# Funktionsweise einer „Automatischen Sortieranlage“

## am Beispiel eines selbstgebauten und programmierten Modelles auf Arduino-Basis

**Arne Ruberg**
Informatik - LK
Betreuender Lehrer: Herr Fassbender

Städtisches Gymnasium Rheinbach
Jahrgangsstufe 11
Schuljahr 2019/2020

## Inhalt


1. Einleitung ..... 3

1.1. Motivation ..... 3
2. Der Sortierroboter ..... 4

2.1. Grundidee/Lösungsansatz ..... 4
2.2. Umsetzung ..... 5
2.2.1. Beschaffung der Bauteile ..... 5
2.2.2. Fahrbahn/Schiene ..... 5
2.2.3. Bewegung auf der Schiene ..... 6
2.2.4. Lichtschranke ..... 8
2.2.5. RFID ..... 8
2.2.6. Greifarm ..... 9
2.2.7. Konsolidierung Fahren und Greifen ..... 11
2.2.8. Sortieren ..... 11
2.3. Ergebnis ..... 12
2.4. Fazit ..... 13
3. Quellen ..... 14

4. Anhänge / zusätzliche Dateien ..... 15

5. Selbstständigkeitserklärung ..... 16


## Abbildungsverzeichnis


Abbildung 1: Innenansicht der Kommissionieranlage ..... 3
Abbildung 2: Außenansicht der Anlage ..... 3
Abbildung 3:Schienensystem ..... 5
Abbildung 4: Motoren des SmartCar mit L298N-Einheit in der Mitte ..... 6
Abbildung 5:L298N-Modul ..... 6
Abbildung 6:Haltepositionen (Orange) je nach Fahrtrichtung (Pfeile) ..... 7
Abbildung 7: Reflexive Lichtschranke ..... 8
Abbildung 8: Prototyp eines Aufsatzes ..... 8
Abbildung 9: Funktionsweise einer Reflexiven Lichtschranke ..... 8
Abbildung 10: 3D-Modell der Halterung ..... 8
Abbildung 11: MFRC522 RFID-Modul mit Transponderchip und -karte ..... 8
Abbildung 12: Halterung am Roboter ..... 9
Abbildung 13: Anschluss an den Arduino ..... 9
Abbildung 14: 16-Channel Servodriver von Sunfounder ..... 9
Abbildung 15: Anordnung der Servos im Arm ..... 9
Abbildung 16: Verschiedene Kartenmodelle ..... 10
Abbildung 17: Prototyp eines Elementes aus Duplo®Steinen ..... 10
Abbildung 18: Armaufsatz für den Roboter ..... 10
Abbildung 19: 2. Prototyp eines Elementes ..... 10

# 1. Einleitung

## 1.1. Motivation

Aufgrund der fortschreitenden Automatisierung und Digitalisierung wird Robotertechnik immer populärer, um Arbeitskräftemängeln und Ineffizienz entgegenzuwirken. In Deutschland verwendet über ein Sechstel der Unternehmer Industrie- und Serviceroboter im Alltag. Dabei kann deren Einsatz ganz unterschiedlich aussehen, die Möglichkeiten reichen von simplen intelligenten Förderbändern bis hin zu Anlagen, die selbstständig Auto für Auto produzieren. Zur Robotertechnik gehören auch Sortieranlagen. Auch diese können in vielen verschiedenen Formen auftreten, unterschiedlich in Größe und Funktionsprinzip. Zum Beispiel werden häufig Sortieranlagen zur Mülltrennung eingesetzt, große und relativ unpräzise Maschinen. Im Gegensatz dazu gibt es hochpräzise Ein- und Auslagerungssysteme, auch Kommissionier-Maschinen genannt.

Auch die Kronenapotheke in Wesseling hat vor kurzem in eine vollautomatische Sortieranlage investiert, um die Kommissionierung von Arzneimitteln in der Krankenhausversorgung zu optimieren. Mit dem System werden über 90.000 Arzneimittel-Packungen vollautomatisch geprüft, eingelagert und für jede Station im Krankenhaus versandfertig zusammengestellt.

*Abbildung 1:*
*Innenansicht der Kommissionieranlage*

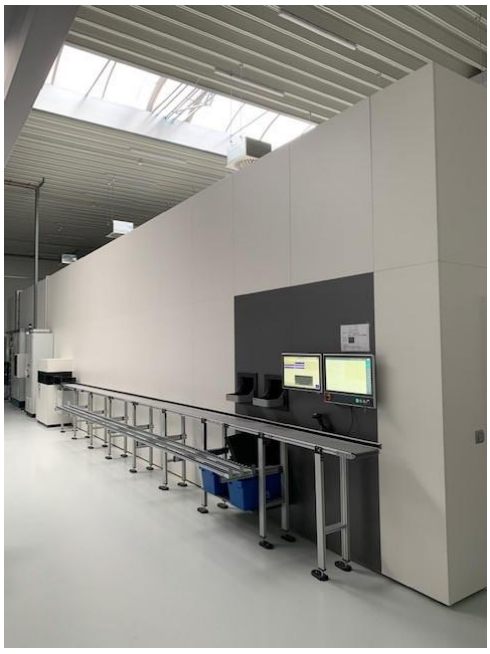

*Abbildung 2: Außenansicht der Anlage*

Diese hocheffiziente Anlage hat mich erst auf die Idee gebracht zu versuchen selbst einen Sortierroboter zu entwickeln.

# 2. Der Sortierroboter

## 2.1. Grundidee/Lösungsansatz

Ein Roboter (Smartcar) soll sich auf Schienen nach links und rechts bewegen und eine Auswahl an zu sortierenden Elementen vor sich haben, die er mit einer noch festzulegenden Methodik erkennen und nach Wert sortieren kann.

Der ausgewählte Roboter, das Eleego Smartcar, fährt mit Hilfe von vier Rädern mit jeweils einem zugehörigen DC-Motor. Diese werden über einen Driver von einem Eleego-Uno angesteuert, welcher in der Mitte des Roboters sitzt. Der Aufbau besteht aus zwei Kunststoffplatten, die übereinander angebracht sind und somit zwei „Stockwerke“ bilden. Das Smartcar wird von einer Batteriebox am hinteren Ende mit Strom versorgt. Der für das Projekt angeschaffte zweite Eleego-Uno sitzt auf der Batteriebox. Der Hersteller des Smartcar stellt ein detailreiches Lernprogramm zur Verfügung. Der Roboter ist bereits mit einigen nützlichen Sensoren und Funktionen ausgestattet, welche sich als durchaus hilfreich erweisen können. Vor allem ein Line-Tracking (Linienverfolgung) Modul, welches aus drei Sensoren besteht, die auf schwarzes Klebeband reagieren. Mit diesen können bis zu 7 verschiedene Bodenmarkierungen vom Roboter erfasst werden. Außerdem sind ein Ultraschallsensor, ein Servomotor, ein Bluetooth-Modul und ein IR-Empfänger (Infrarot für z.B. eine Fernbedienung) vorhanden. Der Ultraschallsensor ist an dem Servomotor in einer Halterung am vorderen Ende des Roboters angebracht. Dieser wird vermutlich entfernt werden, um an dem Servo den Greifarm anbringen zu können. Da der Eleego (Controller-Chip des Smartcars) durch die vorgesehene Benutzung nur sehr wenige freie Ausgänge übrig hat, wird ein Servodriver verwendet, der mit Pulsewidthmanipulation(PWM) bis zu 16 Servos mit nur 4 Ausgängen plus externer Stromzufuhr steuern kann, im Vergleich dazu benötigt ein Servomotor zum Funktionieren drei Anschlüsse: Strom(V+), Ground(GND, Minuspol) und PWM (Steuersignal). Das entspräche dann 48 blockierten Ausgängen, die von dem Arm verwendet würden. Der Servodriver begrenzt diese hohe Anzahl auf vier. Um das Bewegen auf Schienen zu ermöglichen, kann es erforderlich werden, die Räder durch Zahnräder zu ersetzen oder zu modifizieren, damit präzise Bewegung möglich sind.

Das Greifen der Elemente soll mit Hilfe eines simplen 3D-gedruckten oder anderweitig gefertigten Greifarmes erfolgen, der sich nur an einer Achse dreht, sowie mit einem Greifer die Elemente hochhebt und festhält. Um das Zwischenparken eines Objektes zu ermöglichen, welches für einen Tausch notwendig ist, wird es einen „Parkplatz“ geben.

Als Ziel soll der Roboter in der Lage sein, eine beliebige Anzahl Elemente selbstständig einzulesen und mit dem Sortieralgorithmus „Bubble sort“ sortieren zu können.

## 2.2. Umsetzung

### 2.2.1. Beschaffung der Bauteile

Als Erstes mussten die benötigten Bauteile für den Roboter angeschafft werden. Die Materialliste beinhaltete folgende Bauteile:

- 3x40 Kabel
- 5 Lichtschranken
- EeLego®o Smartcar Kit
- EeLego®o Sensor Kit
- Adafruit 16channel Servodriver
- 10 Servos
- Zahnriemen
- 5 RFID Kartenleser
- 1 zusätzlicher EeLego®o R3

Kostenfaktor: ca. 200€

Außerdem wurden zwei Aluminium-Winkel benötigt, die als Schienenträger fungieren und auf ein Brett montiert wurden.

### 2.2.2. Fahrbahn/Schiene

Da der Roboter nicht einfach frei durch den Raum fahren kann, war es nötig ihn durch ein Schienensystem auf eine einzige lineare Bewegung zu beschränken. Die Aluminiumschienen wurden dazu mit Löchern versehen und auf das Holz geschraubt. Hierbei war es wichtig, die Schienen im richtigen Abstand zueinander und vor allem gerade zu befestigen, damit der Roboter sich ohne Probleme bewegen kann. Auch der Abstand zum Rand der Bretter war wichtig, damit genug Platz zum Rangieren der Elemente übrigblieb.

Das Brett ist 1.60m x 30cm breit, die Schienen wurden im Abstand von 16cm voneinander platziert.

*Abbildung 3:Schienensystem*

Erste Fahrversuche im Schienensystem zeigten, dass eine Umrüstung des Smartcars auf Zahnräder gar nicht nötig sein wird, da der Roboter aufgrund der hohen Reibung der Reifen und einstellbarer niedriger Fahrgeschwindigkeit auch so über die nötige Präzision verfügt. Also wurden die Reifen im ursprünglichen Zustand belassen.

### 2.2.3. Bewegung auf der Schiene

Das Smartcar benutzt einen L298N Motordriver, um die vier DC Motoren zu steuern.

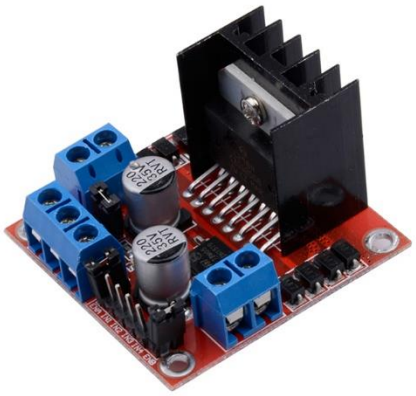

*Abbildung 5:L298N-Modul*

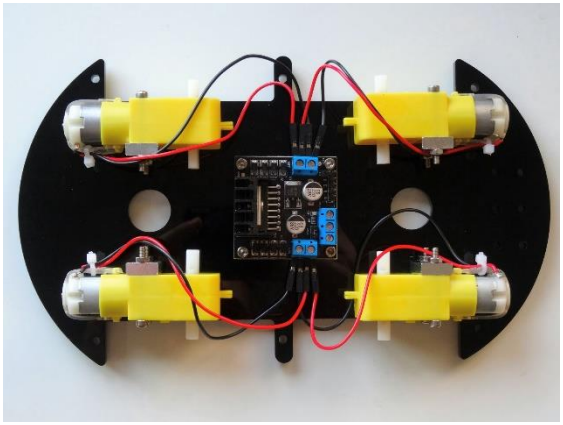

*Abbildung 4: Motoren des SmartCar mit L298N-Einheit in der Mitte*

Das L298N ist dafür verantwortlich, die vier Motoren des Roboters mit Strom zu versorgen. Außerdem verfügt es über eine integrierte Geschwindigkeitssteuerung und ein Kühlelement. Es befindet sich auf der unteren Platte des Smartcars und ist jeweils mit den einzelnen Motoren verbunden. Angesteuert wird es über ein 6-adriges Kabel, welches an den Controller angeschlossen ist und zusätzlich separat Strom für die Motoren erhält. Die Funktionsweise des L298N sieht eine Gruppierung der Motoren in „Links“ und „Rechts“ vor, in diesem Fall also zwei Motoren auf der linken und zwei auf der rechten Seite. Dies erspart unnötig viele Steuersignale, die wiederum Platz am Controller wegnehmen würden. Die 6 Steuersignale für das L298N sind wie folgt bezeichnet: ENA, ENB, IN1, IN2, IN3 und IN4. ENA und ENB sind einstellbare Relaiswiederstände, die die Stromzufuhr und damit die Geschwindigkeit ihrer Seite einstellen. Dabei betreibt ENA die linke Seite und ENB die rechte. IN1-IN4 sind jeweils Richtungsoperatoren für ihre jeweilige Seite (IN1, IN2 links, IN3, IN4 rechts). Folgend entstehen diese Bewegungstabellen:

Linke Seite

| ENA | IN1 | IN2 | Beschreibung |
|---|---|---|---|
| LOW | Irrelevant | Irrelevant | Motor aus |
| HIGH | LOW | LOW | anhalten |
| HIGH | HIGH | LOW | Motor an /vorwärts |
| HIGH | LOW | HIGH | Motor an /rückwärts |
| HIGH | HIGH | HIGH | anhalten |

Rechte Seite

| ENB | IN3 | IN4 | Beschreibung |
|---|---|---|---|
| LOW | Irrelevant | Irrelevant | Motor aus |
| HIGH | LOW | LOW | anhalten |
| HIGH | HIGH | LOW | Motor an /vorwärts |
| HIGH | LOW | HIGH | Motor an /rückwärts |
| HIGH | HIGH | HIGH | anhalten |

Fahrtrichtung

| Linker Motor | Rechter Motor | Beschreibung |
|---|---|---|
| Stopp(aus) | Stopp(aus) | Smartcar angehalten |
| vorwärts | vorwärts | Smartcar fährt vorwärts |
| vorwärts | rückwärts | Smartcar fährt nach rechts |
| rückwärts | vorwärts | Smartcar fährt nach links |
| rückwärts | rückwärts | Smartcar fährt nach rückwärts |

Da das Smartcar für die Problemstellung auf eine lineare Bewegung limitiert wurde, enthält der entwickelte Sourcecode lediglich Methoden zum Vorwärts-, Rückwärtsfahren und zum Anhalten.

Der Controller des L289N wird in diesem Projekt als rein reaktives System verwendet, das heißt, dass er nur Befehle annimmt und ausführt, selbst jedoch keine Evaluation betreibt. Um diese einseitige Beziehung zu ermöglichen, darf das System auch nur aus Elementen bestehen, die ausführen, also keine Sensoren, die eine Rückmeldung an den Hauptcontroller senden müssten. Damit wird der gesamte Handlungsprozess auf einen Controller verlegt, was wesentlich unkomplizierter in der Umsetzung ist. In diesem Fall ist das reaktive System nur für die Fahrbewegungen zuständig, es erhält seine Befehle über serielle Kommunikation mit dem Hauptcontroller. Um dies umzusetzen, werden alle Sensoren von dem Fahrcontroller entfernt und stattdessen an den Hauptcontroller angeschlossen. Der Ultraschallsensor wird wie vorhergesehen nicht benötigt und wird daher komplett aus dem System entfernt. Das Linetracking Modul hingegen wird noch gebraucht und deshalb an den Hauptcontroller angeschlossen. Hierbei ist es wichtig, bei der Verkabelung auf den Anschluss der Abflusswiederstände zu denken, da sich sonst ein Ladungsstau aufbaut und keine Daten eingelesen werden können. Dazu muss ein abzweigender Weg durch einen starken Widerstand (in diesem Fall 10KOhm) auf Ground führen, damit der Strom abfließt, wenn der Sensor „0“ meldet. In diesem Fall würde sich nämlich ohne Widerstandsbrücken ein „Stau“ bilden und der Controller würde weiterhin 1 lesen. Da der Strom aber den Weg mit dem kleinsten Widerstand wählt, wird er den Weg über den Abfluss nur wählen, wenn er keine andere Möglichkeit hat. Somit kann auch wieder fehlerfrei eingelesen werden.

Das Linetracking-Modul wird dafür benötigt, Streckenmarkierungen auf der Schiene einzulesen. Diese bestehen aus schwarzem Tape und werden mit drei voneinander unabhängigen Lichtschranken eingelesen. Ein Elementplatz wird mit einem Streifen auf der rechten Seite markiert, und der Anfang der Schiene und gleichzeitig der Parkplatz mit einem Streifen auf der rechten Seite. So kann der Roboter gezielt anhalten, wenn er einen bestimmten Stopp anfahren möchte. Hierbei ist es wichtig, dass sich die Anhalteposition des Roboters verschiebt, in Abhängigkeit davon, aus welcher Richtung er kommt. Dem liegt zugrunde, dass die Markierung eine gewisse Breite haben muss, um zuverlässig eingelesen werden zu können. Dies bewirkt, dass der Roboter anders anhält, wenn er die Markierung von der anderen Seite erreicht und dadurch früher bremst. Deshalb wird der Roboter zum Aufheben oder Absetzen eines Elementes immer von rechts kommen, auch wenn das bedeutet, dass er wieder an den Anfang fahren muss.


*Abbildung 6:Haltepositionen (Orange) je nach Fahrtrichtung (Pfeile)*

### 2.2.4. Lichtschranke

Der ursprüngliche Ansatz für das Einlesen der Elemente bestand daraus, binärcodierte Aufsätze auf die Elemente zu setzten und diese dann mit einer Lichtschranke einzulesen. Eine reflexive Lichtschranke funktioniert, indem sie mit zwei LEDs den Abstand zu dem Objekt vor ihr misst. Dafür gibt es eine Sender- und eine Empfänger-Diode. Der Sender emittiert Licht, während der Empfänger die Stärke des einfallenden Lichtes misst (siehe Abbildung 7).

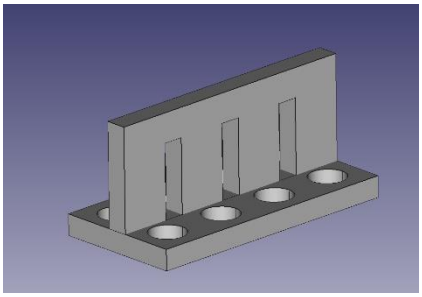

*Abbildung 8: Prototyp eines Aufsatzes*


*Abbildung 7: Reflexive Lichtschranke*


*Abbildung 9: Funktionsweise einer Reflexiven Lichtschranke*

Nach einigen Versuchen stellte sich aber heraus, dass die Lichtschranken leider zu unzuverlässig sind. Einerseits verfügen sie nicht über die nötige Präzision, um den Binärcode einlesen zu können, andererseits erwiesen sie sich als überaus fehleranfällig. Zum Teil lösten sie einfach aus oder reagierten oft gar nicht. Also musste eine neue Lösung her.

### 2.2.5. RFID

Nötig war eine Möglichkeit, Elemente ohne Kontakt und zuverlässig einzulesen. So eine Möglichkeit ist „radio-frequency identification“ (kurz: RFID). RFID ist ein Verfahren, bei dem durch Erzeugung eines elektromagnetischen Feldes ein Transponder aktiviert wird, der die geforderten Informationen kontaktlos zurückschickt. Durch die Kontaktlosigkeit und Fehlersicherheit ist es perfekt für das Projekt geeignet. Verwendet wurde ein MFRC522 RFID-Modul. Da der Platz auf dem Smartcar begrenzt war, wurde eine 3D-gedruckte Halterung für das Lesemodul entwickelt, um diese unterbringen zu können.

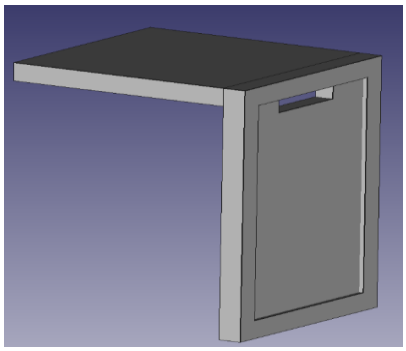

*Abbildung 10: 3D-Modell der Halterung*


*Abbildung 11: MFRC522 RFID-Modul mit Transponderchip und -karte*

Die Halterung wurde mit dem Programm Fr eeCAD entwickelt und mit dem 3D-Drucker der Schule gedruckt. Bevor das Lesegerät in der Halterung verbaut werden konnte, musste zunächst eine Pinleiste an die Platine angelötet werden, damit ein Anschließen überhaupt möglich wurde. Anschliessend konnte das Modul in die Halterung geklebt werden, und diese wurde wiederum an dem Roboter befestigt. Dazu

wurde eine Lego®®-Platte mit Löchern, welche zufällig perfekt auf eine vorhandene Schraube am hinteren Ende des Roboters passten, darunter befestigt, sodass man die Halterung problemlos aufstecken oder abnehmen kann, ohne sie permanent befestigen zu müssen.

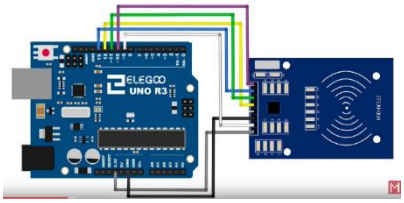

*Abbildung 12: Anschluss an den Arduino*

Das MFRC522 wird mit 7 Verbindungen an den Arduino angeschlossen, davon zwei für Strom (VCC und GND), 4 für Datenübertragung sowie einen Reset-Pin. Alle Pins bis auf den SS-Pin, den RST (Reset) Pin und die Stromversorgung sind vorgegeben, je nach Arduino-Modell. Der RST-Pin ist ein Zurücksetzungsmechanismus analog des roten Knopfes des Arduino selber, welcher einen Neustart auslöst. Der SS-Pin („**S**erial **S**ystem“) versetzt den Chip in den Lesemodus. Aufgrund der Pinvorgaben müssen auch nur diese beiden Pins separat im Code deklariert werden. Als Test wurde einfach die UID von mehreren Transpondern eingelesen und ausgegeben. Wichtig ist, dass der Datensatz der UID in Hexadezimalcode abgegriffen wird, und daher Byte für Byte mit einem entsprechenden Algorithmus eingelesen und in einen String umgewandelt werden muss. Jeder Transponder hat eine einzigartige UID.


*Abbildung 13: Halterung am Roboter*

### 2.2.6. Greifarm

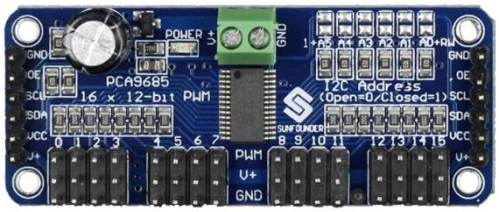

*Abbildung 14: 16-Channel Servodriver von Sunfounder*

Um die Elemente überhaupt sortieren zu können, musste ein Weg gefunden werden, diese zu transportieren. Dafür war die offensichtlichste Lösung ein servobetriebener Greifarm. Um allerdings vier Servos auf einmal zu betreiben, benötigt man einen Servodriver. Der Stromverbrauch der Servos ist ansonsten zu stark für den Arduino. Der „Adafruit 16 Channel Servodriver“ übernimmt genau diese Aufgabe. Bis zu 16 Servos (Pinleisten nummeriert von 0 bis 15) können mit 4 Anschlüssen und externer Stromversorgung präzise kontrolliert werden. Trotzdem wirkt der Chip erstmal sehr unübersichtlich, aber der Hersteller Adafruit hat eine sehr detaillierte Anleitung vom Anschluss bis zu Programmierung auf seiner Internetseite veröffentlicht, die fast alle Probleme klären konnte. Einzig die Stromversorgung warf noch Probleme auf, da die Batteriebox des Smartcars eine zu hohe Spannung aufwies, um die Motoren mit genügend Strom versorgen zu können. Der Servodriver akzeptiert nur Stromquellen mit 5V und mindestens 1A. Zufälligerweise ist dies exakt die Spannung, die handelsübliche Powerbanks liefern. Nach einigen Tests stellte sich heraus, dass eine vorhandene Powerbank perfekt geeignet war, um sowohl die Servos als auch den Hauptcontroller mit Strom zu versorgen. Dadurch konnte dieses Problem gelöst und der Greifarm konstruiert werden.

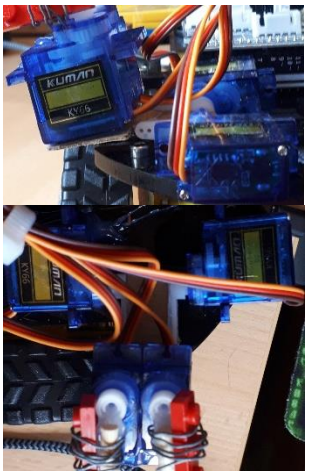
*Abbildung 15: Anordnung der Servos im Arm*

Im Vorfeld wurden zunächst die Servos ohne Aufsatz installiert, um den groben Ablauf zu simulieren. Anschließend musste aber noch ein passender Aufsatz entwickelt werden.

*Abbildung 17: Prototyp eines Elementes aus Duplo®Steinen*

Dazu musste aber ein Element entwickelt werden, dass von der Konstruktion gegriffen werden konnte. Als erster Prototyp wurde ein T-Stück aus Duplo®-Steinen verwendet. Dieses Model entpuppte sich aber als Fehlversuch, da das Gewicht zu hoch für die Servos war. Stattdessen war der zweite Versuch, einfach die Chipkarten als Element an sich zu verwenden erfolgreich. Der Plan bestand darin, eine Aufhängung an der Karte zu befestigen, an welcher der Roboter sie dann greifen konnte, sowie ein Standfuß, damit die Karte nicht umkippt. Mit Lego®-Platten, die mit Draht an den Servoarmen befestigt waren, sollten die Elemente hochgehoben und festgehalten werden. Mit Pappe, Heißkleber und einer Lego®-Platte wurde ein erster Prototyp gefertigt und getestet. Nach geringfügigen Modifikationen konnte der Arm die Karte weitestgehend zuverlässig greifen. Beim Abstellen wurde die Notwendigkeit deutlich, dass die Greifarme die Karte möglichst langsam und behutsam abstellen müssen, damit sie nicht verrutscht oder umkippt. Darüber hinaus wurde klar, dass eine Art Parkbucht benötigt wird, damit die Karte an ihrem Platz stehen bleiben kann. Diese Parkbuchten wurden aus schiefen Lego®steinen gebaut und halten die Karten perfekt auf ihren Plätzen, wodurch ein reibungsloses Aufheben und Abstellen ermöglicht wurde. Im Verlauf der Tests wurden noch diverse Verbesserungen an den Kartenhaltern vorgenommen, was zu vier verschiedenen Modellen geführt hat.

*Abbildung 18: Armaufsatz für den Roboter*


*Abbildung 19: 2. Prototyp eines Elementes*


*Abbildung 16: Verschiedene Kartenmodelle*

Der erste Kartenhalter bestand einfach aus dicker Kartonpappe, die an der Karte befestigt wurde. Der Fuß blieb für alle Kartenmodelle gleich. Der Nachteil dieser Version bestand aber darin, dass die Pappe aufgrund einer hohen Reibung auf der Oberfläche des Greifarmes nicht richtig an in den Greifer rutschte, sodass sie beim Transportieren die anderen Karten aus ihren Parkplätzen warf. Der zweite Versuch sollte dies verhindern, indem dünnere und glattere Pappe, mit Draht stabilisiert und befestigt wurde. Trotz starker Verbesserung bliebt das Problem bestehen. Um dem entgegenzuwirken, wurde beim dritten Halter ein dicker, glatter Strohhalm verwendet. Dies bewirkte, dass die Karte immer in den vorhergesehenen Platz rutscht. Ein weiteres Problem war, dass die Karte durch das Aufheben stark vor- und zurückschwang, was wiederum Probleme beim Vorbeifahren an den anderen Karten verursachte. Eine Verzögerung von fünf Sekunden gab den Karten genügend Zeit, um sich vor dem Weiterfahren auszupendeln.

### 2.2.7. Konsolidierung Fahren und Greifen

Um die Koordination zwischen Fahren und Greifen zu ermöglichen, muss eine einwandfreie Kommunikation bestehen. Diese wurde durch „2-Kanal Serielle Kommunikation“ zwischen den Controllern hergestellt, das heißt, dass die Controller über zwei Schnittstellen miteinander verbunden sind. Da der Roboter nur vorwärtsfahren, rückwärtsfahren und anhalten können muss, reichen die möglichen Kombinationen vollkommen aus. Die beiden Verbindungen wurden mit „Cpn1“ und „Cpn2“ bezeichnet. Der Code hierzu ist einfach, es wird eine im Loop laufende *if*-Abfrage erstellt, die testet, welche der drei Kombinationen vorliegt.

| Cpn1 | Cpn2 | Roboter |
|---|---|---|
| HIGH | HIGH | Vorwärts |
| HIGH | LOW | Rückwärts |
| LOW | LOW | Anhalten |
| LOW | HIGH | Nichts |

Damit die Kommunikation überhaupt stattfinden kann, ist es essenziell, wieder Abflusswiderstände einzubauen, damit sich kein Ladungsstau aufbaut (2.2.3). Wenn das gewährleistet ist, besteht eine solide Koordination. Dies macht es möglich, das Smartcar fahren zu lassen und Elemente aufzuheben, trotz zwei bisher unabhängigen Systemen. Wie bereits erwähnt, ist es beim Greifen wichtig, dass der Roboter von rechts anfährt, da er ansonsten zu früh stehen bleiben würde (2.2.3).

Der fahrende Controller erhält seinen Strom von der Batteriebox am hinteren Ende des Roboters, die gleichzeitig auch das L289N mit Motorstrom versorgt. Der Hauptcontroller hingegen bezieht seine Energie aus der Powerbank genau wie der Servodriver.

### 2.2.8. Sortieren

Als ersten Praxistest sollte der Roboter die Werte von zwei Karten erkennen und diese in ein Array einspeichern. Der Hauptcontroller bekommt im Sourcecode einen Parameter namens *Elementanzahl* zugewiesen, welcher beschreibt, wie viele Elemente sich auf der Bahn befinden. Dies ist wichtig, da der Roboter nur schwer selbst erfassen kann, wann er aufhören muss zu zählen. Also wird stattdessen ein vordefiniertes Array der Länge *Elementanzahl* initialisiert, welches der Roboter dann später füllt. Außerdem werden im Programm die UIDs der verwendeten Karten mit ihren zugehörigem Integer-Wert als Konstanten hinterlegt. Dies erspart Speicherschreibungen auf den Kartentranspondern, die mühsam einzeln übertragen werden müssten.

Nun muss das Smartcar die Karten aber auch einlesen können. Hier agiert der Roboter mit Hilfe von zwei Methoden: *toNextStop()*, und *readCard()*. Erstere bewegt den Roboter auf den jeweils nächsten Haltepunkt auf den Schienen, *readCard*() ist für das Lesen der Karten zuständig. Da der Haltepunkt für das Aufheben und Abstellen markiert ist, muss der Roboter noch ein Stück weiterfahren, bis das RFID-Lesegerät in der richtigen Position steht. Praktischerweise meldet das MFRC522, sobald es eine neue Karte vor sich hat, sodass keine weiteren Streckenmarkierungen notwendig waren. Der Roboter fährt also vom Haltepunkt aus so weit, dass er die dort befindliche Karte einlesen kann. Die eingelesene UID wird dann mit einer Auswahl von bekannten UIDs verglichen, und der entsprechend zugewiesene Integer-Wert in die richtige Array-Position eingetragen. Danach fährt er zum nächsten Punkt weiter. Dies wird für jedes Element einmal durch-

geführt. Nachdem dies fehlerfrei funktionierte, war der nächste Schritt, bei zwei Karten einen Tauschfall zu erkennen, und entsprechend zu handeln. Alles, was dafür nötig war, war eine einfache *if*-Abfrage, die die beiden Werte nach dem Einlesen vergleicht und entscheidet, ob getauscht werden muss. Ist ein Tausch erforderlich, so wurden die Karten über einen Parkplatz getauscht. In der Methode *tauschen(x,y)* werden die Positionen so behandelt, dass der Roboter sich immer die weiter hinten stehende Karte zuerst greift und auf den Parkplatz bringt, damit er danach mit der anderen einfach bis zu dem dann leeren Platz vorfahren kann. Dies erspart unnötiges Vor- und Zurückfahren.

Um einen Bubble-Sort in den Code zu integrieren, musste der bisherige Größenvergleich um die für einen Bubble-Sort nötigen *For*-Schleifen erweitert werden.

Der Bubble-Sort ist ein Algorithmus, der mit Paarweisen Vergleichen ein Array auf- oder absteigend sortiert. Dabei wird das Array immer wieder von vorne nach hinten durchlaufen, was ihn nicht besonders effektiv macht.

### 2.3. Ergebnis

Das Smartcar ist in der Lage, drei Karten auf festen Parkplätzen zu erfassen, abzuspeichern und mit einem Bubble-Sort in aufsteigender Reihenfolge zu sortieren. Die Anzahl der Karten könnte beliebig erhöht werden. Hierfür müssten nur weitere Karten angefertigt und im Programm registriert werden. Des Weiteren müsste die Konstante *Elementanzahl* im Sourcecode aktualisiert werden, sowie entsprechend neue Parkplätze konstruiert werden. Alle weiteren Soft- und Hardwarebestandteile sind generisch auf eine Erweiterung vorbereitet. Insbesondere der Bubble-Sort läuft unabhängig von der Anzahl der zu sortierenden Karten. Zukünftig könnte man über den Einsatz eines effizienteren Sortieralgorithmus nachdenken.

Probleme bereitet nach wie vor das bereits beschriebene Problem, dass das Smartcar nicht immer präzise genug anhält, was zu Fehleranfälligkeit beim Aufnehmen und Absetzen führt. Zusätzlich rutscht er bei zunehmender Geschwindigkeit trotz korrektem Bremsverhalten kleine Strecken weiter, wodurch sich der Anhaltepunkt kritisch verschiebt. Dies könnte vielleicht mit einer Verbesserung der Reibung auf der Fahrbahn, oder durch physische Barrieren, die ihn kurzzeitig stoppen verbessert werden.

Die Lego®-Konstruktion der Greifarme ist nicht glatt genug, um das richtige Rutschen der Karten in Position zu garantieren. Möglich wäre eine Umstellung der Arme auf 3D-gedruckte Bauteile, um die Glätte zu erhöhen.

Beim Sortiervorgang kommt es ab und zu dazu, dass der Roboter aus unersichtlichen Gründen steckenbleibt und nicht in der Lage ist, ohne Anstoß weiterzufahren. Eventuell ist die Übersetzung der Motoren nicht ausgelegt für langsame und präzise Bewegungen, oder die Reibung der Räder mit den Rändern der Schiene hält ihn auf.

## 2.4. Fazit

Die derzeitige Situation durch das Coronavirus hat vor allem dazu geführt, dass bei vielen Bauteilen improvisiert werden musste. Diese Kompromisse haben dazu geführt, dass der eigentlich fertige Sortiervorgang noch Verbesserungspotential hat (2.3).

Im Allgemeinen ist es klar geworden, dass viele Bewegungsprozesse deutlich langsamer als erwartet durchgeführt werden müssen, um Fehler zu vermeiden. Deswegen finden sich an vielen Stellen im Sourcecode Verzögerungen von unterschiedlicher Dauer.

Die Arduino IDE ist für längere Benutzung zu simpel und unkomfortabel gehalten. Vor allem das Fehlen einer Navigationsleiste oder einer Autovervollständigung hat viel Zeit und Nerven gekostet, insbesondere bei steigender Komplexität des Codes. Auch das Fehlen eines Debuggers hat das Testen und Fehlersuchen erschwert.

Es hat sich gezeigt, dass man viel Zeit sparen kann, wenn man von Anfang an auf eine Entwicklung nach CleanCode Standarts achtet, weil man dadurch unnötiges Refaktoring vermeiden kann.

Rückblickend muss man feststellen, dass man die meiste Zeit mit Testen, Fehlersuchen und kleinen Optimierungsschritten verbringt. Außerdem haben sich Tesafilm und die Heißklebepistole als sehr wichtige Hilfsmittel bewährt.

Ich habe bei der Entwicklung dieses Roboters sehr viel gelernt, sowohl über technische, als auch über theoretische Informatik. Vor allem das Entwickeln und Testen verschiedener Lösungsansätze hat viel Spaß gemacht. Ich bin grundlegend zufrieden mit dem Ergebnis, würde mich aber freuen mit dem 3D-Drucker noch einige Optimierungen vorzunehmen.

# 3. Quellen

https://www.destatis.de/DE/Presse/Pressemitteilungen/2018/12/PD18_470_52911.html

https://www.bito.com/de-de/fachwissen/artikel/was-ist-dynamische-lagerhaltung/

http://www.lagerwiki.de/index.php/Festplatzsystem

https://www.knapp.com/loesungen/technologien/lagern/

https://www.amazon.com/Qunqi-Controller-Module-Stepper-Arduino/dp/B014KMHSW6

http://labpacks.blogspot.com/2016/10/assembling-eLego®o-smart-robot-car-kit.html

https://www.amazon.de/ANGEEK-TCRT5000-Reflektierende-Lichtschranke-Op-tisch/dp/B07Q81NTBX/ref=sr_1_21?__mk_de_DE=%C3%85M%C3%85%C5%BD%C3%95%C3%91&dchild=1&keywords=Lichtschranke&qid=1585561589&sr=8-21

https://www.keyence.de/ss/products/sensor/sensorbasics/photoelectric/info/

https://www.amazon.com/SunFounder-Mifare-Reader-Arduino-Raspberry/dp/B07KGBJ9VG

https://www.rfid-grundlagen.de/

https://www.youtube.com/watch?v=M4ULblb9HD4

http://wiki.sunfounder.cc/index.php?title=PCA9685_16_Channel_12_Bit_PWM_Servo_Driver

(Zuletzt abgerufen und kontrolliert: 02.04.2020)

# 4. Anhänge / zusätzliche Dateien

1. FreeCAD-Datei für die RFID-Halterung
2. FreeCAD-Datei für den Aufsatzprototypen
3. Arduino-Programm für den Hauptcontroller
4. Arduino-Programm für den Fahrcontroller
5. Video eines Sortiervorganges
6. Selbständigkeitserklärung (unterschrieben und eingescannt)

# 5. Selbstständigkeitserklärung

**Versicherung der selbständigen Erarbeitung**

Ich versichere, dass ich die vorliegende Arbeit einschließlich evtl. beigefügter Zeichnungen, Kartenskizzen, Darstellungen u. ä. m. selbstständig angefertigt und keine anderen als die angegebenen Hilfsmittel benutzt habe. Alle Stellen, die dem Wortlaut oder dem Sinn nach anderen Werken entnommen sind, habe ich in jedem Fall unter genauer Angabe der Quelle deutlich als Entlehnung kenntlich gemacht.

Rheinbach, den 02.04.2020

_____

(Unterschrift)