

Physical Computing

Projektbericht: Sockensortieranlage

SS 2015

Projektteilnehmer:

Niels Thiele

Sebastian Zeh

Damian Uczulko

Inhaltsverzeichnis

1	EINFÜHRUNG	3
2	PROJEKTBESCHREIBUNG	4
2.1	Konzept	4
2.2	Informationen zur Umsetzung.....	5
2.3	Realisierung	6
2.4	Funktionsbeschreibung	7
3	KOMPONENTEN.....	9
3.1	Aufbau des Gerüsts	9
3.2	Roboterarm	11
3.3	Positionsbestimmung	12
3.3.1	Potentiometereinbau.....	12
3.3.2	Position auslesen.....	13
3.4	Motoransteuerung	14
3.4.1	Hardwareumsetzung	14
3.4.2	Schaltungsaufbau	18
3.4.3	Softwareumsetzung	20
3.5	RFID	21
3.5.1	RFID Reader/Writer	21
3.5.2	RFID Tags.....	22
3.5.3	RFID-Tags Auswertung	25
3.6	Zusätzliche Anbauten / Probleme	26
3.6.1	Betriebsbereitschaft / NOTAUS – Funktion	26
3.6.2	Greifarm-Verbesserungen (Klett und Nadeln)	27
3.7	Kommunikation der Arduino-Boards	27
4	SCHLUSSWORT	29
5	ABBILDUNGSVERZEICHNIS	30
6	ANHANG	31

1 Einführung

Im Rahmen des Wahlfachs Physical Computing, das im weitesten Sinne die Interaktion von Systemen durch die Verwendung von Hardware und Software auf Ereignisse in der realen, analogen Welt beschreibt, wurde eine neue innovative Idee gesucht. Unter Physical Computing werden Systeme verstanden, die sich mit der Beziehung zwischen dem Menschen und der digitalen Welt befassen. Der Begriff wird meistens für Projekte mit einem künstlerischen oder Designhintergrund oder für Do-it-yourself-Hobbyprojekte (kurz: DIY) verwendet.¹ Über Sensoren und Mikrocontroller werden Daten aufgenommen und verwertet, um damit analoge oder digitale Signale, für Software-Anwendungen verfügbar zu machen und/oder elektromechanische Geräte wie Motoren, Servos, Leuchtdioden oder andere Hardware zu steuern.

In Gruppen von 3 Personen sollte nun ein Projekt von der Planung über die Konstruktion bis hin zur Inbetriebnahme umgesetzt werden.

Unsere Idee: Eine Sockensortiermaschine!



Abbildung 1-1 - Fertige Anlage

¹ Vgl. https://de.wikipedia.org/wiki/Physical_Computing Stand:07.07.2015

2 Projektbeschreibung

Vor dem Projektstart wurde ein Konzept für eine mögliche Realisierung des Sockensortierers erstellt. Anhand dieser groben Modell-Darstellung konnten erste Vorstellungen über Aufgaben, Umsetzbarkeit und Realisierbarkeit des Projektes erörtert werden. Es wurden erste Aufgabengebiete unter den Projektteilnehmer diskutiert und entsprechend zugewiesen. Durch regelmäßige Treffen wurden die bereits erreichten Teilziele zusammengetragen, besprochen bzw. weitere Aufgaben überarbeitet und dementsprechend neu verteilt.

Diese Aufgabenbeschreibung wurde auch den Professoren als Anhaltspunkt bezüglich erfolgreicher Konzeption (am Anfang des Projektes) und anschließender Umsetzung (am Abgabetermin) übergeben.

2.1 Konzept

Sobald Socken im Sockenbehälter landen, wird ein Signal (Lichtschranke) gesendet, welches den Roboterarm startet. Der Roboterarm scannt mit Hilfe von RDIF Lesern in einem vorgegebenen Bereich nach RFID Tags in den Socken. Sobald ein Tag erkannt wird, „greift“ der Roboterarm zu. Dabei wird durch die 2 RFID Leser geprüft, ob nur eine! Socke genommen wurde. Falls nicht, also es wurden mehrere Socken zum Beispiel genommen, werden die Socken zurück in den Behälter geworfen. Bei nur einer Socke „fährt“ der Roboterarm weiter zu einem RFID Schreiber um den Tag zu aktualisieren (Anzahl der Waschgänge, Lebenszeit der Socke, usw.). Anschließend wird die Socke in eine Box (separaten Bereich) abgelegt und der RFID Tag bzw. die Box der Socke wird sich gemerkt. Dieser Vorgang wird wiederholt. Bei gleichem RFID Tag wird die nächste Socke in die gleiche Box gelegt. Bei unterschiedlichem RFID Tag wird die Socke in eine andere Box abgelegt. Sind alle Socken sortiert, geht der Arm in Grundstellung zurück (Lichtschranke im Behälter). Ansteuerung aller Aktoren und das Einlesen der Sensoren und anschließende Auswertung, erfolgt mit Raspberry Pi.

Skizze:

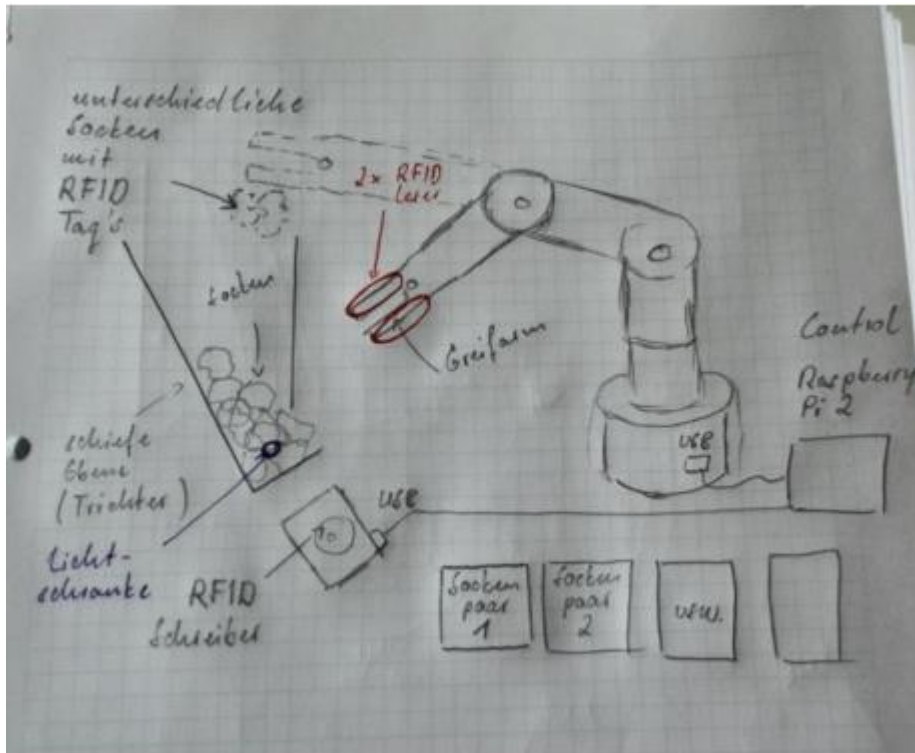


Abbildung 2-1 - Konzeptskizze

2.2 Informationen zur Umsetzung

Die Grundidee der Kommissionierung von Socken welches durch unser Konzept dargestellt wurde, konnte eingehalten werden. Doch gab es einige technische Änderungen bei der Umsetzung. Auf diese Änderungen und die während der Testphase aufgetretenen Probleme wird anschließend eingegangen.

Abgrenzung zum Konzept:

Aufgrund des begrenzten Budgets wurden auf einige weitere technische Details verzichtet.

Nicht benötigt wurden:

- Lichtschranke am Trichter → Sortierer wird manuell aus Programm gestartet.
- 2 RFID Reader → ersetzt durch MifareRC522 Reader/Writer wurde von Hochschule gestellt
- RFID Writer

Zudem waren folgende zusätzliche Bauteile nötig:

- Zweites Arduino UNO Board – gestellt von der Hochschule → da IOs zum Einlesen aller Motorstellungen und Ansteuerung der Motoren bzw. der Anschluss des RFID Reader/Writer nicht ausreichten.
- Bausteine L293D zur Motoransteuerung
- Zusätzliche (kleine) Transponder zum Verkleben in den Socken

2.3 Realisierung

Wir gaben unseren Hauptbaugruppen Namen. Dies klingt auf den ersten Blick kurios, ist aber während der intensiven Testphase des Aufbaus sehr hilfreich gewesen. Hiermit konnte z.B. eindeutig eine bestimmte Anschlussnummer zugewiesen werden. Außerdem ist es einprägsamer in der Funktionsbeschreibung der Dokumentation, mit Namen der unterschiedlichen Arduino Boards zu arbeiten, als ständig zwischen Board 1 und 2, hin und her zu wechseln.

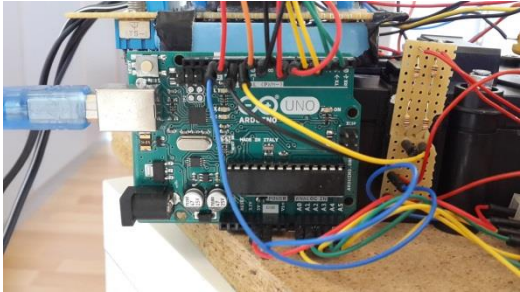
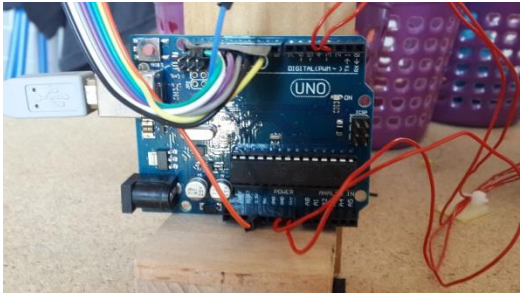
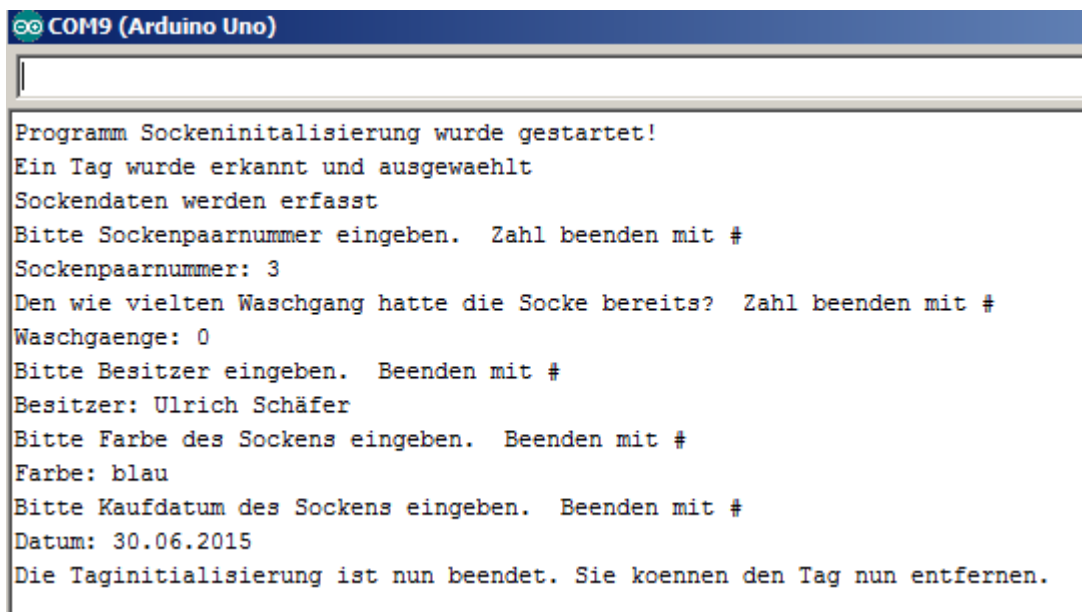
Baugruppe	Name	Funktion
Roboterarm	„Der Deutsche“	Kommissionieren der Socken
Arduino UNO Nr. 1 	„Der Schwede“	Motorstellungen auslesen, Motoren ansteuern, Pegelzustand einlesen (Sockenpaarnummer erkennen)
Arduino UNO Nr. 2 	„Der Japaner“ (asiatische Version)	RFID Reader/Writer, Pegelzustand für Sockenpaar ausgeben

Abbildung 2-2 - Arduino Board "Schwede" am Roboterarm

Abbildung 2-3 - Arduino Board "Japaner" am RFID Halterung

Initialisierung:

Bevor das Programm überhaupt gestartet wird, ist es nötig die einzelnen Socken zu initialisieren. Dafür wurde ein eigenes Arduino Programm geschrieben, welches über den „Japaner“ ausgeführt werden kann. Dieses ermöglicht es den Benutzer manuell über die Konsole, Daten wie z.B. Sockenpaarnummer, Besitzer, Kaufdatum, Farbe, usw. (nach Kundenwunsch erweiterbar) einzugeben und auf dem RFID-Tag zu speichern, welcher in der Socke verklebt ist. Eine Zuordnung der Socken erfolgt über die Sockenpaarnummer. Je eine eindeutige Nummer pro Sockenpaar.



```
COM9 (Arduino Uno)

Programm Sockeninitialisierung wurde gestartet!
Ein Tag wurde erkannt und ausgewaehlt
Sockendaten werden erfasst
Bitte Sockenpaarnummer eingeben. Zahl beenden mit #
Sockenpaarnummer: 3
Den wie vielen Waschgang hatte die Socke bereits? Zahl beenden mit #
Waschgaenge: 0
Bitte Besitzer eingeben. Beenden mit #
Besitzer: Ulrich Schäfer
Bitte Farbe des Sockens eingeben. Beenden mit #
Farbe: blau
Bitte Kaufdatum des Sockens eingeben. Beenden mit #
Datum: 30.06.2015
Die Taginitialisierung ist nun beendet. Sie koennen den Tag nun entfernen.
```

Abbildung 2-4: Konsolenbild der Sockeninitialisierung

2.4 Funktionsbeschreibung

Der Sockensortierer wird manuell vom Arduino Programm (Code zur Motoransteuerung) gestartet. Als Startbefehl, wird die Anzahl der zu sortierenden Sockenpaare übergeben. Nun fährt der Arm zuerst in eine Grundposition. Der Weg für diese Grundposition ist je nach Ausgangslage (Sockenplatz 1, Sockenplatz 2, Sockenplatz 3 oder nicht definierte Lage) unterschiedlich. Dies bedeutet, die Motoren müssen so angesteuert werden, dass der Arm bei der Fahrt in die Ausgangsposition keine Hindernisse berührt. In unseren Fall wurden deshalb zuerst nur die ersten vier Motoren angesteuert, jedoch nicht der Motor 5 welcher die Drehung um die eigene Achse beeinflusst.

Von dieser Grundposition fährt der Arm, wir haben ihn „den Deutschen“ genannt, zum Trichter welcher die Socken beinhaltet. Er greift sich einen Socken (Ansteuerung des Motors 1) und führt diesen Richtung RFID Reader/Writer. Währenddessen wird der Motor des Greifers immer wieder nachgestellt, um den Socken nicht zu verlieren.

Der Arm fährt nun immer wieder eine Routine im Einflussbereich des hochfrequenten Feldes des RFID Readers ab, damit der Transponder welcher im Socken angebracht ist überhaupt erkannt wird.

Einschub: Die Kommunikation zwischen den zwei verwendeten Arduino Boards, konnte nicht über ein einfaches HIGH Signal realisiert werden. Da „der Schwede“, nicht über genügend digitale Eingänge verfügt. Deshalb mussten drei Zustände und zuvor ein Stoppsignal über zwei Pins umgesetzt werden. Dies wurde über eine Pegelzeitsteuerung im Code realisiert. (detaillierte Beschreibung siehe Codeanhang)

Sobald eine Socke erkannt wurde, wird vom „Japaner“ zum „Schweden“ ein Pegelsignal (HIGH) für eine Sekunde gesendet und der Arm hält an. Es wird aus dem Transponder die zuvor initialisierte Sockenpaarnummer ausgewertet. Diese wird über einen Pegelzustand vom „Japaner“ zum „Schweden“ gesendet. Des Weiteren werden alle gespeicherten Daten der Socken beim Auslesen auf der seriellen Schnittstelle dargestellt. Sequenziell werden die Daten der Socken gelesen bzw. modulierte Daten zurückgeschrieben. In unserem konkreten Fall wurde die Anzahl der Waschgänge der Socke erhöht. Nach einer Verzögerungszeit von zwei Sekunden, werden die Pegel zurückgesetzt. Das Auslesen des Transponders wird anschließend für zehn Sekunden „blockiert“. Damit ist eine erneute Erkennung des gleichen Tags sichergestellt.

Mit Übergabe des Signalpegels vom „Japaner“ zum „Schweden“ wird nun im Arduino Motorcode, die Positionsbestimmung des Sockens ausgewählt. Eine zeitliche Verzögerung der Motoransteuerungen, stellt den zurückgesetzten Zustand der Pegel sicher → Funktion driveto() kann ausgeführt werden.

„Der Deutsche“ fährt nun je nach Sockenpaarnummer, die jeweilige Endposition zur Ablage der Socken an. Für jedes Sockenpaar wurde eine eigene Routine (mit unterschiedlichen Motorstellungen in Kombination zueinander) zur erfolgreichen Ablage des Sockens implementiert.

Nach der Ablage eines Sockens wird die Grundposition des Arms wieder angesteuert und der Ablauf beginnt von vorne. Dieser Zyklus wiederholt sich solange, bis die vorher eingegebene Sockenpaaranzahl erreicht ist.

Um genau diesen nahezu reibungslosen Ablauf des automatisierten Systems zu ermöglichen, war ein enormer Testaufwand nötig!

3 Komponenten

3.1 Aufbau des Gerüsts

Der Aufbau der Sockensortieranlage ist aus „Holzresten“ aufgebaut. Die Priorität lag in der Funktionsweise der Anlage und nicht auf einem fachmännisch geschreinerten Aufbau. Der komplette Aufbau steht auf einer robusten Holzplatte, damit er transportfähig ist. Um einen ersten Testablauf zu ermöglichen, musste der Roboterarm fixiert werden. Anschließend wurden die restlichen Komponenten auf der Holzplatte vorerst variabel (mit „Klett“) montiert, damit der Roboterarm seinen vollen Aktionsraum ausschöpfen konnte. Dafür wurde der Arm manuell (mit der Fernbedienung) in die unterschiedlichen Positionen verfahren. Ein zu großer Hebel des Arms (inklusive einer Socke), führte die Motoren an Ihre Leistungsgrenze. Das heißt, der Arm konnte seine Bewegungen nicht mehr bzw. kaum noch ausführen, was zum ständigen Anpassen, der auf dem Holzbrett angebrachten Komponenten führte → Hebel und Aktionsraum optimieren!

Für die Sammlung der Socken wurden mehrere Konzepte aus Pappe ausprobiert. Nachdem die Lösung eines Trichterlagers überzeugte, wurde dieser aus Holz mit der Stichsäge angefertigt und verleimt. Eine Innenseite des Trichters ist mit einer „glatten“, der Rest mit einer „rauen“ Oberfläche versehen, damit die Socken bei der Ablage besser in den Trichter fallen.



Abbildung 3-1 - Trichter

Der RFID Reader/Writer ist an einer Laternenkonstruktion aus Holz angebracht. Dies ist nötig, damit der Arm mit dem Socken eine Auf- und Ab-Routine abfahren kann. Zudem wurde der „Japaner“ unterhalb des Aufbaus befestigt um eine kurze Leitungsverlegung von Arduino und RFID Reader/Writer zu ermöglichen. Kurze Leitungen führen zu einer verbesserten Störfestigkeit.

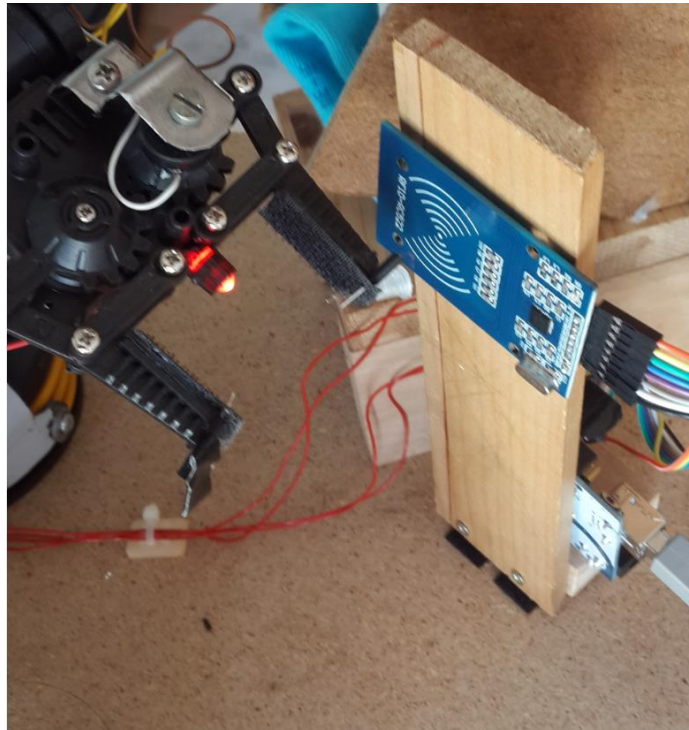


Abbildung 3-2 - Laternenkonstruktion mit RFID-Reader/Writer

Als Behälter für die sortierten Socken wurden verschiedene Varianten ausprobiert. Die Meisten waren jedoch entweder zu groß (der Deutsche konnte Sie durch seinen eingeschränkten Radius nicht mehr alle erreichen), oder zu flach (Socken sind herausgefallen). Nun werden die Sockenpaare in kleine Waschkörbe, die sowohl optimale Höhe als auch Größe haben sortiert.



Abbildung 3-3 - Wäschekörbe mit sortierten Socken

3.2 Roboterarm

Bei dem Roboterarm handelt es sich um den Roboterarmbausatz KSR10 von Velleman.



Abbildung 3-4 - Roboterarm KSR10 mit Kontrollgerät

Dieser Roboterarm wird an fünf Gelenken über fünf Motoren bewegt und standardmäßig über ein Kontrollgerät angesteuert. Die maximale Tragfähigkeit beträgt 100g. Die Produktabmessungen betragen 40,4 x 29,6 x 9,6 cm. Aus diesen Daten kann man entnehmen, dass die Leistung zum Bewegen der geforderten Last (Socken) gerade noch ausreichend ist.

Die Montage des Roboterarms erfolgte nach Bauanleitung.

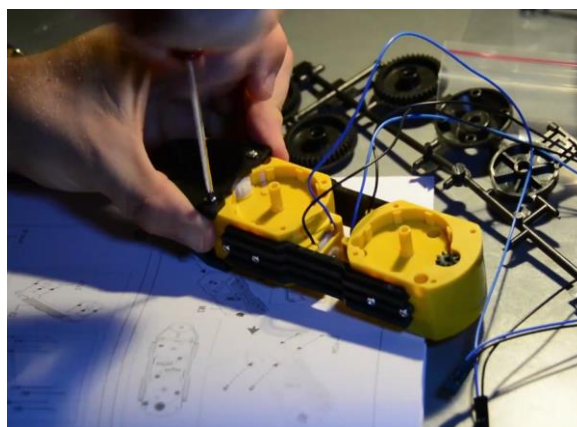


Abbildung 3-5 - Roboterarmmontage

3.3 Positionsbestimmung

3.3.1 Potentiometereinbau

Damit der Roboterarm kontrolliert über ein Steuerungsprogramm angesteuert werden kann, ist es notwendig eine Positionsbestimmung einzubauen. Deshalb wurde im Rahmen der Projektarbeit an jedem Gelenk jeweils ein Potentiometer montiert.

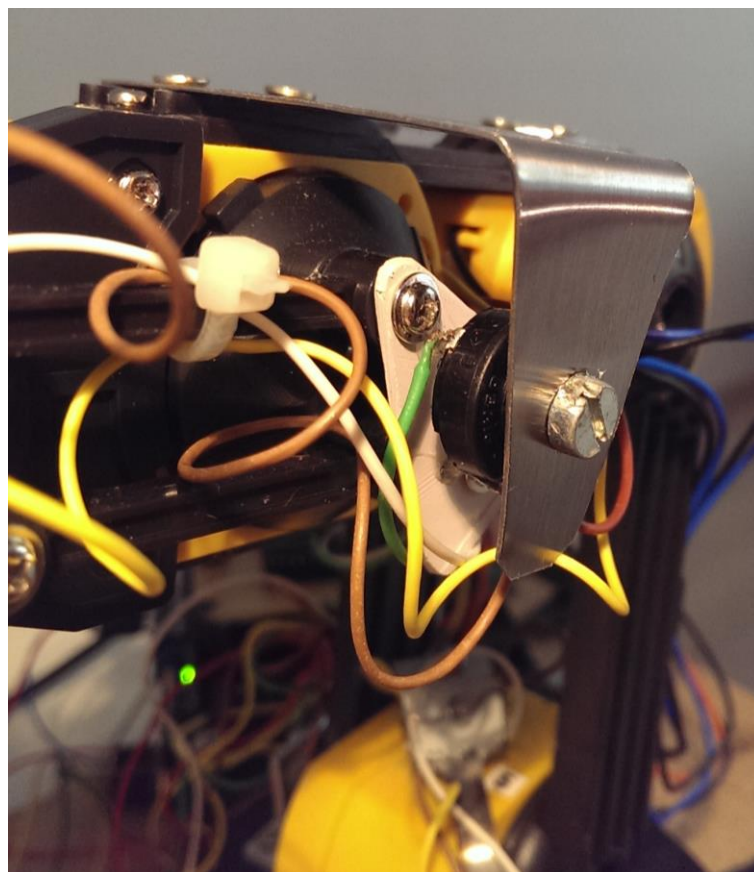


Abbildung 3-6 - Potentiometermontage

Für die Montage hat man die Potentiometer an den jeweiligen Gelenken befestigt. Damit die Motorstellung bzw. die Drehbewegung an den Potentiometern erkannt werden kann muss man das Regelglied über einen Bügel fixieren.

Die Anschlüsse der Potentiometer müssen an GND (Ground), +5V und jeweils einen „ANALOG IN“ verschaltet werden.

3.3.2 Position auslesen

Für das Auslesen der Positionen wurde zunächst ein „ReadPosition.ino“ Programm geschrieben.

Codeinhalt:

```
// Pins for Motor Position Feedback
int Mot1PosPin = 0;
int Mot2PosPin = 1;
int Mot3PosPin = 2;
int Mot4PosPin = 3;
int Mot5PosPin = 4;

// Init Values for Motor Position
int Mot1Pos = 0;
int Mot2Pos = 0;
int Mot3Pos = 0;
int Mot4Pos = 0;
int Mot5Pos = 0;

// Init serial output
void setup() {
  Serial.begin(9600);
  digitalWrite(8, HIGH);
}

// Program cycle
void loop()
{
  // Read the value of the Potis
  Mot1Pos=analogRead(Mot1PosPin);
  Mot2Pos=analogRead(Mot2PosPin);
  Mot3Pos=analogRead(Mot3PosPin);
  Mot4Pos=analogRead(Mot4PosPin);
  Mot5Pos=analogRead(Mot5PosPin);

  // Show Values
  Serial.print("Motor 1: ");
  Serial.println (Mot1Pos);
  Serial.print("Motor 2: ");
  Serial.println (Mot2Pos);
  Serial.print("Motor 3: ");
  Serial.println (Mot3Pos);
  Serial.print("Motor 4: ");
  Serial.println (Mot4Pos);
  Serial.print("Motor 5: ");
  Serial.println (Mot5Pos);
  Serial.println ();

  // Wait time
  delay(2000);
}
```

Dieses Programm wurde nun auf das Arduino-Board hochgeladen und anschließend gestartet. Als Ergebnis werden auf dem seriellen Monitor die Positionen zyklisch ausgegeben.

Nun kann man mit dem Kontrollgerät per Hand alle Positionen anfahren, die man im späteren Verlauf für das Automatisierungsprogramm benötigt und die „Koordinaten“, also die Potentiometerwerte, notieren.



The screenshot shows the Arduino IDE with the 'ReadPosition.ino' file open. The code is identical to the one provided in the previous block. The serial monitor on the right displays the output of the program, showing the position values for five motors in a repeating cycle. The values are: Motor 4: 474, Motor 5: 430, Motor 1: 58, Motor 2: 503, Motor 3: 413, Motor 4: 474, Motor 5: 430, Motor 1: 58, Motor 2: 503, Motor 3: 400, Motor 4: 475, Motor 5: 430, Motor 1: 58, Motor 2: 503, Motor 3: 372, Motor 4: 476, Motor 5: 430, Motor 1: 58, Motor 2: 503, Motor 3: 413, Motor 4: 474, Motor 5: 430.

Abbildung 3-7 - Positionen auslesen

3.4 Motoransteuerung

3.4.1 Hardwareumsetzung

Ein weiterer wichtiger Punkt in dem Projekt „Sockensortieranlage“ war die Ansteuerung der Motoren mit dem „Schweden“.

Das Hauptproblem war, dass man die fünf Motoren des Roboterarms nicht direkt über die digitalen Anschlüsse ansteuern kann. Der Hintergrund dafür ist, dass die Anschlüsse D0 bis D13 nur eine maximale Strombelastung von 40 mA standhalten. Jedoch beträgt der Nennstrom pro Motor ca. 200 mA. Der Anlaufstrom ist mit ca. 1,8 A wiederum um ein vielfaches höher. Zudem müssen die Motoren mit einer Spannung von 3 V bis 5 V angesteuert werden.

Aufgrund dieser Erkenntnisse mussten wir, das Projektteam, eine Möglichkeit zur Ansteuerung der Motoren finden. Zum einen müssen die Motoren über einen leistungsstarken Anschluss angesteuert werden und zum anderen müssen die Motoren in zwei Richtungen betriebsfähig sein.

Es gibt mehrere Schaltungen, die sowohl mit den benötigten Strömen, als auch in den gewünschten Spannungsbereichen belastbar sind.

Wie man in der Vorlesung „Elektrische Bauelemente und Schaltungen“ vermittelt bekommt, können Verstärkerschaltungen mit Transistoren oder nicht invertierende Verstärkerschaltungen mit Operationsverstärkern zwar leicht die nötige Spannung aufbringen aber um den benötigten Strom zu treiben, ist ein sehr aufwendiger elektronischer Aufbau erforderlich.

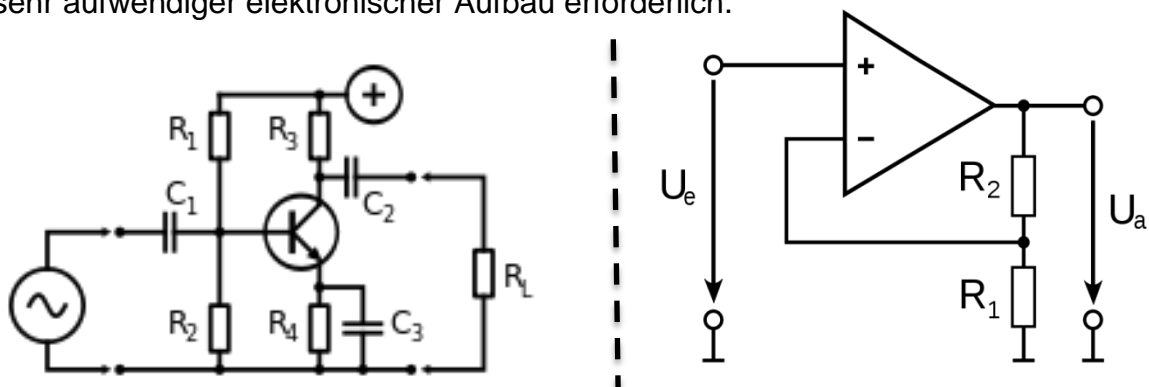


Abbildung 3-8 - Verstärkerschaltungen

Jedoch mussten wir feststellen, dass dieser Aufwand für die Entwicklung und das Löten der Schaltung, welche zur Ansteuerung von fünf Motoren mit zwei

Drehrichtungen zu zeitaufwendig ist. Auf die Funktion der Transistor- und Operationsverstärkerschaltung wird nicht genauer eingegangen.

Um die Motoren in zwei Richtungen betreiben zu können gibt es die sogenannte „H-Brückenschaltung“. Diese ermöglicht es, mit einer Gleichspannungsquelle Motoren in zwei Richtungen zu betreiben.

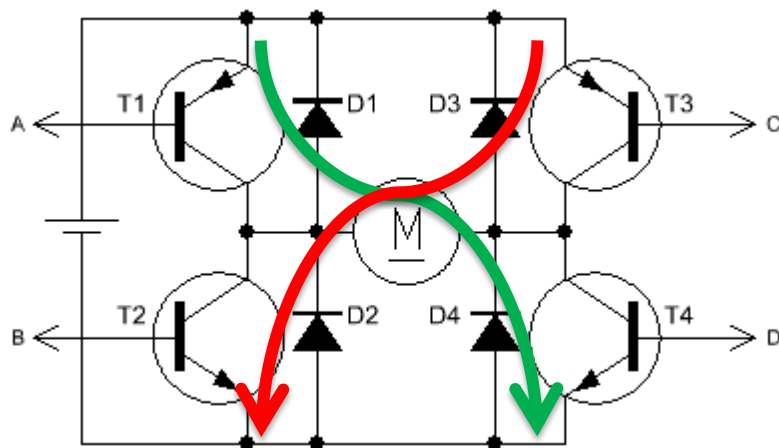


Abbildung 3-9 - H-Brücke

Durch gezielte Ansteuerung, wie in der Abbildung zu sehen, kann über die vier Transistoren der Stromfluss in der Schaltung gesteuert werden. Damit der Motor M in eine Richtung dreht, müssen über die Basen T1 und T4 Strom fließen. Wenn der Motor in die andere Richtung angesteuert werden soll, müssen die Basen T2 und T3 durchgeschaltet werden. Dies ist über die Pinansteuerung des „Schweden“ realisiert.

Wir recherchierten auf der Webseite www.adafruit.com. Hier stießen wir auf den „Motor/Stepper/Servo Shield for Arduino“. Dieser würde in den Themen Strom und Spannung perfekt zu unseren Anforderungen passen. Jedoch kann man mit diesem Shield nur zwei Motoren ansteuern. Daraufhin haben wir den Motor Shield und dessen Bauteile genauer betrachtet.

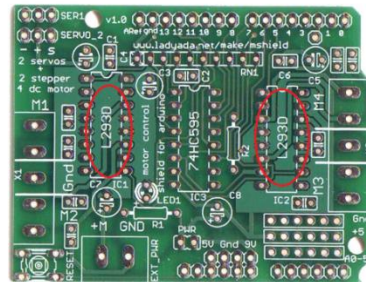


Abbildung 3-10 - Motor Shield Rückseite

Bei der Betrachtung der Rückseite des Motor Shields ist uns der Baustein „L293D“ aufgefallen und wir haben nach dessen Funktion in den dafür zugehörigem Datenblatt recherchiert. Dieser fertige IC wurde für die Motoransteuerung verwendet. Anschließend möchten wir auf die einzelnen Funktionen des Treiberbausteins eingehen.

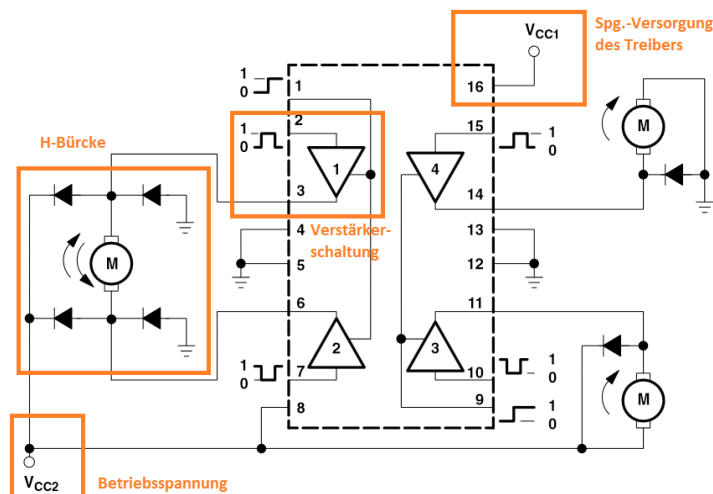


Abbildung 3-11 - L293D Aufbau

Der Treiberbaustein enthält alle Funktionen die für die Ansteuerung der Roboterarmmotoren über das Arduino-Board notwendig sind. Zu den Funktionen gehören:

- Eine stromlose Ansteuerung über die Verstärkerschaltung
- Motorbetrieb in zwei Drehrichtungen (H-Brücke)
- Antrieb der Motoren mit der Betriebsspannung V_{CC2}

Es muss jedoch beachtet werden, dass V_{CC1} und V_{CC2} mindestens 4,5V betragen musste. Zudem kann der L293D-Baustein Spitzenströme von bis zu 2A und Dauerströme von bis zu 600mA aushalten. Das sind genau die Strom und Spannungsbereiche, für die die Motoren des Roboterarms ausgerichtet sind.

Bevor die Steuerplatine gelötet wurde, haben wir noch einen Testlauf mit dem Baugruppentreiber durchgeführt.

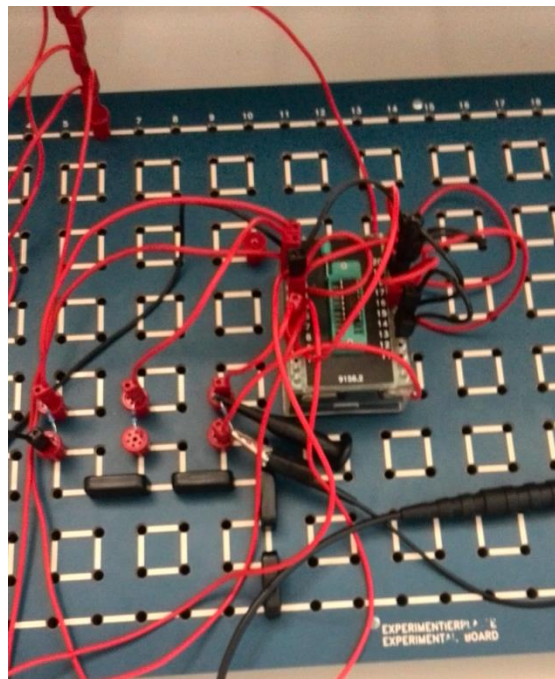


Abbildung 3-12 - L293D Test

Der Test im Labor zeigte das wir den L293D-Baustein für die Ansteuerung der Roboterarmmotoren verwenden konnten.

3.4.2 Schaltungsaufbau

Die Versorgung der Platine wurde über einen NOTAUS-Schalter, durch zwei parallel geschaltete Netzteile (welche dafür geeignet sind) umgesetzt. Die Einspeisung erfolgt mit 9V und maximal 2A Ausgangsstrom. Der Spannungsabfall an den ICs betrug bis zu 6V. An den Pins für die Motoransteuerung, liegen somit genau die für Betrieb der Motoren erforderlichen 3V Spannung an.

Für die Ansteuerung der fünf Achsen (Motoren) wurden drei L293D Bausteine (siehe Bild) benötigt und auf einer Platine angebracht. Sockel, in welche die Bausteine gesteckt werden, sorgen für eine erleichterte Austauschbarkeit (Wartung, Defekt).



Abbildung 3-13 - Platinenoberseite

Durch zeitintensives Testen der zusammenspielenden Motoren, kam es zu starken Erwärmungen der Bausteine und der Anschlussleitungen. Es war notwendig eine Kühlung für die Bausteine zu entwickeln. Hierfür wurde eine abnehmbare Konstruktion direkt über der Platine installiert und die ICs mit Kühlkörpern versehen. Der Lüfter wird mit umlegen des Notaus-Schalter eingeschaltet.

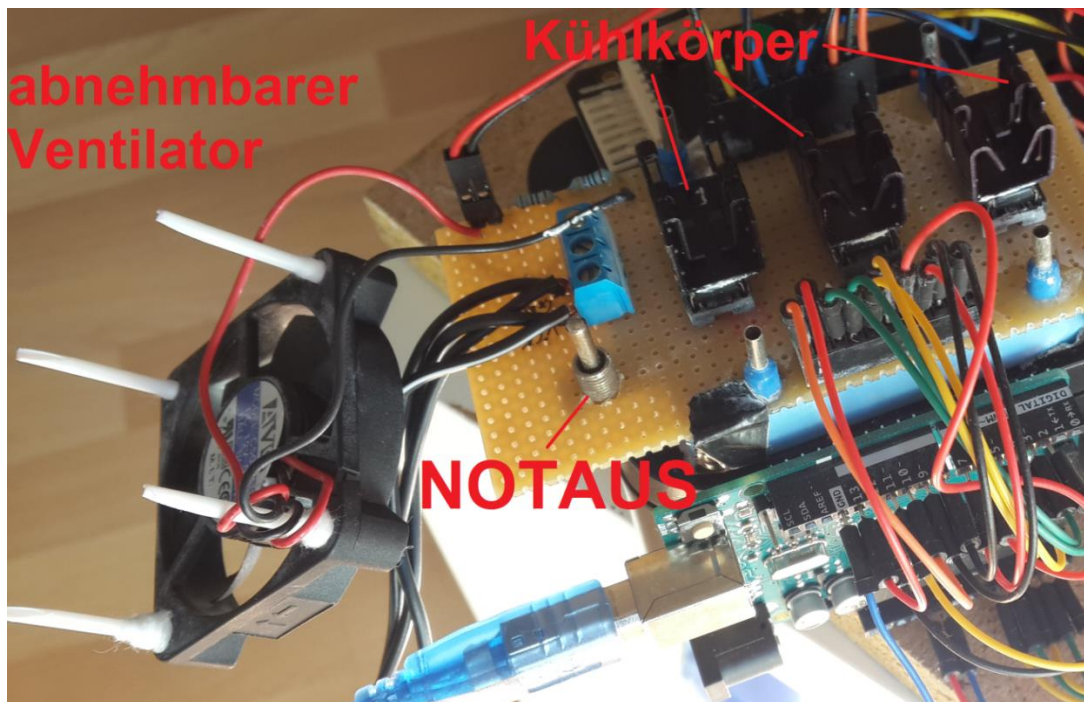


Abbildung 3-14 – Platinenoberseite ohne Ventilator

Zudem wurden die Leitungen in großem Abstand zueinander verlegt, um die thermische Entwicklung bzw. das Schmelzen der dünnen Isolationsschicht der Leitungen vorzubeugen.

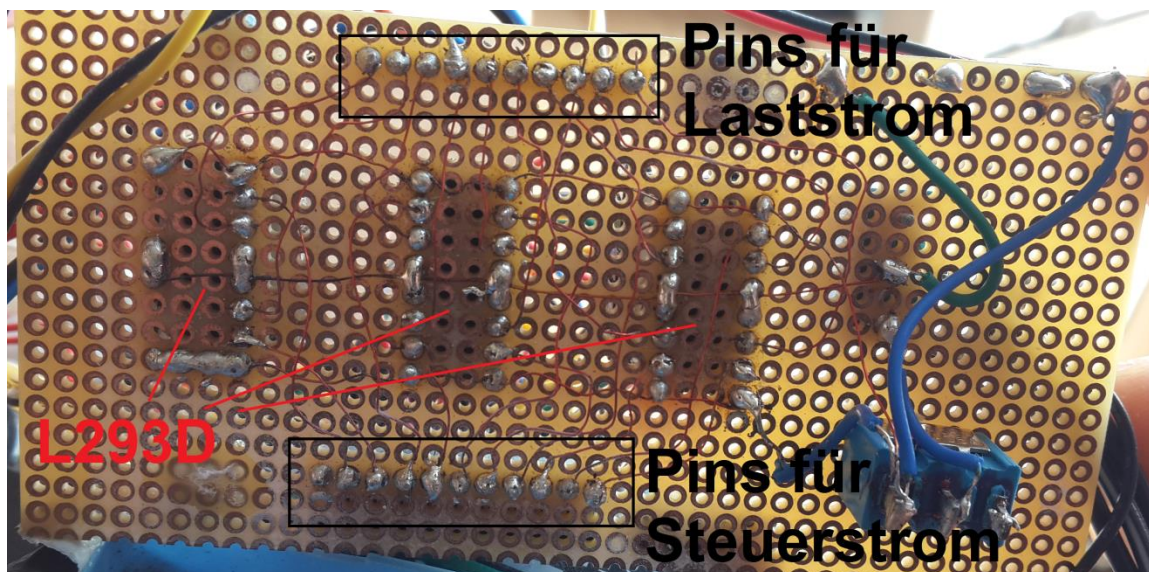


Abbildung 3-15 - Platinenunterseite

Eine Box an der Unterseite der Platine schützt die Leitungen und deren Anordnung zu- und gegeneinander (Abstand halten).

3.4.3 Softwareumsetzung

Da der vollständige Quellcode, inklusive Kommentaren im Anhang zu finden ist, wird hier nur auf die grundlegende Idee eingegangen.

Grundsätzlich: Der Motor hat eine zweiadrige Leitung. Die Drehrichtungsumkehrung erfolgt über die Änderung der Ankerspannung, welche wiederum über die Pegel zweier Pins umgesetzt wird.

Um die Motoren anzusteuern, wurde eine `driveto` – Funktion geschrieben, die immer wieder mit neuen Werten, also wohin der „Deutsche“ als nächstes fahren soll, aufgerufen wird. Dabei wird verglichen, ob der aktuelle Potentiometer Wert dem übergebenden Wert entspricht. Ist dies nicht der Fall, wird der entsprechende Motor solange betrieben, bis der vorgegebene Potentiometer Wert erreicht ist. Die Motoren müssen dabei über den L293D-Baustein so angesteuert werden, dass ein Pegel HIGH und der andere Pegel LOW ist. Ist der Potentiometer Wert der Achse erreicht, müssen die Pegel auf LOW zurückgesetzt werden, da der Motor ansonsten einfach weiter laufen würde. Um ein dauerhaftes Nachstellen der Motoren zu vermeiden, wurde eine Hilfsvariable definiert. So wird der Motor nach erstmaligem Erreichen des geforderten Potentiometer Wertes angehalten und auch nicht mehr angesteuert, selbst wenn andere Motoren noch laufen und diesen Potentiometer Wert beeinflussen.

Dies ist nur eine kurze einfache Beschreibung eines kleinen Teils des Motorencodes, welcher zum Grundverständnis beitragen soll. Der Code durchlief etliche Testphasen und wurde für eine reibungslose Bewegung des Arms immer wieder angepasst.

```
if (Position[1] != 1)
{
    if (Position[1] > Mot2Pos)
    {
        digitalWrite(Motor2rechts, HIGH);
        digitalWrite(Motor2links, LOW);
    }
    else if (Position[1] < Mot2Pos)
    {
        digitalWrite(Motor2links, HIGH);
        digitalWrite(Motor2rechts, LOW);
    }
    else if (Position[1] == Mot2Pos)
    {
        digitalWrite(Motor2links, LOW);
        digitalWrite(Motor2rechts, LOW);
        Position[1] = 1;
    }
    Mot2Pos = analogRead(Mot2PosPin);
    Serial.print("Motor 2: ");
    Serial.println (Mot2Pos);
}
```

Abbildung 3-16 - Teilausschnitt des Motorcodes

3.5 RFID

RFID (engl. *radio-frequency identification* also eine „Identifizierung mit Hilfe elektromagnetischer Wellen“) bezeichnet eine Technologie für Sender-Empfänger-Systeme zum automatischen und berührungslosen Identifizieren und Lokalisieren von Objekten und Lebewesen mit Radiowellen.

Ein RFID-System besteht aus einem Transponder (umgangssprachlich auch *Funketikett* genannt), der sich am oder im Gegenstand bzw. Lebewesen befindet und einen kennzeichnenden Code enthält, sowie einem *Lesegerät* zum Auslesen dieser Kennung.²

3.5.1 RFID Reader/Writer

Die Einarbeitung in den RFID Thematik bzw. die Funktionsweise wurde ganz am Anfang des Projektes umgesetzt, da dies das unbekannteste Gebiet für uns war. Aufgrund der zahlreichen Tutorials im Internet und der auf der Arduino Homepage, wurde uns diese Einarbeitung doch erleichtert. Von der Hochschule wurde uns ein Reader/Writer Mifare RC522, welcher kompatibel mit Arduino-Boards ist, gestellt. Dieser arbeitet im HF Bereich (high frequency) mit einer Frequenz von 13,56 MHz, welche auch die Transponder aufweisen mussten.

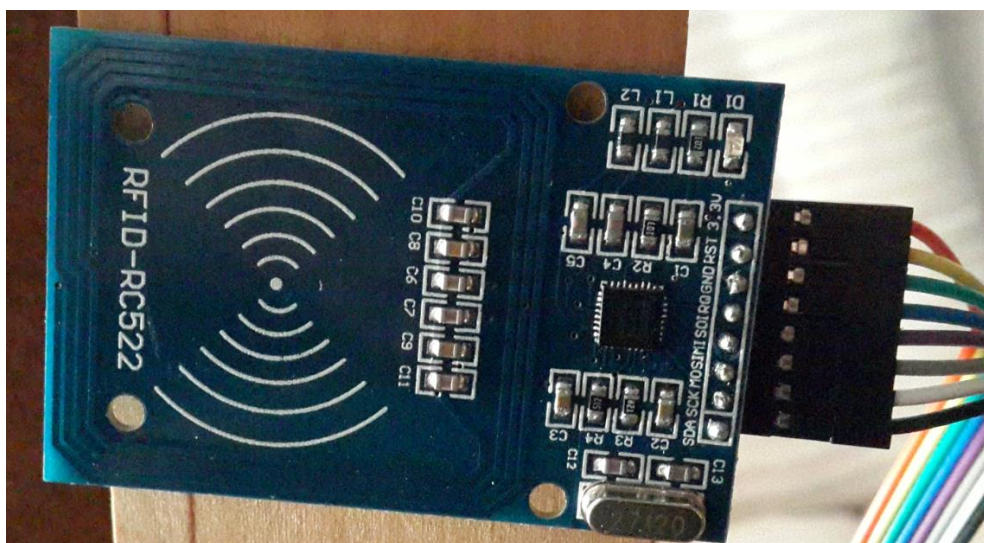


Abbildung 3-17 - RFID Reader/Writer

² Vgl. <https://de.wikipedia.org/wiki/RFID> Stand:07.07.2015

3.5.2 RFID Tags

Unsere Tags sollten sowohl klein, als auch waschbar sein, damit die Tags beim Tragen der Socken nicht stören und natürlich in der Waschmaschine nicht kaputt gehen. Im Handel existieren solche Tags auch. Sie sind jedoch meist teurer als nicht waschbare Tags und werden vor allem von ausländischen Firmen vertrieben. In diesem Zusammenhang haben wir auch mehrere deutsche Firmen, die sich auf RFID spezialisiert haben, angeschrieben und nachgefragt, ob sie uns derartige Tags zukommen lassen könnten. Leider waren die Tags entweder zu groß, nicht waschbar oder hatten eine andere Frequenz.

Da aber nur die Funktionsweise unseres Sockensortierers dargestellt werden sollte, reichte es für uns aus auf nichtwaschbare RFID Tags zurückzugreifen.



Abbildung 3-18 - Socke mit kleinem Transponder

Nach einigen Tests wurde uns jedoch schnell klar, dass es extrem schwierig ist die kleinen Tags genau über die Leseeinrichtung des Mifare Readers zu positionieren, weshalb wir schlussendlich die RFID Transponder Karten des Typs NXP Mifare Classic 1k 13,56 MHz benutzt haben, welche doch um einiges größer sind und dadurch leichter und schneller vom Reader erkannt und wahrgenommen werden.

Ein Nachteil, der jedoch dadurch entstand, war, dass die Socken einiges an Gewicht dazubekommen haben und das Zugreifen bzw. das Halten der Socken mit dem Greifer erschwert wurde.



Abbildung 3-19 - Socke mit großem Transponder

Funktionsweise der Transponder:

Es musste der HEXDUMP bis auf den Kern verstanden werden, da ansonsten, dass einlesen und beschreiben der Transponder nicht umgesetzt werden konnte.

COM9 (Arduino Uno)

Scan a MIFARE Classic card
card selected
2 is a data block:
block was written
Card UID: 70 C2 51 A9
PICC type: MIFARE 1KB

Sector	Block	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	AccessBits
15	63	00	00	00	00	00	00	FF	07	80	69	FF	FF	FF	FF	FF	FF	[0 0 1]
	62	32	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	[0 0 0]
	61	33	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	[0 0 0]
	60	4E	69	65	6C	73	20	54	68	69	65	6C	65	00	00	00	00	[0 0 0]
14	59	00	00	00	00	00	00	FF	07	80	69	FF	FF	FF	FF	FF	FF	[0 0 1]
	58	57	65	69	DF	00	00	00	00	00	00	00	00	00	00	00	00	[0 0 0]
	57	33	30	2E	36	2E	32	30	31	35	00	00	00	00	00	00	00	[0 0 0]
	56	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	[0 0 0]
13	55	00	00	00	00	00	00	FF	07	80	69	FF	FF	FF	FF	FF	FF	[0 0 1]
	54	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	[0 0 0]
	53	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	[0 0 0]
	52	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	[0 0 0]
12	51	00	00	00	00	00	00	FF	07	80	69	FF	FF	FF	FF	FF	FF	[0 0 1]
	50	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	[0 0 0]
	49	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	[0 0 0]
	48	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	[0 0 0]
11	47	00	00	00	00	00	00	FF	07	80	69	FF	FF	FF	FF	FF	FF	[0 0 1]
	46	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	[0 0 0]
	45	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	[0 0 0]
	44	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	[0 0 0]
10	43	00	00	00	00	00	00	FF	07	80	69	FF	FF	FF	FF	FF	FF	[0 0 1]
	42	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	[0 0 0]
	41	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	[0 0 0]
	40	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	[0 0 0]
9	39	00	00	00	00	00	00	FF	07	80	69	FF	FF	FF	FF	FF	FF	[0 0 1]
	38	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	[0 0 0]
	37	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	[0 0 0]
	36	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	[0 0 0]

Abbildung 3-20 - HEXDUMP Teil1

8	35	00 00 00 00	00 00 FF 07	80 69 FF FF	FF FF FF FF	[0 0 1]
	34	00 00 00 00	00 00 00 00	00 00 00 00	00 00 00 00	[0 0 0]
	33	00 00 00 00	00 00 00 00	00 00 00 00	00 00 00 00	[0 0 0]
	32	00 00 00 00	00 00 00 00	00 00 00 00	00 00 00 00	[0 0 0]
7	31	00 00 00 00	00 00 FF 07	80 69 FF FF	FF FF FF FF	[0 0 1]
	30	00 00 00 00	00 00 00 00	00 00 00 00	00 00 00 00	[0 0 0]
	29	00 00 00 00	00 00 00 00	00 00 00 00	00 00 00 00	[0 0 0]
	28	00 00 00 00	00 00 00 00	00 00 00 00	00 00 00 00	[0 0 0]
6	27	00 00 00 00	00 00 FF 07	80 69 FF FF	FF FF FF FF	[0 0 1]
	26	00 00 00 00	00 00 00 00	00 00 00 00	00 00 00 00	[0 0 0]
	25	00 00 00 00	00 00 00 00	00 00 00 00	00 00 00 00	[0 0 0]
	24	00 00 00 00	00 00 00 00	00 00 00 00	00 00 00 00	[0 0 0]
5	23	00 00 00 00	00 00 FF 07	80 69 FF FF	FF FF FF FF	[0 0 1]
	22	00 00 00 00	00 00 00 00	00 00 00 00	00 00 00 00	[0 0 0]
	21	00 00 00 00	00 00 00 00	00 00 00 00	00 00 00 00	[0 0 0]
	20	00 00 00 00	00 00 00 00	00 00 00 00	00 00 00 00	[0 0 0]
4	19	00 00 00 00	00 00 FF 07	80 69 FF FF	FF FF FF FF	[0 0 1]
	18	00 00 00 00	00 00 00 00	00 00 00 00	00 00 00 00	[0 0 0]
	17	00 00 00 00	00 00 00 00	00 00 00 00	00 00 00 00	[0 0 0]
	16	00 00 00 00	00 00 00 00	00 00 00 00	00 00 00 00	[0 0 0]
3	15	00 00 00 00	00 00 FF 07	80 69 FF FF	FF FF FF FF	[0 0 1]
	14	00 00 00 00	00 00 00 00	00 00 00 00	00 00 00 00	[0 0 0]
	13	00 00 00 00	00 00 00 00	00 00 00 00	00 00 00 00	[0 0 0]
	12	00 00 00 00	00 00 00 00	00 00 00 00	00 00 00 00	[0 0 0]
2	11	00 00 00 00	00 00 FF 07	80 69 FF FF	FF FF FF FF	[0 0 1]
	10	00 00 00 00	00 00 00 00	00 00 00 00	00 00 00 00	[0 0 0]
	9	00 00 00 00	00 00 00 00	00 00 00 00	00 00 00 00	[0 0 0]
	8	00 00 00 00	00 00 00 00	00 00 00 00	00 00 00 00	[0 0 0]
1	7	00 00 00 00	00 00 FF 07	80 69 FF FF	FF FF FF FF	[0 0 1]
	6	00 00 00 00	00 00 00 00	00 00 00 00	00 00 00 00	[0 0 0]
	5	00 00 00 00	00 00 00 00	00 00 00 00	00 00 00 00	[0 0 0]
	4	00 00 00 00	00 00 00 00	00 00 00 00	00 00 00 00	[0 0 0]
0	3	00 00 00 00	00 00 FF 07	80 69 FF FF	FF FF FF FF	[0 0 1]
	2	6D 61 6B 65	63 6F 75 72	73 65 5F 5F	5F 5F 5F 00	[0 0 0]
	1	00 00 00 00	00 00 00 00	00 00 00 00	00 00 00 00	[0 0 0]
	0	70 C2 51 A9	4A 08 04 00	62 63 64 65	66 67 68 69	[0 0 0]

Abbildung 3-21 - HEXDUMP Teil2

Bei dem HEXDUMP wurden als erstes kartenspezifische Informationen, wie die UID und der Transpondertyp, um den es sich handelt, ausgegeben. Bei letzterem handelt es sich um den Tag „MIFARE 1KB“. Hierfür ist es notwendig das richtige Protokoll zu verwenden. Dabei ist unter 1KB, 1024 Byte zu verstehen und beschreibt die Datenmenge des Transponders. Aufgebaut ist dieser in 16 Sektoren, wobei jeder Sektor 4 Zeilen beinhaltet. Die Sektoren beinhalten Schlüsselnummern, welche den Lese-/Schreibzyklus verwalten. Jede/r vierte Block/Zeile enthält die Zugangsberechtigung bzw. Sicherheitsinformationen für den jeweiligen Sektor. Jede Zeile besteht aus 16 Byte, die beschrieben werden können, ausgenommen den Berechtigungszeilen. Die Informationsdarstellung erfolgt in HEX Schreibweise auf dem „Tag“.

Wir benutzen folgende Blöcke für Dateninformationen:

62,61,60,58,57 → Maximale Speicherkapazität des Transponders wurde bei weitem nicht erreicht.

Mehr Informationen zum Beschreiben und Lesen der Tags, siehe Code im Anhang.

3.5.3 RFID-Tags Auswertung

Anschließend ist die serielle Monitoranzeige des RFID Reader/Writer während des regulären Betriebs zu sehen. Dieser Auszug des Transponders zeigt die in ihm gespeicherten Daten/Informationen in lesbarem Zeichensatz.

Weiterhin ist zu erkennen, dass bei erneutem Beschreiben bzw. Sortieren eines Sockens, die Waschgänge inkrementiert werden.

```
COM9 (Arduino Uno)

Sockendaten werden erfasst
Sockenpaarnummer: 1
Waschgaenge: 51
Der Besitzer der Socke ist: Damian Uczulko
Die Farbe des Socken ist: Orange
Das Kaufdatum der Socke war der: 30.6.2015

Sockendaten werden erfasst
Sockenpaarnummer: 2
Waschgaenge: 51
Der Besitzer der Socke ist: Niels Thiele
Die Farbe des Socken ist: Weiß
Das Kaufdatum der Socke war der: 30.6.2015

Sockendaten werden erfasst
Sockenpaarnummer: 3
Waschgaenge: 54
Der Besitzer der Socke ist: Sebastian Zeh
Die Farbe des Socken ist: Blau
Das Kaufdatum der Socke war der: 30.6.2015

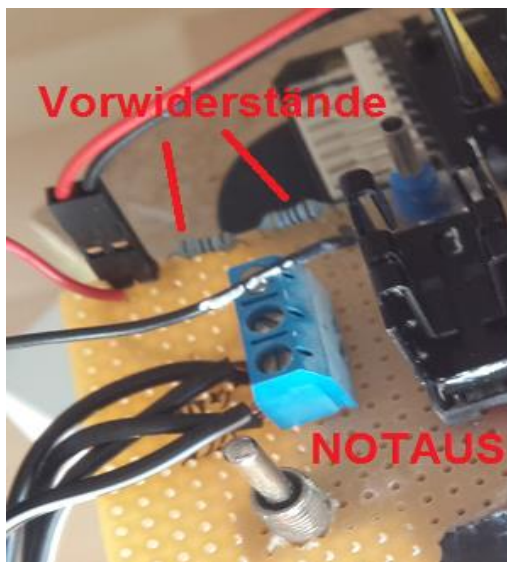
Sockendaten werden erfasst
Sockenpaarnummer: 2
Waschgaenge: 52
Der Besitzer der Socke ist: Niels Thiele
Die Farbe des Socken ist: Weiß
Das Kaufdatum der Socke war der: 30.6.2015
```

Abbildung 3-22 - Auswertung der RFID Transponder während dem Betrieb

3.6 Zusätzliche Anbauten / Probleme

3.6.1 Betriebsbereitschaft / NOTAUS – Funktion

Um die Anlage, also die Motoren/Getriebe und Aufbauten vor Zerstörung zu schützen, realisierten wir einen hardwaretechnischen Notausschalter in Reihe mit einem Leiter der Spannungseinspeisung. Dieser Schalter war beim Testen der Anlage von enormer Bedeutung, da der Arm des Öfteren über seine mechanische Begrenzung hinaus gefahren ist. → Abschaltung der Anlage!



$$I_{max} = \frac{U}{R} = \frac{9V}{20mA} = 450\Omega$$

→ Summe 500Ω mit je 0,1W

Abbildung 3-23 - Vorwiderstände LED & NOTAUS

Die Betriebsbereitschaft der Anlage wurde durch eine rote LED realisiert. Unsere Anlage ist mit 9V DC gespeist (siehe [3.4.2 Schaltungsaufbau](#)). Der maximale Strom durch die LED musste durch Vorwiderstände begrenzt werden.

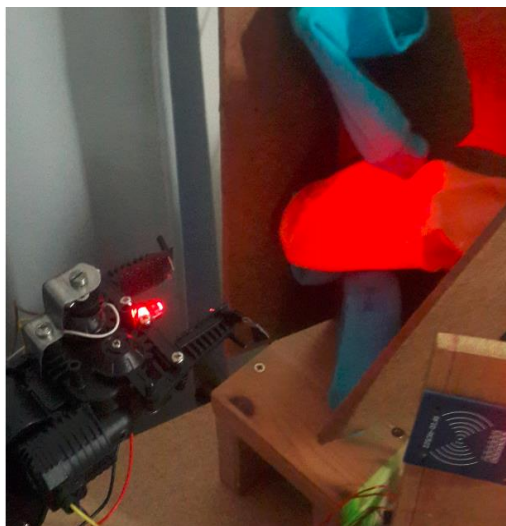


Abbildung 3-24 - LED Betriebsbereitschaft

3.6.2 Greifarm-Verbesserungen (Klett und Nadeln)

Schon nach den ersten Tests war ersichtlich, dass der Greifer ohne zusätzliche Hilfsmittel die Socken nur in den seltensten Fällen richtig aufnehmen kann. Um dies zu verbessern, brachten wir Klettstreifen im inneren der Zange an. Zusätzliche spitze Nadeln sorgten für das bestmögliche Endergebnis. Die ideale Aufnahme stand jedoch im direkten Zielkonflikt mit der Ablage der Socken. Diese mussten wir also Softwaretechnisch durch eine Bewegungsroutine des „Deutschen“ lösen (siehe [Funktionsbeschreibung](#)).



Abbildung 3-25 - Greifer mit Klett und Nadeln

3.7 Kommunikation der Arduino-Boards

Wie bereits schon beschrieben, waren die PINs von einem Arduino-Board nicht ausreichend, um sowohl mit den Arm, als auch den RFID-Reader/Writer zu kommunizieren, weshalb wir auf zwei Boards zurückgreifen mussten. Nun standen zwei serielle Monitore zur Verfügung, welche die Steuerung und die Überwachung der Anlage vereinfachten. Um die Pegelsteuerung wie oben beschrieben umzusetzen, mussten noch einige Probleme gelöst werden. Darunter das Hauptproblem der Massenverschiebung der Arduino-Boards bei Anschluss an zwei Laptops und das digitale Einlesen eines HIGH-Pegels mit Pulldown Widerständen.

1. Problem der Massenverschiebung

Nach ersten Tests, bei denen wir trotz HIGH-Pegel am „Japaner“ keinen HIGH-Pegel am „Schweden“ empfangen konnten, haben wir mit einem Multimeter den

Aufbau vermessen und sind dabei auf eine Potentialdifferenz von bis zu 8V gestoßen.

Lösung: Um dieses Problem zu beheben, musste die Spannungsversorgung eines Laptops für beide Arduino-Boards verwendet werden. Dies wurde durch zusätzliche Spannungsleitungen vom „Schweden“ zum „Japaner“ realisiert.

Wichtig hierbei: Die korrekte Spannungsversorgung eines Arduino-Boards wird laut Spezifikation intern erkannt und automatisch umgeschaltet. Dies bedeutet, dass der Arduino trotz Anschluss am Laptop (USB-Schnittstelle), nicht von dieser mit Spannung versorgt wird, sondern der externe Eingang +Vcc und Ground (siehe Stromlaufplan und Skizze) bevorzugt zur Einspeisung genutzt wird.

2. Problem des Einlesens eines HIGH-Pegels mit Pulldown Widerständen

Sobald die Pins des Arduino Boards als Eingänge belegt werden, wird intern ein Widerstand auf die Pins geschaltet bzw. werden die Pins mit einem Spannungspegel angesteuert. Um aber einen definierten HIGH Eingangspegel des „Japaners“ einzulesen, mussten die Eingänge des „Schwedens“ auf einen definierten Massepegel gezogen werden.

Lösung: Pulldown - Widerstände mit über 30 k Ω

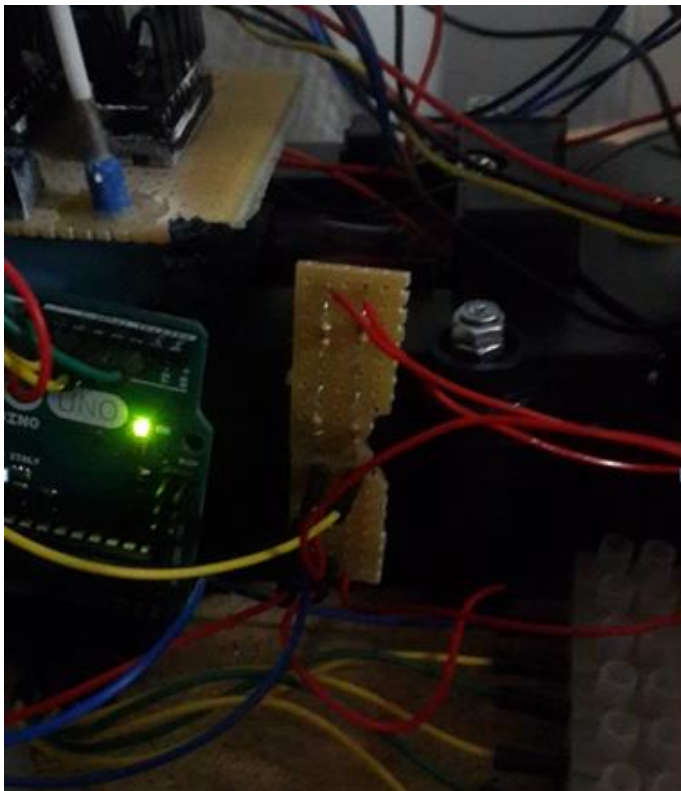


Abbildung 3-26 - Pulldown Widerstände

4 Schlusswort

Die Möglichkeit aus einer Idee ein Konzept bzw. ein komplettes fertiges Projekt zu realisieren, ist uns gelungen! Dazu gehörten insbesondere die sinnvolle Aufteilung von Verantwortungsbereichen und das gemeinsame Finden von Problemlösungen.

Das Wahlfach „Physical Computing“ hat zu dem das Interesse an weiteren Projekten, die man auch privat mit relativ geringem finanziellem Aufwand durchführen kann, geweckt.

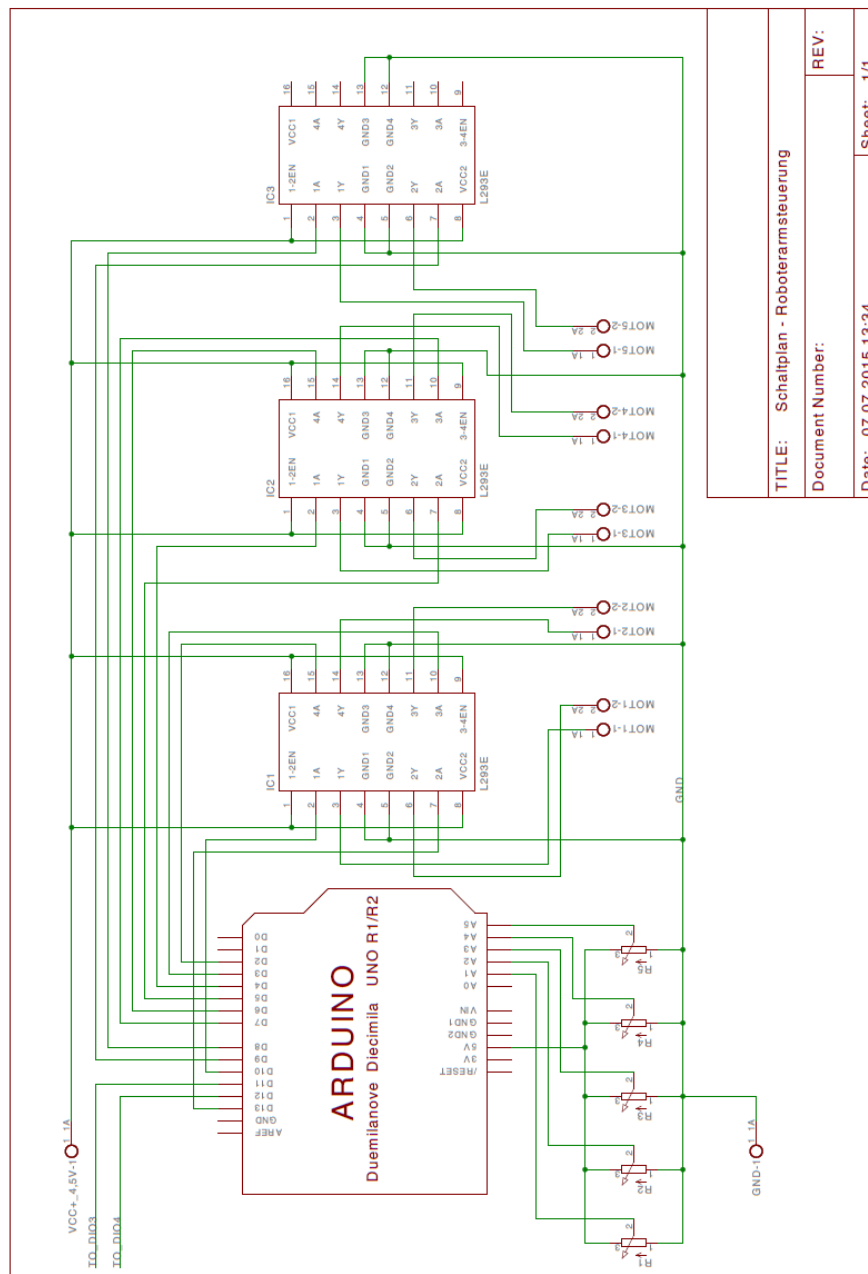
Auch wenn uns das Projekt oftmals viel Zeit und Geduld gekostet hat, so haben wir daraus viel gelernt. Die Teamfähigkeit wurde geschult, sowie eigene Stärken und Schwächen aufgedeckt. Wir empfehlen solche Projektarbeiten noch mehr ins Studium einzubinden, da die Motivation und der durch die praktischen Arbeiten gekennzeichnete Lerneffekt, für den erfolgreichen Abschluss eines durchgängigen Projektes enorm groß sind!

5 Abbildungsverzeichnis

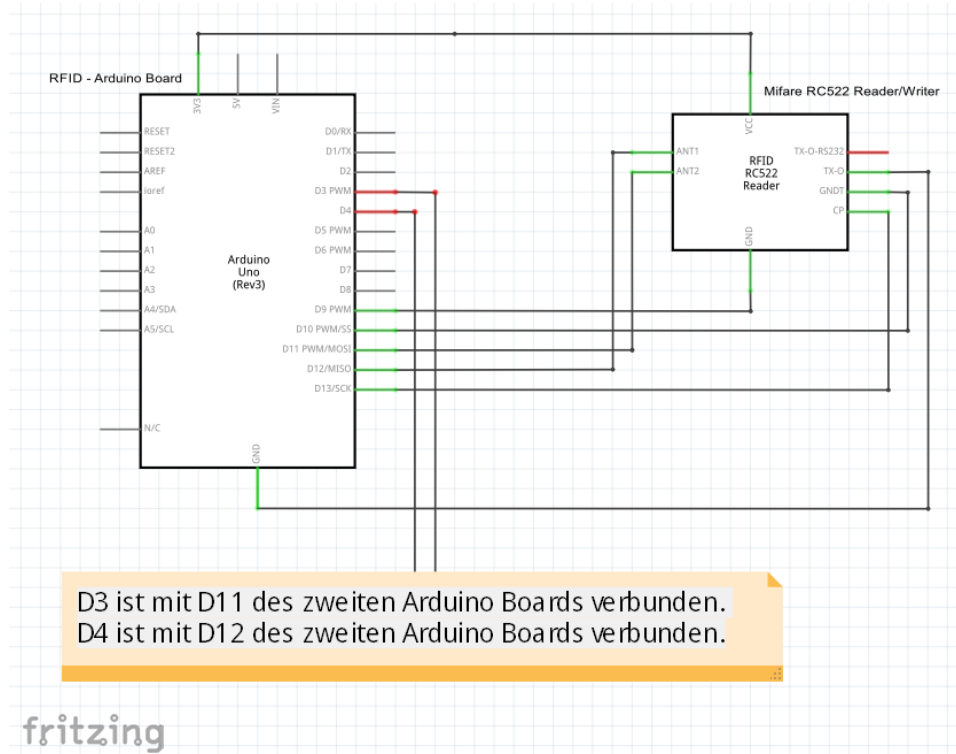
Abbildung 1-1 - Fertige Anlage	3
Abbildung 2-1 - Konzeptskizze	5
Abbildung 2-2 - Arduino Board "Schwede" am Roboterarm	6
Abbildung 2-3 - Arduino Board "Japaner" am RFID Halterung	6
Abbildung 2-4: Konsolenbild der Sockeninitialisierung	7
Abbildung 3-1 - Trichter	9
Abbildung 3-2 - Laternenkonstruktion mit RFID-Reader/Writer	10
Abbildung 3-3 - Wäschekörbe mit sortierten Socken	10
Abbildung 3-4 - Roboterarm KSR10 mit Kontrollgerät	11
Abbildung 3-5 - Roboterarmmontage	11
Abbildung 3-6 - Potentiometermontage	12
Abbildung 3-7 - Positionen auslesen	13
Abbildung 3-8 - Verstärkerschaltungen	14
Abbildung 3-9 - H-Brücke	15
Abbildung 3-10 - Motor Shield Rückseite	16
Abbildung 3-11 - L293D Aufbau	16
Abbildung 3-12 - L293D Test	17
Abbildung 3-13 - Platinenoberseite	18
Abbildung 3-14 – Platinenoberseite ohne Ventilator	19
Abbildung 3-15 - Platinenunterseite	19
Abbildung 3-16 - Teilausschnitt des Motorcodes	20
Abbildung 3-17 - RFID Reader/Writer	21
Abbildung 3-18 - Socke mit kleinem Transponder	22
Abbildung 3-19 - Socke mit großem Transponder	23
Abbildung 3-20 - HEXDUMP Teil1	23
Abbildung 3-21 - HEXDUMP Teil2	24
Abbildung 3-22 - Auswertung der RFID Transponder während dem Betrieb	25
Abbildung 3-23 - Vorwiderstände LED & NOTAUS	26
Abbildung 3-24 - LED Betriebsbereitschaft	26
Abbildung 3-25 - Greifer mit Klett und Nadeln	27
Abbildung 3-26 - Pulldown Widerstände	28

6 Anhang

1. Code: Motoransteuerung → siehe Projektablageordner
2. Code: Initialisierung der Socken → siehe Projektablageordner
3. Code: Socken Auslesen → siehe Projektablageordner
4. Code: ReadPosition → siehe Projektablageordner
5. Schaltplan zur Roboterarmsteuerung (Eagle)



6. Schaltplan für RFID Read/Writer (Fritzing)



Steckplatine:

