistet. Teilweise sind Code-Stücke schon in den vorherigen Kapitel erklärt worden, deshalb werden diese hier nicht mehr genauer erläutert.

Finaler Code

In diesem Code sind die Symbolanzeigen und die Musteranzeigen kombiniert, auch habe ich noch einen Hebel der Kasse angefügt, der einfach über einen Continuous-Servo angesteuert wird. Auch werden andere Codes aus anderen Dateien importiert, welche später aufgelsitet werden.

```
#importiere benoetigte Libraries und Methoden
import RPi.GPIO as GPIO
import time
#importiere benoetigte Servokit Library
from adafruit_servokit import ServoKit
import board
import busio
import digitalio
#importiere Touchsensor library
from adafruit_cap1188.i2c import CAP1188_I2C
i2c = busio.I2C(board.SCL, board.SDA)
cap = CAP1188_I2C(i2c)
#importiere Symbolanzeige und zumUrsprung Funktion aus anderen Dateien
from T3_Symbolanzeige import Symbolanzeige,zumUrsprung
from T3_Symbolanzeige2 import Symbolanzeige2, zumUrsprung2
from T3_Symbolanzeige3 import Symbolanzeige3, zumUrsprung3
from T3_Symbolanzeige1 import Symbolanzeige1, zumUrsprung1
#importiere validchoices Funktionen aus anderer Datei
from T2_validChoices import validchoices2,validchoices3, validchoices4, patterns
#setzt GPIO pin 4 fest fuer den  Hallsensor
GPIO.setmode(GPIO.BCM)
#Nullpunkt des Hallsensors wird festgelegt
GPIO.setup(4, GPIO.IN, pull_up_down=GPIO.PUD_DOWN)
#Servo Hat, besitzt 16 Anschluesse, diese werden als channels im Code definiert
kit = ServoKit(channels=16)

GPIO.add_event_detect(4,GPIO.RISING)
#definiere callback Funktion, welche sobald Hallsensor Magnetfeld entdeckt ausgefuehrt wird.
def my_callback(n):
#Continuous_servo der Musteranzeige stoppt
    kit.continuous_servo[4].throttle = 0
#erstelle leeres Array names "kombination", fuer gwuenschte Musterkombination, fuer die Ueberpruefung des Codes wird dieses Array ausgegeben
    kombination = []
    while True:
#Sobald While-Schleife erneut durchlaufen wird, wird das
```

```
    Kombination-Array geleert
36          kombination.clear()
37          print("Wahl 1")
38  #erste Symbolwahl
39          first = int(Symbolanzeige(0))
40  #kommt in kombination-Array, gleichzeitig wird geschaut, welche
    Moeglichkeiten fuer die naechste Wahl besteht
41          validchoices2(first)
42          kombination.append(first)
43          time.sleep(0.5)
44          print("Wahl 2")
45  #zweite Symbolwahl
46          print(validchoices2(first))
47          second = int(Symbolanzeige1(1))
48  #ueberprueft, ob zweite Wahl moeglich ist, falls True in
    kombinationen-Array
49          if second in validchoices2(first):
50              validchoices3(first, second)
51              kombination.append(second)
52  #Falls nicht moelich: Programm gibt Fehler aus und While-Schlaufe
    von neuem durchlaufen
53          else:
54              print("\n")
55              print("Fehler2")
56              print("\n")
57  #bereits gewaehlte Symbole, werden zum Ursprung zurueckgesetzt
58              zumUrsprung(0)
59              zumUrsprung(1)
60              continue
61          time.sleep(0.5)
62          print("Wahl3")
63  #dritte Symbolwahl
64          print(validchoices3(first, second))
65          third = int(Symbolanzeige2(2))
66  #ueberpruefen ob moeglich, falls ja in Array
67          if third in validchoices3(first, second):
68              validchoices4(first,second, third)
69              kombination.append(third)
70          else:
71              print("\n")
72              print("Fehler3")
73              print("\n")
74  #bereits gewaehlte Symbole zum Ursprung
75              zumUrsprung(0)
76              zumUrsprung(1)
77              zumUrsprung(2)
78              continue
79          time.sleep(0.5)
80          print("Wahl4")
81  #die vierte Symbolwahl
82          print(validchoices4(first,second,third))
83          last = int(Symbolanzeige3(3))
84  #ueberpruefen ob moeglich
85          if last in validchoices4(first,second,third):
86              kombination.append(last)
87  #hier wird Array ausgegeben
88              print(kombination)
89
```

```
90 #Hebel ueber continuous-Servo gesteuert
91             kit.continuous_servo[6].throttle = 1
92             time.sleep(1)
93             kit.continuous_servo[6].throttle = 0
94 
95         else:
96 #Fehlerausgegeben und While-Schlaufe von neuem durchlaufen
97             print("\n")
98             print("Fehler4")
99             print("\n")
100 #alle Symbole zum Ursprung zuruecksetzen
101             zumUrsprung(0)
102             zumUrsprung1(1)
103             zumUrsprung2(2)
104             zumUrsprung3(3)
105             continue
106 #fuer jede Musterkombination wird der continuous-Servo an eine
    andere Stelle bewegt, zeigt Muster fuenf Sekunden lang an und
    dreht danach zurueck zum Ursprung, danach wird While-Schlaufe
    verlassen
107         if kombination == [1,1,1,1]:
108             kit.continuous_servo[4].throttle = 1
109             time.sleep(1)
110             kit.continuous_servo[4].throttle = 0
111             time.sleep(5)
112             kit.continuous_servo[4].throttle = -1
113             time.sleep(1.5)
114             kit.continuous_servo
115             break
116         elif kombination == [1,2,1,2]:
117             kit.continuous_servo[4].throttle = 1
118             time.sleep(2)
119             kit.continuous_servo[4].throttle = 0
120             time.sleep(5)
121             kit.continuous_servo[4].throttle =-1
122             time.sleep(2.5)
123             kit.continuous_servo[4].throttle = 0
124             break
125         elif kombination == [1,1,2,2]:
126             kit.continuous_servo[4].throttle = 1
127             time.sleep(3)
128             kit.continuous_servo[4].throttle = 0
129             time.sleep(5)
130             kit.continuous_servo[4].throttle =-1
131             time.sleep(3.5)
132             kit.continuous_servo[4].throttle = 0
133             break
134         elif kombination == [3,3,3,3]:
135             kit.continuous_servo[4].throttle = 1
136             time.sleep(4)
137             kit.continuous_servo[4].throttle = 0
138             time.sleep(5)
139             kit.continuous_servo[4].throttle =-1
140             time.sleep(4.5)
141             kit.continuous_servo[4].throttle = 0
142             break
143 #While-Schlaufe zu ende, alle Symbole zum Ursprung zuruecksetzten
144     print("finished")
```

```
145     zumUrsprung(0)
146     zumUrsprung1(1)
147     zumUrsprung2(2)
148     zumUrsprung3(3)
149 #continuous_servo beginnt von Neuem zu drehen, bis Hallsensor
        wieder Magnetfeld detektiert
150     kit.continuous_servo[4].throttle = 1
151 GPIO.add_event_callback(4, my_callback)
152
153
154 #Wenn man Programm neu startet wird diese Zeile zuerst ausgefuehrt
        -> Start des Codes
155 kit.continuous_servo[4].throttle = 1
156 #Damit Programm staendig l uft und Hallsensor immer Werte misst
        muss While-Schleife unendlich lang ausgefuehrt werden
157 while True:
158     time.sleep(1)
159     print(".")
```

Auswahleinschränkung

Dieser Code wird bereits im Kapitel 6 "Bau des Modells"genauer erklärt, deshalb wird er hier nicht mehr kommentiert.

```
1
2 patterns = [ [1,1,1,1], [1,2,1,2], [1,1,2,2],[3,3,3,3] ]
3
4 def validchoices2(first):
5     choices = []
6     for i in range (0,len(patterns)):
7         if patterns[i][0] == first:
8             nextchoice = patterns[i][1]
9             choices.append(nextchoice)
10    print(choices)
11    return choices
12
13
14
15 def validchoices3(first, second):
16    choices = []
17    for i in range (0, len(patterns)):
18        if patterns[i][0] == first and patterns[i][1] == second:
19            nextchoice = patterns[i][2]
20            choices.append(nextchoice)
21    print(choices)
22    return choices
23
24
25 def validchoices4(first, second, third):
26    choices =[]
27    for i in range (0, len(patterns)):
28        if patterns[i][0] == first and patterns[i][1] == second and
      patterns[i][2] == third:
29            nextchoice = patterns[i][3]
30            choices.append(nextchoice)
31    print(choices)
32    return choices
```

Symbolanzeigen
Hier ist der Code für die Symbolanzeigen. Dieser wurde im Kapitel 6 schon kommentiert. Die neuen Code-Stücke sind einfach Wiederholungen des Codes in neuen Dateien mit neuen Zeit- und Geschwindigkeitsangaben für die einzelnen Servos. (Symbolanzeige 1-4)

```

import time
from adafruit_servokit import ServoKit
import board
import busio
import digitalio
import T3_Symbolanzeige
from adafruit_cap1188.i2c import CAP1188_I2C
i2c = busio.I2C(board.SCL, board.SDA)
cap = CAP1188_I2C(i2c)

kit = ServoKit(channels=16)
choices = 0

def Symbolanzeige(a):
    i = 0
    global choices
    while i < 1:
        if cap[1].value:


            kit.continuous_servo[a].throttle= 0.1
            time.sleep(0.25)
            kit.continuous_servo[a].throttle=0
            i = i + 1
            choices = 1

            return choices
        if cap[2].value:


            kit.continuous_servo[a].throttle= 0.1
            time.sleep(0.48)
            kit.continuous_servo[a].throttle = 0
            i = i + 1
            choices = 2
            return choices

        if cap[3].value:


            kit.continuous_servo[a].throttle= 0.1
            time.sleep(0.7)
            kit.continuous_servo[a].throttle = 0
            i = i + 1
            choices = 3
            return choices

        if cap[4].value:


```

```
            kit.continuous_servo[a].throttle= 0.1
            time.sleep(0.9)
            kit.continuous_servo[a].throttle = 0
            i = i + 1
            choices = 4
            return choices


def zumUrsprung(a):

        if choices == 1:
            kit.continuous_servo[a].throttle= - 0.2
            time.sleep(0.25)
            kit.continuous_servo[a].throttle=0
        if choices == 2:
            kit.continuous_servo[a].throttle= - 0.2
            time.sleep(0.45)
            kit.continuous_servo[a].throttle = 0
        if choices == 3:
            kit.continuous_servo[a].throttle= - 0.2
            time.sleep(0.6)
            kit.continuous_servo[a].throttle = 0

        if  choices == 4:
            kit.continuous_servo[a].throttle = - 0.2
            time.sleep(0.75)
            kit.continuous_servo[a].throttle = 0
```

Listing 1: Symbolanzeige1

```
import time
from adafruit_servokit import ServoKit
import board
import busio
import digitalio
import T3_Symbolanzeige
from adafruit_cap1188.i2c import CAP1188_I2C
i2c = busio.I2C(board.SCL, board.SDA)
cap = CAP1188_I2C(i2c)

kit = ServoKit(channels=16)
choices1 = 0


def Symbolanzeige1(a):
    i = 0
    global choices1
    while i < 1:
        if cap[1].value:
            kit.continuous_servo[a].throttle= 0.1
            time.sleep(0.25)
            kit.continuous_servo[a].throttle=0
            i = i + 1
            choices1 = 1
            return choices1
        if cap[2].value:


```

```
            kit.continuous_servo[a].throttle= 0.2
            time.sleep(0.45)
            kit.continuous_servo[a].throttle = 0
            i = i + 1
            choices1 = 2
            return choices1

        if cap[3].value:


            kit.continuous_servo[a].throttle= 0.2
            time.sleep(0.6)
            kit.continuous_servo[a].throttle = 0
            i = i + 1
            choices1 = 3
            return choices1

        if cap[4].value:


            kit.continuous_servo[a].throttle= 0.2
            time.sleep(0.7)
            kit.continuous_servo[a].throttle = 0
            i = i + 1
            choices1 = 4
            return choices1


def zumUrsprung1(a):
        if choices1 == 1:
            kit.continuous_servo[a].throttle= - 0.2
            time.sleep(0.2)
            kit.continuous_servo[a].throttle= 0
        if choices1 == 3:
            kit.continuous_servo[a].throttle= - 0.2
            time.sleep(0.45)
            kit.continuous_servo[a].throttle = 0
        if choices1 == 4:
            kit.continuous_servo[a].throttle= - 0.2
            time.sleep(0.55)
            kit.continuous_servo[a].throttle = 0
        if  choices1 == 5:
            kit.continuous_servo[a].throttle = - 0.1
            time.sleep(2)
            kit.continuous_servo[a].throttle = 0
```

Listing 2: Symbolanzeige2

```
import time
from adafruit_servokit import ServoKit
import board
import busio
import digitalio
import T3_Symbolanzeige
from adafruit_cap1188.i2c import CAP1188_I2C
i2c = busio.I2C(board.SCL, board.SDA)
cap = CAP1188_I2C(i2c)

```

```
kit = ServoKit(channels=16)
choices2 = []

def Symbolanzeige2(a):
    i = 0
    while i < 1:
        if cap[1].value:


            kit.continuous_servo[a].throttle= 0.2
            time.sleep(0.1)
            kit.continuous_servo[a].throttle=0
            i = i + 1
            choices2.insert(0, 1)

            return choices2[0]
        if cap[2].value:


            kit.continuous_servo[a].throttle= 0.2
            time.sleep(0.3)
            kit.continuous_servo[a].throttle = 0
            i = i + 1
            choices2.insert(0, 2)
            return choices2[0]

        if cap[3].value:


            kit.continuous_servo[a].throttle= 0.2
            time.sleep(0.5)
            kit.continuous_servo[a].throttle = 0
            i = i + 1
            choices2.insert(0, 3)
            return choices2[0]

        if cap[4].value:


            kit.continuous_servo[a].throttle= 0.2
            time.sleep(0.7)
            kit.continuous_servo[a].throttle = 0
            i = i + 1
            choices2.insert(0, 4)
            return choices2[0]


def zumUrsprung2(a):

        if choices2[0] == 1:
            kit.continuous_servo[a].throttle= - 0.2
            time.sleep(0.2)
            kit.continuous_servo[a].throttle=0
        if choices2[0] == 2:
            kit.continuous_servo[a].throttle= - 0.2
            time.sleep(0.45)
            kit.continuous_servo[a].throttle = 0
        if choices2[0] == 3:
```

```
            kit.continuous_servo[a].throttle= - 0.2
            time.sleep(0.6)
            kit.continuous_servo[a].throttle = 0

        if  choices2[0] == 4:
            kit.continuous_servo[a].throttle = - 0.2
            time.sleep(0.82)
            kit.continuous_servo[a].throttle = 0
```

Listing 3: Symbolanzeige3

```
import time
from adafruit_servokit import ServoKit
import board
import busio
import digitalio
import T3_Symbolanzeige
from adafruit_cap1188.i2c import CAP1188_I2C
i2c = busio.I2C(board.SCL, board.SDA)
cap = CAP1188_I2C(i2c)

kit = ServoKit(channels=16)
choices3 = []

def Symbolanzeige3(a):
    i = 0
    while i < 1:
        if cap[1].value:


            kit.continuous_servo[a].throttle= 0.1
            time.sleep(0.25)
            kit.continuous_servo[a].throttle=0
            i = i + 1
            choices3.insert(0,1)

            return choices3[0]
        if cap[2].value:


            kit.continuous_servo[a].throttle= 0.1
            time.sleep(0.4)
            kit.continuous_servo[a].throttle = 0
            i = i + 1
            choices3.insert(0,2)
            return choices3[0]

        if cap[3].value:


            kit.continuous_servo[a].throttle= 0.1
            time.sleep(0.65)
            kit.continuous_servo[a].throttle = 0
            i = i + 1
            choices3.insert(0, 3)
            return choices3[0]

        if cap[4].value:
```

```
48
49
50              kit.continuous_servo[a].throttle= 0.1
51              time.sleep(0.78)
52              kit.continuous_servo[a].throttle = 0
53              i = i + 1
54              choices3.insert(0, 4)
55              return choices3[0]
56
57
58  def zumUrsprung3(a):
59
60          if choices3[0] == 1:
61              kit.continuous_servo[a].throttle= - 0.2
62              time.sleep(0.2)
63              kit.continuous_servo[a].throttle=0
64          if choices3[0] == 2:
65              kit.continuous_servo[a].throttle= - 0.2
66              time.sleep(0.43)
67              kit.continuous_servo[a].throttle = 0
68          if choices3[0] == 3:
69              kit.continuous_servo[a].throttle= - 0.2
70              time.sleep(0.65)
71              kit.continuous_servo[a].throttle = 0
72
73          if  choices3[0] == 4:
74              kit.continuous_servo[a].throttle = - 0.2
75              time.sleep(0.78)
76              kit.continuous_servo[a].throttle = 0
```

Listing 4: Symbolanzeige4