

# **FH Aachen**

**Fachbereich Elektrotechnik und Informationstechnik**

Studiengang Informatik

Bachelorarbeit

## **Integration eines eingebetteten Systems in eine Cloud-Infrastruktur am Beispiel eines autonomen Schachttischs**

vorgelegt von

**Marcel Werner Heinrich Friedrich Ochsendorf**

Matrikel-Nr. **3120232**

Referent:

Prof. Dr.-Ing. Thomas Dey

Korreferent:

Prof. Dr. Andreas Claßen

Datum:

14.06.2021

# **Erklärung**

Hiermit erkläre ich, dass ich die vorliegende Arbeit eigenständig und ohne fremde Hilfe angefertigt habe. Textpassagen, die wörtlich oder dem Sinn nach auf Publikationen oder Vorträgen anderer Autoren beruhen, sind als solche kenntlich gemacht. Die Arbeit wurde bisher keiner anderen Prüfungsbehörde vorgelegt und auch noch nicht veröffentlicht.

Aachen, den 11.06.2021 \_\_\_\_\_

# **Abstract**

Das Ziel der Arbeit ist es, einen autonomen Schachtisch zu entwickeln, welcher in der Lage ist, Schachfiguren autonom zu bewegen und auf Benutzerinteraktionen zu reagieren. Die Kernfrage der Arbeit bezieht sich somit auf die Überprüfung der Ausführbarkeit inklusive Erstellung und Umsetzung eines eingebetteten Systems und einer Cloud-Infrastruktur. Der Schwerpunkt liegt dabei insbesondere auf der Programmierung des eingebetteten Systems und dem Zusammenspiel von diesem mit einem aus dem Internet erreichbaren Servers, welcher als Vermittlungsstelle zwischen verschiedenen Schachtischen und anderen Endgeräten dient.

Zuerst werden die zum Zeitpunkt existierenden Ansätze und deren Umsetzung beleuchtet. Hier wurde insbesondere darauf geachtet, die Grenzen bestehender Systeme darzulegen und auf nur für dieses Projekt zutreffende Funktionen zu vergleichen. Aus den Ergebnissen werden anschließend die Anforderungen des autonomen Schachttischs abgeleitet, welcher in dieser Arbeit umgesetzt werden sollen.

Anschließend wurden iterativ nacheinander Prototypen umgesetzt, welche sich vom mechanischen Design sowie des elektrischen stark unterscheiden. Die entwickelte Software wurde hingegen so modular entwickelt, dass diese auf beiden Prototypen zum Einsatz kommt. Die unterschiedlichen Designs kommen dadurch zustande, dass in der ersten Iteration des autonomen Schachttischs, nach einem Dauertest zahlreiche Verbesserungspotentiale erkannt wurden. Dies führte zu einer kompletten Neugestaltung in der zweiten Iteration und somit wurde ein autonomer Schachttisch entwickelt, welcher alle gestellten Anforderungen erfüllt.

Grundsätzlich ist festzuhalten, dass es sich beim Resultat der Arbeit um kein finalisiertes Produkt, sondern um einen strukturellen Prototyp handelt. Weitere Prüfungen, wie Nutzungsstatistiken oder Sicherheitsprüfungen, müssten durchgeführt werden, ehe der Schachttisch als kommerzielles Produkt betrachtet werden kann.

Das System und insbesondere der implementierte Cloud-Service sind online erreichbar und erweiterbar. Dies ermöglicht unter anderem das Bauen eines eigenen Tisches unter der Verwendung des AtomicChess Systems, aber auch die Integration weiterer Komponenten. Erfahrene Entwickler können somit das Spiel beliebig ausweiten oder sogar andere Spiele ergänzen. Die für das Projekt entworfene Mechanik und Spielführung kann dementsprechend auch für diverse andere Tischbrettspiele verwendet werden.

# Inhalt

<b>1 Einleitung</b>	<b>1</b>
1.1 Motivation . . . . .	1
1.2 Zielsetzung . . . . .	3
1.3 Aufbau der Bachelorarbeit . . . . .	4
<b>2 Analyse bestehender Systeme</b>	<b>6</b>
2.1 Existierende Systeme im Vergleich . . . . .	6
2.1.1 Kommerzielle Produkte . . . . .	7
2.1.2 Open-Source Projekte . . . . .	10
2.2 User Experience . . . . .	14
<b>3 Anforderungsanalyse</b>	<b>15</b>
<b>4 Machbarkeitsanalyse und Verifikation ausgewählter Technologien</b>	<b>17</b>
4.1 Erprobung Buildroot-Framework . . . . .	17
4.2 Verifikation NFC Technologie zur Identifizierung der Schachfiguren . . . . .	20
4.3 Auswahl des eingebetteten Systems . . . . .	22
4.4 Verifikation der Mechanik zur Bewegung der Schachfiguren . . . . .	24
4.5 3D Druck für den mechanischen Aufbau . . . . .	25
<b>5 Erstellung erster Prototyp</b>	<b>27</b>
5.1 Mechanik . . . . .	27
5.2 Parametrisierung der Schachfiguren . . . . .	29
5.3 Schaltungsentwurf . . . . .	31
5.4 Implementierung Hardware Abstraction Layer (HAL) . . . . .	34
5.4.1 TMC5160 SPI Treiber . . . . .	37
5.5 Fazit bezüglich des ersten Prototyps . . . . .	42
<b>6 Aufbau des zweiten Prototypen</b>	<b>44</b>
6.1 Modifikation der Mechanik . . . . .	44
6.1.1 Gehäuse und Design . . . . .	44
6.1.2 3D-Komponenten . . . . .	47

6.1.3 Positions-Mechanik . . . . .	48
6.2 Optimierung der Spielfiguren . . . . .	50
6.3 Änderungen der Elektronik . . . . .	52
6.4 Anpassungen HAL . . . . .	54
6.4.1 Implementierung GCODE-Sender . . . . .	54
6.4.2 I2C-Seriell Umsetzer . . . . .	57
6.5 Fazit bezüglich des finalen Prototyps . . . . .	60
<b>7 Entwicklung der Cloud Infrastruktur</b>	<b>63</b>
7.1 API Design . . . . .	63
7.2 Service Architektur . . . . .	65
7.3 Service: Backend . . . . .	69
7.4 Service: MoveValidator . . . . .	74
7.5 Service: Webclient . . . . .	77
7.6 Service: AutoPlayer . . . . .	79
<b>8 Embedded System Software</b>	<b>81</b>
8.1 Ablaufdiagramm . . . . .	81
8.2 Figur Bewegungspfadberechnung . . . . .	84
8.3 Schachfeld Scan Algorithmus zur Erkennung von Schachzügen . . . . .	87
8.4 Inter Prozess Communication . . . . .	88
8.5 User Interface . . . . .	90
<b>9 Fazit</b>	<b>95</b>
9.1 Persönliches Fazit . . . . .	97
9.2 Ausblick . . . . .	99
<b>Literaturverzeichnis</b>	<b>100</b>
<b>Abbildungsverzeichnis</b>	<b>107</b>
<b>Tabellenverzeichnis</b>	<b>108</b>

# 1 Einleitung

## 1.1 Motivation

Eingebettete Systeme (Englisch “embedded Systems”) sind technische Zusammensetzungen, welche für eine spezifische Funktion entwickelt werden. Im Gegensatz zu Mehrzwecksystemen (Englisch “multi-purpose systems”), wie zum Beispiel einem Personal Computer, welcher in der Lage ist, diverse Funktionen auszuführen und nicht zwingend an eine Funktion gebunden ist, dienen eingebettete Systeme einer bestimmten Logik. Daraus resultieren einfachere und auch ressourcen-schonendere Systeme, die wesentlich näher an der Technik und der für den Zweck nötigen Komponenten und Software entwickelt werden. Systeme können günstiger zusammengesetzt und Fehlerquellen schneller entdeckt und behoben werden. Nicht für den Prozess notwendige Komponenten werden gar nicht erst verwendet. Bei einem Mehrzwecksystem wird akzeptiert, dass Komponenten und Schnittstellen existieren, die nicht benötigt werden. Diese verursachen Kosten und können mögliche Fehlerquellen sein.

Dennoch ist die Entwicklung eines solchen Systems nicht banal. Es ist abzuwägen, welche Komponenten derzeit auf dem freien Markt erhältlich sind, welche Eigenschaften diese mitbringen oder ermöglichen und wie diese optimal kombiniert werden können. Es bedarf im Vorhinein intensiverer Recherche und einer größeren Perspektive über mögliche Zusammenhänge. Im Falle eines Mehrzwecksystems ist die Auswahl simpler, da man den Prozess auch im Nachhinein noch anpassen kann, da zusätzliche Funktionen und Komponenten gegeben sind oder leichter ergänzt werden können. Das eingebettete System muss in der Regel aufgewertet oder sogar völlig ersetzt werden, wenn zu einem späteren Zeitpunkt festgestellt wird, dass Funktionen nicht gegeben oder umsetzbar sind. Fertige Systeme sind komplexer in der Aufwertung.

Die Fähigkeit zur Erstellung eines solchen Systems ist daher nicht als eine leichte Aufgabe anzunehmen und es ist mir wichtig, zum Abschluss meines Studiums mein gewonnenes Wissen über Systeme, Komponenten, Zusammenhänge und deren Verbin-

dung bis hin zur Programmierung nachzuweisen. Die Auswahl eines fertigen Computers oder sogar das simple Nutzen existierender Betriebssysteme erweckt nicht den gleichen Reiz, wie es die eigene Erstellung dieser Komponenten auf mich hat. Ich halte es für essenziell, möglichst fachlich die Inhalte meines Studiums in Verbindung mit meinen Vorlieben zu bringen, um ein optimales Projekt zu erstellen.

Das dafür gewählte Projekt beinhaltet die Konstruktion und Automatisierung eines Schachtischs. Schach ist ein strategisches Brettspiel für gewöhnlich zwei Personen, bei welchem jeder Spieler 16 Figuren mit diversen Zug-Möglichkeiten nutzen kann, um den König des gegnerischen Spielers zu schlagen. Das Spiel basiert auf logischen Regeln und Spielverfahren, die bis hin zu komplexen Formationen und Bewegungen ausgebaut werden können.

Die Erstellung eines autonomen Schachtischs vereinbart in meinen Augen im großen Umfang die wesentlichen Komponenten des Informatikstudiums mit meiner Vorliebe zur mechanisch-elektrischen Gestaltung. Angefangen mit den Grundlagen der Informatik, insbesondere mit technischem Bezug, über die Berechnung und Auslegung von Systemkomponenten, zudem die objektorientierte Projektplanung und Architektur von Systemen bis hin zu Datenbanken und Webtechnologien und Softwareentwicklung. Zudem wird mein Studienschwerpunkt, die technische Informatik, mit einem einbetteten System manifestiert.

Schach ist ein bewährtes, ausnahmslos bekanntes und immer logisches Spiel, welches jedoch im kommerziellen Rahmen nie an Bedeutung gewonnen hat. Die Auswahl der verfügbaren elektrifizierten und programmgesteuerten Schachtische ist auffallend gering; zudem sind existierende Lösungen oftmals nicht erschwinglich und bedürfen erheblicher Anpassungen des Spielers an das Spiel. Innerhalb der vergangenen drei Jahrzehnte bewiesen immer mehr Konzerne ihre technische Kompetenz und Überlegenheit und die Fähigkeit ihrer Maschinen mittels der Optimierung von Schachalgorithmen und dem möglichst schnellen Besiegen derzeitiger Schach-Meister und -Meisterinnen. Die Algorithmen stehen heute in einer Vielzahl frei zugänglich zur Verfügung, jedoch ist das Interesse daran, für Spieler mögliche Anwendungen zu generieren, verschwindend gering und wird oftmals nur von Experten und Enthusiasten genutzt und auch hinterfragt.

Mit dieser Arbeit möchte ich mich diesem Problem stellen und einen möglich günstigen Tisch entwickeln, welcher das Spielerlebnis ohne Einschränkungen dem Spieler transferiert. Zudem möchte ich gewonnene Erkenntnisse und aktuelle Ressourcen wie die Cloud-Infrastruktur einbinden, um das Schachspiel, welches zweier Spieler bedarf, für

einen einzelnen Spieler zu ermöglichen. Das Ergebnis soll nicht nur aus vielen Zeilen Code bestehen, sondern auch ein handfestes Produkt sein, das meine Qualitäten und meinen Enthusiasmus widerspiegelt.

## **1.2 Zielsetzung**

Das Ziel der hier vorliegenden Arbeit ist es, einen autonomen Schachtisch zu entwickeln, welcher in der Lage ist, Schachfiguren autonom zu bewegen und auf Benutzerinteraktionen zu reagieren. Darüber hinaus sollte der autonome Schachtisch weitere folgende Funktionalitäten aufweisen:

- Erkennung Figur-Stellung
- Automatisches Bewegen der Figuren
- Livestream des Spiels
- Parkposition für ausgeschiedene Figuren
- Stand-Alone Funktionalität

Die Kernfrage der Arbeit bezieht sich somit auf die Überprüfung der Ausführbarkeit inklusive Erstellung und Umsetzung eines eingebetteten Systems und einer Cloud-Infrastruktur.

Der Schwerpunkt liegt dabei insbesondere auf der Programmierung des eingebetteten Systems und dem Zusammenspiel von diesem mit einem aus dem Internet erreichbaren Server, welcher als Vermittlungsstelle zwischen verschiedenen Schachtischen und anderen Endgeräten dient. Das Projekt umfasst zum einem die Positionserkennung und Steuerung der Hardwarekomponenten (Schachfiguren) und zum anderen die Kommunikation zwischen dem Tisch selbst und einem in einer Cloud befindlichem Server. Mittels der Programmierung werden diverse Technologien von verschiedenen Einzelsystemen zu einem Gesamtprodukt zusammengesetzt. Insgesamt gilt es, einen für Anwender ansprechenden Schachtisch zu entwickeln, der das Spielerlebnis nicht nur originalgetreu widerspiegelt, sondern das Einzelspieler-Modell zusätzlich noch verbessert.

Der Grundgedanke dabei ist, dem Spieler die Arbeit des Versetzens der Spielfiguren und das Erwägen von gegnerischen Zügen abzunehmen. Somit ist ein komfortables Spiel auch gegen einen virtuellen Gegner möglich, was dazu führt, dass bei einem Einzelspielerspiel das Überlegen und Ausführen von Spielzügen der Gegenseite entfällt.

Dem Spieler wird zudem die Möglichkeit geboten, gegen andere Spieler an diversen Orten oder gegen eine Schachlogik zu spielen und so Züge auszuführen, die jener im besten Fall nicht einmal vorhergesehen hat. Zudem wird die Korrektheit der getätigten Züge überprüft und sämtliche traditionellen Spielregeln in das Spiel mit einbezogen. Somit ist es nicht nur möglich, dass Anfänger das Spiel erlernen können, sondern auch bewährten Spielern mit unerwarteten Zügen des virtuellen oder realen Gegners neue Methodiken darzustellen.

Dies soll mittels eines kompakten und minimalistischen Designs realisiert werden. Darüber hinaus spielt nicht nur das Design eine Rolle, sondern auch die Handhabung. Dazu muss der Benutzer in der Lage sein, den Tisch in wenigen Handgriffen betriebsbereit machen zu können und über eine einfache Bedienoberfläche eine neue Partie gegen den Computer oder einen anderen Menschlichen spieler beginnen zu können.

### **1.3 Aufbau der Bachelorarbeit**

Im zweiten Kapitel der hier vorliegenden Arbeit werden die zum Zeitpunkt des Projektstarts existierenden Produkte beziehungsweise öffentlich einsehbare Projekte und deren Umsetzung beleuchtet. Hier wird insbesondere darauf geachtet, die Grenzen bestehender Systeme darzulegen und nur für dieses Projekt zutreffende Funktionen zu vergleichen.

Die Anforderungsanalyse im dritten Kapitel fasst alle zuvor recherchierten Funktionen bestehender Systeme zusammen und leitet daraus eine Auflistung der Anforderungen ab, welche in den nachfolgenden Prototypen realisiert werden sollen. Hierbei wird darauf geachtet, dem Benutzer einen Mehrwert in Bezug auf die Benutzerfreundlichkeit und dem Umfang an Features zu bieten.

Nach der Festlegung der Anforderungen wird im vierten Kapitel eine Machbarkeitsanalyse durchgeführt. In dieser wird untersucht, welche Technologien benötigt werden um, diese Anforderungen durch einen Prototyp erfüllen zu können.

Anschließend werden im fünften Kapitel die zuvor verwendeten Technologien betrachtet, welche bei den beiden darauffolgenden Prototypen verwendet wurden. Hierbei stehen insbesondere solche Technologien im Vordergrund der Untersuchung, welche möglichst einfach zu beschaffen sind und optimalerweise uneingeschränkt und lizenzunabhängig zur Verfügung stehen.

Das sechste Kapitel widmet sich der Realisierung eines ersten Prototyps des autonomen Schachtischs. Hier werden die Erkenntnisse der zuvor evaluierten Technologien verwendet, um ein Modell zu entwickeln, welches den im ersten Abschnitt erarbeiteten Vorgaben entspricht. Der nach der Implementierung durchgeführte Dauertest soll zudem weitere Risiken, mögliche Probleme und Fehlerquellen aufdecken.

Im anschließenden siebten Kapitel wird auf der Basis des ersten Prototypens und dessen im Betrieb verzeichneten Probleme der finale Prototyp entwickelt.

Hier werden die Schwierigkeiten durch die Vereinfachung der Elektronik sowie der Mechanik gelöst. Die Zuverlässigkeit wurde mittels stetiger Testläufe mit kontrollierten Schachzug-Szenarien überwacht und so ein produktreifer Prototyp entwickelt.

Im darauffolgenden achten Kapitel wird die Cloud-Infrastruktur thematisiert, welche für eine Kommunikation zwischen den autonomen Schachtischen entscheidend ist. Auch wird dabei die Software, welche auf dem eingebetteten System ausgeführt wird, im Detail beschrieben und deren Kommunikation mit der Cloud-Infrastruktur, sowie mit den elektrischen Komponenten beleuchtet. Zusätzlich zu dieser system-internen Software wurde ein WebClient entwickelt, mit dem es Benutzern möglich ist, über einen Webbrower gegen den Tisch zu spielen. Dieser Client bot außerdem die Möglichkeit, das System schon im Laufe des Entwicklungsprozesses testen zu können.

Das neunte Kapitel beschreibt die Software, welche auf dem eingebetteten System ausgeführt wird. Dessen Hauptaufgabe ist die Ansteuerung der Hardware-Komponenten, welche im autonomen Schachtisch verbaut sind. Darunter zählt die Ansteuerung der Motoren um die Figuren autonom bewegen zu können und die Erkennung der vom Benutzer getätigten Schachzüge. Das System befindet sich unterhalb des Spielbretts und übersetzt die von der Cloud-Infrastruktur abgefragt über das Internet empfangenen Spieldaten in Spielzüge. Dabei gilt ein besonderes Augenmerk der Berechnung der Figur-Bewegungen und dem Erkennen von durch den Benutzer getätigten Schachzügen.

Das zehnte und abschließende Kapitel umfasst das Fazit und gibt einen Ausblick auf mögliche Erweiterungen und Verbesserungen.

# 2 Analyse bestehender Systeme

## 2.1 Existierende Systeme im Vergleich

Im Folgenden werden vier kommerzielle und drei lizenzenabhängige (Open-Source) Schachtische miteinander verglichen. Bei den ausgewählten Tischen handelt es sich um

- kommerziell
  - Square Off - Kingdom [13]
  - Square Off - Grand Kingdom [12]
  - DGT Smartboard [4]
  - DGT Bluetooth Wenge [3]
- open-source:
  - Automated Chessboard (Michael Guerero) [10]
  - Automated Chessboard (Akash Ravichandran) [23]
  - DIY Super Smart Chessboard [16]

Für die kommerziell käuflichen Schachtische und (intelligenten) Schachbretter 4.1 gibt es kein sehr großes Marktangebot, weswegen für den Vergleich nur zwei Hersteller mit jeweils zwei verschiedenen Modellen gewählt werden konnten. Derzeit integriert nur eines dieser Unternehmen, [Square Off](#), eine Funktion, welche die Figuren auf dem Tisch mittels einer Mechanik unterhalb der Tischplatte bewegen kann.

Der zweite Hersteller [DGT](#) wurde dennoch zum Vergleich von zusätzlichen Funktionen herangezogen, da dessen Schachbretter die aktuelle Figur-Stellungen erkennen können. Die Tische des Herstellers [DGT](#) unterscheiden sich kaum in ihren Basis-Funktionen; mit steigendem Preis werden zusätzliche Funktionen in Form von Sensoren oder Verbindungsoptionen implementiert.

Das Angebot von open-source Schachtisch-Projekten 2.2 hingegen ist breiter, jedoch sind die einzelnen Modelle oftmals Kopien oder Revisionen voneinander. Die möglichen Funktionen unterscheiden sich daher kaum. Für die hier dargestellte Übersicht wurden drei Modelle gewählt, welche in ihren Funktionen signifikante Auffälligkeiten und einen hohen Stellenwert und Bekanntheitsgrad aufweisen. Wie bereits aus den zum Teil identischen Namen ersichtlich, streben alle Tische das gleiche Ziel an und unterscheiden sich daher nur geringfügig in ihren Funktionen, was im Folgenden nun näher erläutert wird.

### **2.1.1 Kommerzielle Produkte**

Tabelle 2.1: Auflistung kommerzieller autonomer Schachtische

	Square Off - Kingdom	Square Off - Grand Kingdom	DGT Smart Board	Bluetooth Wenge	DGT
Erkennung Figur-Stellung	nein (manuell)	nein (manuell)	ja	ja	
Abmessungen (LxBxH)	486mm x 486mm x 75mm	671mm x 486mm x 75mm	540mm x 540mm x 20mm	540mm x 540mm x 20mm	
Konnektivität	Bluetooth	Bluetooth	Seriell	Bluetooth	
Automatisches Bewegen der Figuren	ja	ja	nein	nein	
Spiel Livestream	ja	ja	ja	ja	
Cloud-Anbindung (online Spiele)	ja (App)	ja (App)	ja (PC + App)	ja (PC + App)	
Parkposition für ausgeschiedene Figuren	nein	ja	nein	nein	
Stand-Alone Funktionalität	nein (Mobiltelefon erforderlich)	nein (Mobiltelefon erforderlich)	nein (PC)	nein (PC)	
Besonderheiten	Akku für 30 Spiele	Akku für 15 Spiele	-	-	

Die für den Vergleich gewählten Eigenschaften sind jene, welche die im Projekt ange strebten Funktionen möglichst äquivalent reflektieren. Dennoch schränkt das geringe Angebot an autonomen Tischen die Auswahl stark ein; daher wurde hierbei Wert gelegt auf Automation, Cloud-Anbindung und die Abmessungen, welche das Spielerlebnis am deutlichsten beeinflussen.

Die Bretter des Herstellers DGT erkennen die Position der verwendeten Figuren. Information über die dafür verwendete Technologie ist jedoch nicht verfügbar. Die Square-Off-Schachtische verfügen über keine solche Funktion.

Die Abmessungen (Außenmaße) der autonomen Schachtische unterscheiden sich nur beim Hersteller Square Off deutlich; der Grand Kingdom Schachtisch ist rechteckig konstruiert worden, was das Spielerlebnis deutlich verändert. Der simple Kingdom-Tisch ist dabei kleiner als das vorgegebene Turniermaß, was negativen Einfluss auf das Spielererlebnis hat. Mit den Standardmaßen der DGT-Spielbretter und zudem ihrer geringen Höhe gleichen diese deutlich einem Turniertisch. Die Kombination aus geringer Höhe und Erkennung der Figur-Stellung bei den DGT-Brettern ist positiv bemerkenswert. Die Größe der einzelnen Schachfelder aller autonomen Schachtische befinden sich jedoch in den spezifikationen der United States Chess Federation (USCF)[15], welche sich von 5 bis 6.35cm belaufen.

Beide Hersteller bieten eine Bluetooth-Schnittstelle an. Einzig das Smart-Board des Herstellers DGT nutzt eine serielle, kabelgebundene Schnittstelle.

Bei den DGT-Schachbrettern ist zu beachten, dass diese die Schachfiguren nicht autonom bewegen können. Sie wurden jedoch in die Liste aufgenommen, da sie einen Teil der Funktionalitäten der Square Off Schachbrettern abdecken und lediglich die automatische Bewegung der Schachfiguren fehlt. Die DGT-Bretter können die Position der Figuren erkennen und ermöglichen so auch Spiele über das Internet; diese können sie auch als Livestream anbieten. Bei Schachturnieren werden diese für die Übertragung der Partien sowie die Aufzeichnung der Spielzüge verwendet und bieten Support für den Anschluss von weiterer Peripherie wie zum Beispiel Schachuhren.

Somit gibt es zum Zeitpunkt der Recherche nur einen Hersteller von autonomen Schachbrettern, welcher auch die Figuren bewegen kann: Grand Kingdom.

Ein Spiel-Livestream, d.h. eine Darstellung des aktuellen oder vergangener Spiele über eine Webanwendung, ist mit allen Tischen möglich. Da alle Tische eine Cloud-Anbindung besitzen, in der Regel mittels Applikation auf dem Smartphone oder Com-

puter, wird lediglich das Versetzen von Figuren detektiert und in einer Oberfläche dargestellt.

Auffallend ist, dass nur einer der ausgewählten Tische über eine Parkposition für ausgeschiedene Figuren verfügt. Der Square-Off-Grand, welcher Figuren automatisch verschieben kann, besitzt dank der rechteckigen Tischform die Möglichkeit, Figuren selbstständig aus dem Spiel zu entfernen und bei Bedarf wieder ins Spiel zurückzuführen.

Ebenfalls erwähnenswert ist, dass keiner der Tische eine Stand-Alone-Funktionalität besitzt. Jeder Tisch benötigt eine Verbindung zu einem externen Gerät, wie einem Smartphone oder Computer, welches dann die Berechnung der Gegnerzüge vornimmt. Keiner dieser Tische kann ein simples Spiel nach einem verbindungslosen Start ausführen.

Für die Schachtische der Firma [Square Off](#) ist eine Smartphone App [Square Off - Chess App](#)[11] für die Verwendung notwendig. Die App wiederum fordert eine Registrierung inklusive Profilerstellung beim Hersteller. Erst danach ist ein Spiel gegen den Computer ohne Internet möglich. Alle weiteren Optionen (wie bspw. Spiel gegen andere Spieler, Live-Stream) sind nur über einen Online-Zugang möglich und erfordern je nach gewählten Optionen auch einen weiteren Account bei anderen Schach-Cloud-Anbietern wie [Chess.com](#) oder [Lichess](#).

Beide Square-Off-Modelle ermöglichen durch eingebaute Akkus auch eine mobile Nutzung, was dem Nutzer mehr Flexibilität, zum Beispiel Spielen im Freien erlaubt.

Zusammenfassend ist festzustellen, dass alle vier Tische dank unterschiedlicher Ausführung von Spiel-Eigenschaften zu unterschiedlichen Spiel-Erlebnissen führen. Für Nutzer ist eine Entscheidung anhand von Funktionen kaum möglich; letztlich bedarf es vor einem Kauf der Auswertung von gewünschten und gegebenen Funktionen. Es ist erkennbar, dass nur die Firma [Square Off](#) einen absolut autonomen Schachtisch anbietet, auch wenn dieser nicht alle in diesem Projekt angestrebten Funktionalitäten bietet. So hat der Nutzer im Hinblick auf kommerzielle Angebote kaum Auswahlmöglichkeiten.

### **2.1.2 Open-Source Projekte**

Bei allen Open-Source Projekten wurden die Eigenschaften anhand der Beschreibung und der aktuellen Software extrahiert 2.2.

Besonders bei Projekten, welche sich noch in der Entwicklung befinden, können sich die Eigenschaften noch verändern und so weitere Funktionalitäten hinzukommen. Alle Eigenschaften der Projekte wurden zum Zeitpunkt der Recherche analysiert und dokumentiert und mit Beginn der Entwicklung als Struktur-Fixpunkt festgelegt. Nachfolgende Entwicklungen wurden nach diesem Zeitpunkt nicht mehr berücksichtigt.

Zusätzlich zu den genannten Projekten sind weitere derartige Projekte verfügbar; in der Tabelle wurde nur jene aufgelistet, welche sich von anderen Projekten in mindestens einem Feature unterschieden.

Auch existieren weitere Abwandlungen von autonomen Schachbrettern, bei welchen die Figuren von oberhalb des Spielbretts gegriffen bzw. bewegt werden. In einigen Projekten wird dies mittels eines Industrie-Roboters [31] oder eines modifizierten 3D-Druckers[17] realisiert. Diese wurden hier aufgrund der Mechanik, welche über dem Spielbrett montiert werden muss und damit das Spielerlebnis erheblich beeinflusst, nicht berücksichtigt.

Tabelle 2.2: Auflistung von Open-Source Schachtisch Projekten

	Automated Chess Board (Michael Guerero)	Automated Chess Board (Akash Ravichandran)	DIY Super Smart Chessboard
Erkennung Figur-Stellung	nein (manuell)	ja (Kamera)	nein
Abmessungen (LxBxH)	keine Angabe	keine Angabe	450mm x 300mm x 50mm
Konnektivität	Universal Serial Bus (USB)	Wireless Local Area Network (WLAN)	WLAN
Automatisches Bewegen der Figuren	ja	ja	nein
Spiel Livestream	nein	nein	nein
Cloud-Anbindung (online Spiele)	nein	nein	ja
Parkposition für ausgeschiedene Figuren	nein	nein	nein
Stand-Alone Funktionalität	nein (PC erforderlich)	ja	ja
Besonderheiten	-	Sprachsteuerung	Zuganzeige (Light-Emitting Diode (LED) Matrix)
Lizenz	General Public License (GPL) 3+	GPL	-

In den bestehenden Projekten ist zu erkennen, dass ein autonomer Schachtisch sehr einfach und mit simplen Mittel konstruiert werden kann. Hierbei fehlen in der Regel einige Features, wie das automatische Erkennen von Figuren oder das Spielen über das Internet. Einige Projekte setzen dabei auf eingebettete Systeme, welche direkt im Schachtisch montiert sind. Andere hingegen nutzen einen externen PC, welcher die Steuerbefehle an die Elektronik sendet.

Bei der Konstruktion der Mechanik und der Methode, mit welcher die Figuren über das Feld bewegt werden, ähneln sich jedoch die meisten dieser Projekte. Hier wurden in der Regel einfache X- und Y-Achsen verwendet, welche von je einem Schrittmotoren bewegt werden. Die Schachfiguren werden dabei mittels eines Elektromagneten über die Oberseite gezogen. Dabei ist ein Magnet oder eine kleine Metallplatte als Gegenpol in den Fuß der Figuren eingelassen.

Die Erkennung der Schachfiguren ist augenscheinlich die schwierigste Aufgabe. Hier wurde in der Mehrzahl der Projekte eine Kamera im Zusammenspiel mit einer auf OpenCV-basierten Figur-Erkennung verwendet. Diese Variante ist je nach Implementierung des Vision-Algorithmus fehleranfällig bei sich ändernden Lichtverhältnissen. Auch muss die Kamera oberhalb der Schachfiguren platziert werden, wenn kein transparentes Schachfeld verwendet werden soll.

Eine weitere Alternative ist die Verwendung einer Matrix aus Reed-Schaltern oder Halleffekt-Sensoren. Diese werden in einer 8x8 Matrix Konfiguration unterhalb der Platte montiert und reagieren auf die Magnete in den Figuren. So ist es möglich zu erkennen, welches der Schachfelder belegt ist, jedoch nicht konkret von welchem Figurtyp. Dieses Problem wird durch eine definierte Ausgangsstellung beim Spielstart gelöst. Nach jedem Zug durch den Spieler und der dadurch resultierenden Änderungen in den Figurpositionen in der Matrix können die neuen Figurstellungen berechnet werden.

Abschließend ist festzuhalten, dass es auch bei den Open-Source Projekten kein Projekt gibt, welches alle gewünschten Features abbildet. Auch fehlen dort weitestgehend Features, welche die kommerziellen Projekte bieten. Ebenso gilt dies für die kommerziellen Projekte, welche zwar viele Features bieten, jedoch dies zumeist in einem geschlossenen Ökosystem. Somit gibt es auch hier keinen klaren Testsieger, welche alle gewünschten Features abbildet. Das Ziel der hier vorliegenden Arbeit soll nun sein, all die positiven Eigenschaften dieser Tische zu vereinigen und mittels noch zusätzlicher Verbesserungen ein eigenes Produkt zu entwickeln.

## 2.2 User Experience

Ein wichtiger Aspekt bei diesem Projekt stellt die User-Experience dar. Diese beschreibt die Ergonomie der Mensch-Maschine-Interaktion und wird durch die DIN 9241[30] beschrieben. Darin geht es primär um das Erlebnis, welches der Benutzer beim Verwenden eines Produkts erlebt und welche Erwartungen der Benutzer an die Verwendung des Produkts hat.

Beim autonomen Schachtisch soll der Benutzer eine ähnlich authentische Erfahrung erleben wie bei einer Schachpartie mit einem menschlichen Gegenspieler. Der Benutzer soll direkt nach dem Einschalten des Tisches und dem Aufstellen der Figuren in der Lage sein, mit dem Spiel beginnen zu können. Dies soll wie ein reguläres Schachspiel ablaufen; der Spieler vor dem Tisch soll die Figuren mit der Hand bewegen können und der Tisch soll den Gegenspieler darstellen. Der Tisch soll somit die Figuren der Gegenseite bewegen.

Nach Beendigung einer Partie soll das Spielbrett wieder in die Ausgangssituation gebracht werden. Dies kann zum einem vom Tisch selbst oder vom Benutzer manuell geschehen. Danach ist der Tisch für die nächste Partie bereit, welche der Benutzer einfach per Knopfdruck starten können sollte.

Dies soll auch für abgebrochene Spiele gelten, welche von Benutzer oder durch das System abgebrochen wurden. Auch in diesen Fällen soll das Schachbrett sich selbstständig zurücksetzen können.

Ein weiterer Punkt, welcher bei der User-Experience beachtet werden soll, ist der zeitliche Aspekt. Ein Spiel auf einem normalen Schachspiel hat je nach Spielart kein Zeitlimit. Dies kann für das gesamte Spiel gelten oder auch für die Zeit zwischen einzelnen Zügen. Der autonome Schachtisch soll es dem Spieler zum Beispiel ermöglichen, ein Spiel am Morgen zu beginnen und dieses erst am nächsten Tag fortzusetzen.

Auch muss die Frage beantwortet werden, was mit den ausgeschiedenen Figuren geschehen soll. Bei den autonomen Schachbrettern von Square Off[12] werden die Figuren an die Seite auf vordefinierte Felder bewegt und können so bei der nächsten Partie vom System wieder aufgestellt werden. Viele andere Projekte schieben die Figuren aus dem Feld heraus, können diese aber im Anschluss nicht mehr gezielt in das Feld zurückholen. So muss diese Aufgabe vom Benutzer manuell durchgeführt werden. Von einigen Projekten wird die Behandlung ausgesiedelter Figuren gar nicht abgedeckt und der Benutzer muss die Figuren selbstständig vom Feld entfernen.

# 3 Anforderungsanalyse

Nach Abschluss der Recherche, kann somit eine Auflistung aller Features 3.1 angefertigt werden, welche ein autonomer Schachtisch aufweisen sollte. In diesem Projekt werden vor allem Funktionalitäten berücksichtigt, welche bei Bedienung und Benutzung des autonomen Schachttisches dem Benutzer einen Mehrwert in Bezug auf die Benutzerfreundlichkeit bieten.

Tabelle 3.1: Auflistung der Anforderungen an den autonomen Schachtisch

Atomic Chess Table (ATC)	
Erkennung Figur-Stellung	ja
Konnektivität	WLAN, USB
Automatisches Bewegen der Figuren	ja
Spiel Livestream	ja
Cloudanbindung (online Spiele)	ja
Parkposition für ausgeschiedene Figuren	ja
Stand-Alone Funktionalität	ja (Bedienung direkt am Tisch)

Die Abmessungen und das Gewicht des autonomen Schachttisches ergeben sich aus der mechanischen Umsetzung und werden hier aufgrund der zur Verfügung stehenden Materialien und Fertigungstechniken nicht festgelegt. Dennoch wird Wert darauf gelegt, dass der Unterschied zwischen den Spielfeldabmessungen und den Abmessungen des

### **3 Anforderungsanalyse**

---

Tisches so gering wie möglich ausfällt.

Auch müssen die Figuren für den Benutzer eine gut handhabbare Größe aufweisen, um ein angenehmes haptisches Spielerlebnis zu gewährleisten. Diese richten sich hierbei an die Größe des Spielfeldes. Des Weiteren gibt es auch hier von der USCF[15] Spezifikationen über die Größe der Figuren, jedoch sollen diese hier nicht zwingend angewendet werden, da diese speziell für den autonomen Schachtisch angepasst werden müssen um eine automatische Erkennung dieser durch den Schachtisch gewährleisten zu können.

Ebenfalls wird kein besonderer Augenmerk auf die Geschwindigkeit der Figur-Bewegung gelegt, solange dies nicht das Spielerlebnis stört. Die Zuverlässigkeit und Wiederholgenauigkeit dieser Bewegungen sollen dabei im Vordergrund stehen um einen reibungsfreien Spielablauf zu gewährleisten..

# **4 Machbarkeitsanalyse und Verifikation ausgewählter Technologien**

Da dieses Projekt aus vielen ineinander greifenden Komponenten besteht wurden zuerst Technologien ausgewählt, welche sich augenscheinlich für die Umsetzung eignen. Hier werden Technologien für die folgenden Komponenten benötigt:

- Erstellung von Software-Paketen für das eingebettete System
- Auswahl des eingebetteten Systems
- Identifizierung der Schachfiguren
- Mechanische Bewegung der Schachfiguren

## **4.1 Erprobung Buildroot-Framework**

Eine Hürde, welche bei diesem Projekt genommen werden musste, war die Erstellung der Software, welche auf dem autonomen Schachtisch ausgeführt wird. Hierbei sollte diese nicht von Grund auf neu entwickelt werden, sondern auf einer soliden Basis aufbauen. Allgemein soll hier auf ein minimales Linux-System gesetzt werden, in welches die Software des autonomen Schachtisch integriert wird. Auf dem Basis-System müssen die folgenden Software-Pakete installiert sein, bzw einfach integrierbar sein:

- Secure Shell (SSH) für den Remote-Zugriff
- Dynamic Host Configuration Protocol (DHCP) Client zur automatischen IPv4-Adressvergabe
- userspace device file system (UDEV) zur Ein-/Ausgabe Geräteverwaltung (Touchscreen)
- Qt[9] - Graphical User Interface (GUI) Framework

- SW-Update zur Durchführung von Remote-Software-Updates der Schachtisch-Software

Auf Seiten der Entwicklung war eine Toolchain notwendig, mit welcher es möglich sein sollte, in C++ geschriebene Programme auf dem System ausführen und mittels GNU Debugger (GDB) auf Fehler überprüfen zu können. Zusätzlich sollte der C++ Compiler mindestens den C++17 Standard unterstützen.

Zusätzlich zu diesen auf dem Linux-System benötigten Paketen sollte es möglich sein, ein durch das eingebettete System bootbares Dateisystemimage zu erzeugen. Für diesen Zweck existieren einige Open-Source Projekte, welche solch ein Build-System bereitstellen. Hierbei existieren zwei weit verbreitete Systeme: Das [Yocto](#)-Projekt[8] und das Buildroot[1]-Framework. Diese unterscheiden sich im Aufbau und der Funktionsweise teils stark, vor allem während der ersten Verwendung.

Tabelle 4.1: Vergleich Yocto - Buildroot

---

	YOCTO	BUILDROOT
Dependency-Management	Nein	Ja
Partielle Updates	Ja	Nein
Automatischer Paket-Download	Nein	Ja
Verwendung Build-Cache	Ja	Nein
Einfache Konfiguration	Nein	Ja

---

Hierbei stellt das [Yocto](#)- Projekt eine größere Einstiegshürde dar, aufgrund seines komplexen Layer-Systems.

Es bietet sich jedoch für komplexe Projekte an, welche einen hohen Grad an Individualisierung benötigen. Ein Nachteil dessen, ist da dadurch auch viel vom Nutzer selber konfiguriert werden muss, bevor ein minimales System in Betrieb genommen werden kann. Somit muss zuerst eine grundlegende Konfiguration für das eingebettete System

angelegt werden und es kann nicht direkt mit einem fertigen minimalen System mit der eigentlichen Software-Entwicklung gestartet werden. Allgemein sind die Anforderungen an die zu erstellende Software und deren benötigte requirements sehr gering, sodass hier eine detaillierte Einarbeitung zeitlich nicht möglich war.

Der Aufbau dieser Konfiguration des [Yocto](#)- Projekt besteht aus drei Schichten:

- Layer - Build-Metadaten
- Recipes - Pakete
- Bitbake - Tool zum erstellen

Eigene Pakete werden dabei in [Recipes](#) angelegt, welche in einer Makefile ähnlichen Syntax das Paket und die Schritte zum erstellen beschreiben. Diese einzelnen Pakete werden anschließend in [Layern](#) zu einem Gesamtsystem zusammengesetzt. Dies geschieht anschließend mittels des [Bitbake](#)-Tools, welches das System erstellt.

Das [Buildroot](#)-Framework hingegen bietet bereits eine große Anzahl an vorkonfigurierten Ziel-Systemen an, für welche es bereits alle nötigen Parameter enthält, um ein minimales solches System erstellen zu können. Auch ist bereits eine optimierte Konfiguration für das im Rahmen des hier vorliegenden Projekts gewählte eingebettete System vorhanden, welche direkt gestartet werden kann.

Nach einem erfolgreichen Erstellen des Images kann dieses direkt über das eingebettete System gestartet werden. Bei jedem Build-Vorgang müssen jedoch alle Pakete erneut gebaut werden, bevor diese zu einem finalen Image zusammengefügt werden. Hierbei kann dieser Vorgang je nach Umfang der verwendeten Pakete mehrere Stunden dauern. Das [Yocto](#)-Projekt unterstützt hierbei das Erstellen einzelner Pakete, somit müssen nur Änderungen neu gebaut werden. Da im Rahmen dieses Projekts nur eine minimale Anzahl von Paketen benötigt werden, hält sich dieser Buildvorgang zeitlich in Grenzen und ist allgemein für dieses Projekt nicht entscheidend.

Dann wurde prototypisch evaluiert, wie aufwändig es ist, zum [Buildroot](#) Framework eigene in C++ geschriebene Software hinzuzufügen. Zu diesem Zweck wurde ein eigenes C++ Paket erstellt und in das [Buildroot](#)-Framework integriert. Hierzu wurden dem [Buildroot](#) Quellcode zwei weitere Dateien hinzugefügt.

Die [Config.in](#) beschreibt das Paket und legt Abhängigkeiten zu anderen Paketen fest.

```
1 #atctp - Config.in - Package Configuration
2 config BR2_PACKAGE_ATCTP
```

```

3      bool "ATC_TEST_PACKAGE"
4      help
5          This package is a test package to verify the buildroot
              build process

```

Die zweite Datei ist die `PAKET_NAME.mk` - `Makefile`, welche die Schritte beschreibt, welche zum Erstellen und Installieren des Paketes durchgeführt werden müssen.

```

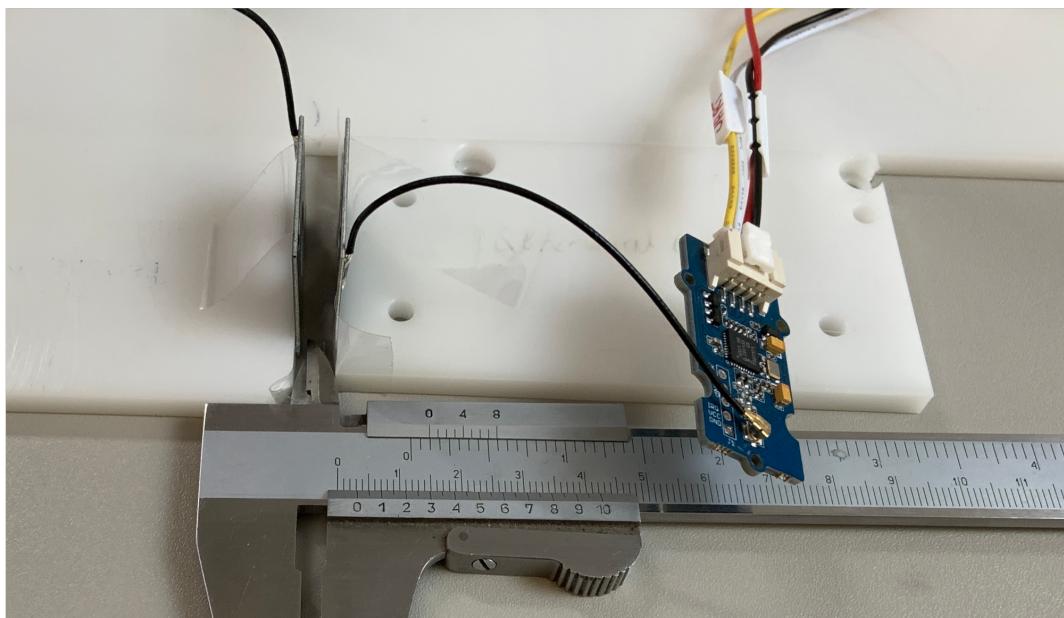
1 # atctp - atctp.mk - Makefile
2
3 ATCTP_VERSION = 1.0.0
4 ATCTP_SITE = ./package/atctp/src
5 ATCTP_SITE_METHOD = local
6 ATCTP_LICENSE = GPL-2.0+
7
8 define ATCTP_BUILD_CMDS
9     @echo ATCTP_BUILD!
10    @echo $(@D)
11    @echo -----
12    $(MAKE) $(TARGET_CONFIGURE_OPTS) -C $(@D)
13 endef
14
15 define ATCTP_INSTALL_TARGET_CMDS
16     @echo ATCTP_INSTALL!
17     $(INSTALL) -D -m 0755 $(@D)/hello $(TARGET_DIR)/usr/ATC/
18         atc_testpackage
19 endef
20 $(eval $(generic-package))

```

Das somit erstellte Test-Paket `atctp` bildete eine funktionierende Grundlage für das System. Somit eignet sich das `Buildroot`-Framework optimal für dieses Projekt, da hier der Prozess zur Integration von eigener Software sehr einfach gestaltet ist.

## 4.2 Verifikation NFC Technologie zur Identifizierung der Schachfiguren

Ein weiterer wichtiger Bestandteil sollte die Erkennung der sich auf dem Feld befindlichen Schachfiguren sein. Hierbei muss zum einen der Figur-Typ (König, Dame, Türme, Läufer, Springer, Bauern) und zum anderen die Figur-Farbe (schwarz, weiss) vom System erkannt werden.



**Bild 4-1:** Grove PN532 NFC Reader mit kabelgebundener Antenne

Da hier keine aufwendige Elektronik entwickelt werden, sondern auf Standard-Komponenten zurückgegriffen werden sollte, schied ein komplexes high frequency (HF) Antennen-Array unter dem Schachfeld aus, wie es bei einigen kommerziellen Produkten umgesetzt ist. Eine einfache 8x8 Matrix aus Drucktastern oder Hall-Effekt-Sensoren schied ebenfalls aus, da hier die Eingabe über den Benutzer erfolgt und nur Rückschlüsse auf die veränderte Figur anhand einer manuellen Bewegung der Figur nachvollzogen werden kann.

Stattdessen eignete sich hier die Near Field Communication (NFC) Technologie, welche auch bei modernen Smartphones eingesetzt wird. Hierzu werden kleine NFC-Tags bzw. Aufkleber, welche aus einem Chip und einer Antenne bestehen, so programmiert, dass diese eine definierte Aktion beim Lesegerät auslösen. Dies kann zum Beispiel das Öffnen einer Internetseite auf dem mobilen-Endgerät nach dem Scan eines mit einem NFC Tag ausgestatteten Filmplakats sein.

Ein Vorteil dieser Technologie ist, dass diese auch im Konsumerbereich bereits breit verfügbar ist. Durch das einfache Programmieren dieser NFC-Tags über das Smartphone wird kein zusätzliches Lese-/Schreib-Gerät benötigt. Hier musste jedoch zuvor getestet werden, welcher maximale Abstand erlaubt ist, um solch einen Tag noch scannen zu können. Auch ist der Abstand zwischen den einzelnen Tags entscheidend, d.h. wie nah diese beieinander platziert werden können, um trotzdem noch einwandfrei individuell ausgelesen werden zu können.

Hierzu wurde ein kleiner Testaufbau 4-1 entwickelt, um verschiedene Abstände testen zu können.

Als Lesegerät wurde ein [PN532](#) Modul zum Auslesen der NFC Tags eingesetzt, da dieses Modul einfach angesteuert werden kann und eine abnehmbare Antenne besitzt. Dieses Modul wurde vom Autor der hier vorliegenden Arbeit bereits in anderen Projekten eingesetzt und erwies sich dort als zuverlässig.

Die im Test verwendeten NFC Tags hatten einen Durchmesser von 22mm und waren somit ein Standard-Produkt. Im Test stellte sich heraus, dass diese Tags im gewählten Setup einen Abstand von mindestens 5mm zueinander benötigten. Der Abstand des Lesegeräts bzw. der Antenne zu einem Tag konnte dabei bis zu 14mm betragen.

Somit eignet sich die Kombination aus Tag und Lesegerät für eine Positionserkennung der Schachfiguren, wobei sich das Lesegerät unter der Schachtischplatte befindet.

### **4.3 Auswahl des eingebetteten Systems**

Zur Ansteuerung aller elektrischen Komponenten, welche den autonomen Schachtisch antreiben, wird ein eingebettetes System benötigt auf welchem die zur vor erstellte Software ausgeführt wird. Dies ist in diesem Fall, durch das [Buildroot](#)-Framework generierte Linux-Image. Um mit den elektrischen Komponenten kommunizieren zu können, werden zusätzlich verschiedene Ein- und Ausgabe-Schnittstellen benötigt:

- High Definition Multimedia Interface (HDMI) oder Display Serial Interface (DSI)  
Anschluss für externen Bildschirm
- USB
- General Purpose Input/Output (GPIO) Header mit Serial Peripheral Interface (SPI)  
/ Inter-Integrated Circuit (I2C)-Bus
- Local Area Network (LAN) oder WLAN

Da es sich hier um einen Prototyp handelt und die finalen Anforderungen der Software und Hardware noch nicht final feststehen, sollte das eingebettete System genug Spielraum bieten um spätere Erweiterungen umsetzen zu können. Auch spielte bei der Auswahl die Verfügbarkeit zum Zeitpunkt der Evaluation eine Rolle, da hier einige andere Systeme mit unter anderen langen Lieferzeiten aufwiesen. Zudem wurde darauf geachtet, dass das System bereits in einigen anderen Projekten verwendet wird, sodass

bei möglichen Problemen, Ressourcen für die Fehlerbehebung bereitstehen. Somit beschränkte sich die Auswahl für diesen Prototyp, angesichts des für die Evaluation abgesteckten zeitlichen Rahmen auf den [Raspberry-Pi 3b+](#) und das [STM32MP157A-DK1](#).

: Getestete eingebettete Systeme

| Raspberry-Pi 3b+ | STM32MP157A-DK |

|:-----|-----|-----:| | Prozessor | Arm Cortex-A53 | Arm Cortex A7 + Arm Cortex A4 | | Arbeitsspeicher | 1GB LPDDR2 SDRAM | 4-Gbit DDR3L | | Flash-Speicher | SD-Karte | SD-Karte | | GPIO-Header | ja (40 Pol, (+i2c),SPI) | ja (40 Pol, (+i2c),SPI) | | Netzwerkanschluss | LAN, WLAN | LAN, WLAN | | Besonderheiten | 64-bit Prozessor | Arm Cortex A4 Co-Prozessor, Secure-Boot, Debugger |

Diese bieten vergleichbare technische Spezifikationen und siedeln sich im gleichen Preissegment an. Je nach Ausführung des [STM32MP157A-DK](#) ist in diesem bereits ein DSI-Display vorhanden und bietet von der Softwareseite her bereits eine Unterstützung für das [Yocto](#)-Projekt an. Ein Vorteil des [STM32MP157A-DK](#) gegenüber dem bekannten [Raspberry-Pi 3b+](#) ist sein zusätzlicher [Arm Cortex A4](#)-Co-Prozessor, welcher als zusätzlicher Mikrokontroller verwendet werden kann. Somit kann die Ansteuerung der GPIO-Anschl sowie Timer und Interrupt-Aufgaben von diesem übernommen werden und die Ergebnisse und Steuersignale von [Arm Cortex A7](#) Hauptprozessor und dessen Linux-System bearbeitet werden.

Der [Raspberry-Pi 3b+](#) hingegen bietet einen großen Software-Support an, welches an der großen Open-Source-Community liegt, welche das Erfolgreiche System über die Jahre aufbauen konnte. Der [Arm Cortex-A53](#) 64-bit Quad-Core Prozessor bietet dabei viele Leistungsreserven und die GPIO Anschlüssen können direkt über das Dateisystem angesteuert werden. Einzig der 1 Gigabyte große Arbeitsspeicher und der Anschluss des Netzwerk-Chips über USB sind der Flaschenhals des Systems dar. Somit eignen sich beide Systeme von den Spezifikationen her für diesen Prototypen.

Anschließend wurde eine Test-Software erstellt und mittels des [Buildroot](#)-Frameworks ein passendes Linux-System erstellt. Auf beiden Systemen ist es möglich dieses Auszuführen, jedoch war es nicht möglich eine mittels [Qt](#) und der [Quick Controls II](#) Erweiterung erstellte GUI auf dem [STM32MP157A-DK](#) auszuführen. Dies liegt am zum Zeitpunkt der Evaluation nicht mit der Graphics Processing Unit (GPU) des Systems lauffähigen Version des [ELGFS](#)-Frameworks. Dieses wird verwendet, um Grafik mittels Hardwarebe-

schleunigung auf dem Display anzeigen zu können.

Somit wurde nach mehreren Versuchen dieses Problem zu lösen das [Raspberry-Pi 3b++](#)-System als eingebettetes System für den autonomen Schachtisch ausgewählt und die Entwicklung auf diesem fortgesetzt.

## **4.4 Verifikation der Mechanik zur Bewegung der Schachfiguren**

Da die einzelnen Figuren über das Schachfeld bewegt werden sollen, ist hierfür eine akkurate Positionierung dieser notwendig. Da die Figuren einen Durchmesser von 22mm haben und ein einzelnes Schachfeld ein Größe ca. 55mm besitzt, reicht eine Wiederholgenauigkeit von +-1mm. Auch wird bei der Wahl der passenden Motoren angenommen, dass das Spiel, welches durch die Mechanik in das System eingebracht wird, vernachlässigbar klein ist. Es ist auch davon auszugehen, dass die Kraft, welche von den Motoren benötigt wird, um eine Achse zu bewegen, nicht mehr als 45 Ncm betragen muss.

Dies entspricht den Werten einer X-Y-Achsenkonfiguration, wie sie in einem handelsüblichen 3D-Drucker zu finden ist, die mit [Nema 17](#)-Schrittmotoren ausgestattet sind. Der geplante Aufbau des autonomen Schachtischs ähnelt einer solchen Konfiguration sehr, da auch hier die Figuren in X-Y Richtung verfahren werden. Einzig die Z-Achse kommt hier nicht zum Einsatz. Somit wurden für erste Tests diese Motoren gewählt.

Deswegen bietet sich hier auch die Verwendung von Schrittmotoren an, da diese sehr kostengünstig in der geforderten Leistungsklasse zu erwerben sind und zudem die Ansteuerung einfach realisiert werden kann. Hierzu gibt es verschiedene Schrittmotor-Treiber, welche die Ansteuerung übernehmen. Diese bieten in der Regel ein [STEP](#), [DIR](#) Interface an. Der Schrittmotor-Treiber besitzt diese beiden Eingänge und jeder elektrische Impuls sorgt dafür, dass der Motor einen Schritt ausführt. Je nach gewähltem Motor entspricht dies einer Rotation um 1.8 Grad und dies reicht somit für die Positioniergenauigkeit aus.

Da das eingebettete System auf einem nicht-echtzeitfähigen Linux-System aufsetzt, ist hier eine zeitkritische Ansteuerung der Motortreiber nicht gewährleistet. Somit stellt sich das [STEP,DIR](#)-Interface für diesen Anwendungsfall als nicht ideal heraus. Um dieses

Problem zu umgehen, kann hier ein zusätzlicher Mikrokontroller eingesetzt werden, welcher die benötigten Impulse generiert.

Diese Option wurde zuvor getestet und erwies sich als eine robuste Alternative. Jedoch existieren Schrittmotor-Treiber, welche über zusätzliche Bus Schnittstellen verfügen. Hier wurde auf den [TMC5160-BOB](#) gesetzt, da dieser über ein SPI Interface verfügt, welches direkt an das eingebettete System angeschlossen werden kann.

Somit stellen die Schrittmotoren und die gewählte Ansteuerung ein vielversprechendes Antriebskonzept für den autonomen Schachtisch dar.

## 4.5 3D Druck für den mechanischen Aufbau

Da es sich hier nur um einen Prototyp handelt, wurde auf ein einfache zu verarbeitendes Filament vom Typ Polylactic Acid (PLA) zurückgegriffen. Dieses ist besonders gut für die Prototypentwicklung geeignet und kann mit nahezu jeden handelsüblichen Fused Deposition Modeling (FDM) 3D-Drucker verarbeitet werden.

Es wurden einige Testdrucke durchgeführt, um die Qualität der gewählten Druckparameterwerte 4.2 zu überprüfen und diese gegebenenfalls anzupassen. Auch wurden verschiedene weitere Bauteile gedruckt, an welchen die Toleranzen für die späteren Computer-Aided Design (CAD) Zeichnungen abgeschätzt werden konnten. Dies betraf vor allem die Genauigkeit der Bohrungen in den gefertigten Objekten, da hier später Bolzen und Schrauben nahezu spielfrei eingeführt werden müssen. Ein Test, welcher die Machbarkeit von Gewinden zeigen würde, wurde nicht durchgeführt, da alle Schrauben später mit der passenden Mutter gesichert werden sollten. So sollte eine Abnutzung durch zu häufige Montage der gedruckten Bauteile verhindert werden.

Beim Design der zu druckenden Bauteile wurde darauf geachtet, dass diese den Bauraum von 200x200x200mm nicht überschreiten und somit auch von einfachen FDM 3D-Druckern erstellt werden können.

Als Software wurde der Open-Source Slicer Ultimaker Cura [18] verwendet, da dieser zum einen bereits fertige Konfigurationen für den verwendeten 3D-Drucker enthält und zum anderen experimentelle Features bereitstellt.

Hier wurde für die Bauteile, welche eine Stützstruktur benötigen, die von Cura bereit-

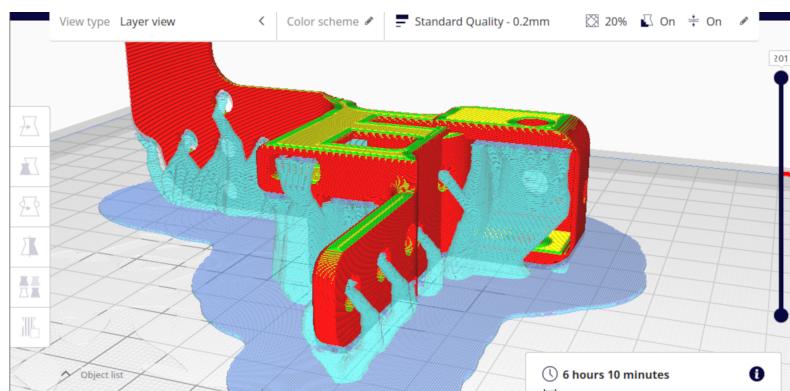


Bild 4-2: 3D Druck: Objekt (rot,gelb,grün),Tree Structure (cyan)

gestellte experimentelle Tree Support Structure aktiviert. **4-2** Diese bietet den Vorteil gegenüber anderen Stützstrukturen, dass sich diese leichter entfernen lässt und weniger Rückstände an den Bauteilen hinterlässt. Diese Vorteile wurden mit verschiedenen Testdrucken verifiziert und kamen insbesondere bei komplexen Bauteilen mit innenliegenden Elementen zum Tragen, bei denen eine Stützstruktur erforderlich war.

Tabelle 4.2: Verwendete 3D Druck Parameter. Temperatur nach Herstellerangaben des verwendeten PLA Filaments.

---

Ender 3 Pro 0.4mm Nozzle    PLA Settings

---

Layer Height	0.2mm
--------------	-------

Infill	50.00%
--------	--------

Wall Thickness	2.0mm
----------------	-------

Support Structure	Tree
-------------------	------

Top Layers	4
------------	---

Bottom Layers	4
---------------	---

---

Zusätzliche Parameter wie die Druckgeschwindigkeit wurden hierbei individuell für den gewählten 3D Drucker ermittelt. Allgemein wurden hier die Standardeinstellungen verwendet, welche in diesem Falle einen guten Kompromiss zwischen Qualität und Druckzeit lieferten.

# 5 Erstellung erster Prototyp

## 5.1 Mechanik

Bei dem mechanischen Aufbau wurde auf ein einfaches Design geachtet. Die Konstruktion wurde im Vorfeld in einem CAD Programm durchgeführt und die Grundkonstruktion in mehreren Iterationsschritten verfeinert. Das verwendete CAD Programm [Autodesk Fusion 360](#) bietet eine einfache Umsetzung auch für Personen, welche keine Ausbildung im Bereich der Mechanik und Entwicklung vorweisen können.

Bei der initialen Planung wurde beachtet, einen möglichst kleinen Fußabdruck des Schachttischs zu realisieren. Darüber hinaus wurde beabsichtigt, eine fertige Schachttischplatte als Basis zu verwenden und die Mechanik unter diese zu konstruieren. Um dies zu ermöglichen, wurde ein IKEA Lack Tisch verwendet, welcher die idealen Abmessungen von 55x55cm hat und somit eine erforderliche Schachtfeldgröße von 55mm ermöglicht. Durch den bereits vorhandenen Rahmen ist es auf einfache Art möglich, weitere Komponenten an diesem zu befestigen. Somit stellt diese Tischplatte eine ideale Basis für den autonomen Schachttisch dar.

Für die Achsenführung der beiden X- und Y-Achsen wurden konventionelle 20x20mm V-Slot Aluminium-Profile verwendet, welche mit einfachen Mitteln und wenig Geschick passend zugeschnitten werden können. Allgemein wurde eine X-Y Riemenführung verwendet, wobei jede Achse einen separaten [Nema 17](#) Schrittmotor inklusive des passenden Endschalters montiert hat. Bei den Schlitten, welche auf den Aluminium-Profilen laufen, wurden fertige Standardkomponenten verwendet, um das Spiel in der Mechanik zu minimieren. Diese stellen jedoch einen großen Posten in der Preiskalkulation dar. Die Vorteile überwogen jedoch, da diese nicht manuell erstellt und getestet werden mussten.

Bereits während des Designprozesses konnte anhand einer statischen Simulation des Modells erkannt werden, dass trotz der Optimierung des Fahrweges beider Achsen

## 5 Erstellung erster Prototyp



Bild 5-1: Prototyp Hardware: Erster Prototyp des autonomen Schachtisches

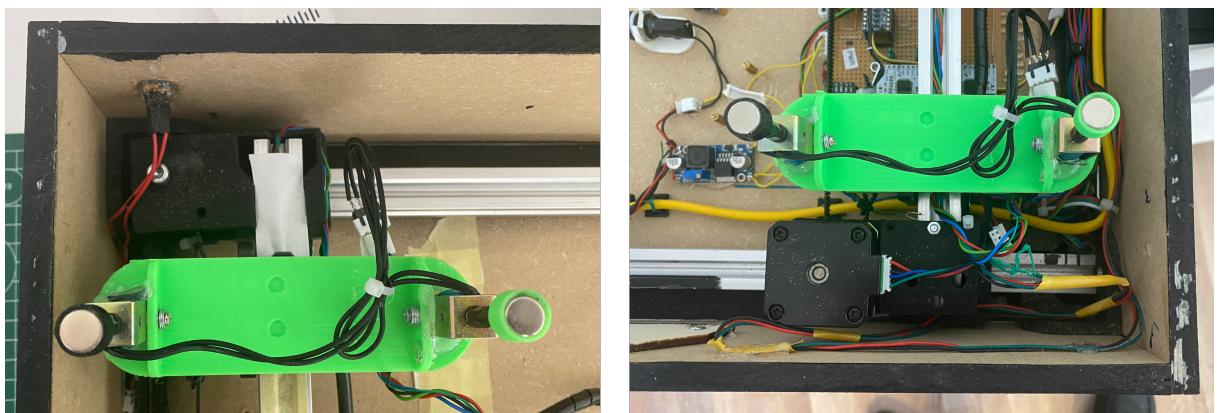


Bild 5-2: Zwei Elektromagnete. Schlitten befindet sich jeweils in den Ecken

durch die Verkleinerung der Halterungen der Aluminium-Profile die Gesamtausdehnung der Fahrwege nicht ausreichte. Mit dieser Konstellation konnten die Figuren nicht ausreichend weit aus dem Spielfeld platziert werden und verblieben in den äußeren Spielfeldern. Dieser Effekt war unerwünscht und schränkte das Spielerlebnis deutlich ein.

Um dies zu verhindern, wurde der zentrale Schlitten der Y-Achse, auf welchem der Elektromagnet für die Figur-Mitnahme platziert ist, um einen weiteren Elektromagneten erweitert. Dieser befindet sich nun nicht mehr mittig auf dem Schlitten, sondern wurde um 110mm in Richtung der X-Achse versetzt 5-2. So war es möglich, Figuren bis ganz an den Rand verschieben zu können.

Diese Lösung erfordert jedoch einen komplexeren Bahnplanungs-Algorithmus, da die Elektromagneten zwischen einzelnen Zügen gewechselt werden müssen. Dies führt zu einem kurzzeitigen Stillstand der Figur auf dem Schachfeld im Rahmen des Verschie-

bens der Figur an den Spielfeldrand.

Alle selbst-konstruierten Teile wurden anschließend mittels 3D Druck erstellt und konnten in die Tischplattenbasis eingeschraubt werden. Die Verwendung der aus Holz bestehenden Grundplatte erschwerte jedoch eine akkurate Platzierung der Teile und die bereits existierenden Seitenwände schränkten diese noch zusätzlich ein. Somit erforderte der komplette Zusammenbau mehrere Tage und zusätzliche Iterationen des 3D-Designs, um den Einbau spezifischer Teile zu ermöglichen. Das Design stellte damit jedoch eine solide Grundlage dar, welche für die weitere Software- und Hardware-Entwicklung essentiell ist.

## 5.2 Parametrisierung der Schachfiguren

Da das System die auf dem Feld befindlichen Schachfiguren anhand von NFC Tags erkennt, müssen diese zuerst mit Daten beschrieben werden. Die verwendeten NXP [NTAG 21](#)[21] Integrated Circuit (IC) besitzen einen vom Benutzer verwendbaren Speicher von 180 Bytes. Dieser kann über ein NFC-Lese/Schreibgerät mit Daten verschiedenster Art beschrieben und wieder ausgelesen werden. Moderne Mobiltelefone besitzen in der Regel auch die Fähigkeit, mit passenden NFC Tags kommunizieren zu können. Somit sind keine Stand-Alone Lesegeräte mehr notwendig.

Der Schachtisch verwendet dabei das NFC Data Exchange Format (NDEF) Dateiformat, welches festlegt, wie die Daten auf dem NFC Tag gespeichert werden. Da dieses ein standardisiertes Format ist, können alle gängigen Lesegeräte und Chipsätze diese Datensätze lesen. Der im autonomen Schachtisch verwendete Chipsatz [PN532](#) von NXP ist dazu ebenfalls in der Lage.

Um das NDEF Format verwenden zu können, müssen die NFC Tags zuerst entsprechend formatiert werden. Die meisten käuflichen Tags sind bereits derart formatiert. Alternativ kann dies mittels Mobiltelefon und passender Applikation geschehen. Da NDEF eigene Verwaltungsinformationen über die Art der Formatierung und über die gespeicherten Einträge speichert, stehen nach der Formatierung nur noch 137 Bytes des NXP NTAG 21 für die Speicherung von Nutzdaten zur Verfügung.

Per Lesegerät können anschließend mehrere NDEF Records auf den Tag geschrieben werden. Diese sind mit Dateien auf einer Festplatte vergleichbar und können verschiedene Dateiformate und Dateigrößen annehmen. Ein typischer Anwendungsfall ist der

SETTINGS		
FIGURE TYPE		
<input type="radio"/> KING <input type="radio"/> QUEEN <input type="radio"/> ROOK <input type="radio"/> BISHOP <input type="radio"/> KNIGHT <input checked="" type="radio"/> PAWN		
FIGURE COLOR		
<input checked="" type="radio"/> BLACK <input type="radio"/> WHITE		
RESULT		
No	DATA	NDEF_RECORD_CONTENT
0	1,1,0,1,0,0,0,0 =[208]	=D=
1	1,1,0,1,0,0,0,1 =[209]	=Ñ=
2	1,1,0,1,0,0,1,0 =[210]	=Ò=
3	1,1,0,1,0,0,1,1 =[211]	=Ó=
4	1,1,0,1,0,1,0,0 =[212]	=Ô=
5	1,1,0,1,0,1,0,1 =[213]	=Õ=
6	1,1,0,1,0,1,1,0 =[214]	=Ö=
7	1,1,0,1,0,1,1,1 =[215]	=×=

**Bild 5-3:** Prototyp Hardware: Tool zur Erstellung des NDEF Payloads: ChessFigureID-Generator.html

NDEF Record Type Definition (NDEF-RTD) URL Datensatz. Dieser kann dazu genutzt werden, eine spezifizierte URL auf dem Endgerät aufzurufen, nachdem der NFC Tag gescannt wurde. [26]

Der autonome Schachtisch verwendet den einfachsten NDEF-RTD Typ, den sogenannten Text-Record, welcher zum Speichern von Zeichenketten genutzt werden kann, ohne dass eine Aktion auf dem Endgerät ausgeführt wird. Jeder Tag einer Schachfigur, welche für den autonomen Schachtisch verwendet werden kann, besitzt diesen NDEF Record 5-4 an der ersten Speicher-Position. Alle weiteren eventuell vorhandenen Records werden vom Tisch ignoriert. [25]

Um die Payload für den NFC Record zu erstellen, wurde ein kleine Web-Applikation 5-3 erstellt, welche den Inhalt der Text-Records erstellt. Dieser ist für jede Figur individuell und enthält den Figur-Typ und die Figur-Farbe. Das Tool unterstützt auch das Speichern weiterer Attribute wie einem Figur-Index, welcher aber in der finalen Software-Version nicht genutzt wird.

Nach dem Beschreiben eines NFC Tags ist es zusätzlich möglich, diesen gegen Auslesen mittels einer Read/Write-Protection zu schützen. Diese Funktionalität wird jedoch nicht verwendet, um das Kopieren einzelner Figuren durch den Benutzer zu ermöglichen. Somit kann dieser leicht seine eigenen Figuren erschaffen, ohne auf das Tool angewiesen zu sein. Auch ist es so möglich, verschiedene Figur-Sets zu mischen;

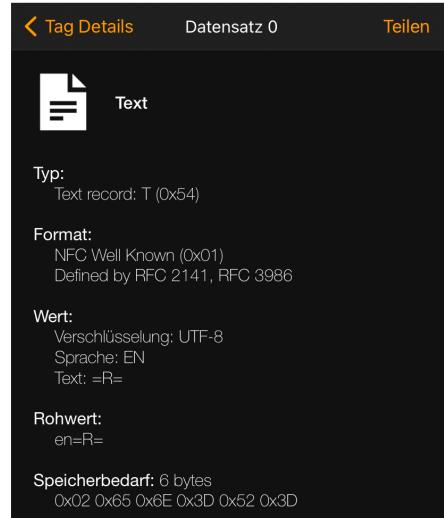


Bild 5-4: Prototyp Hardware: NDEF Text Record Payload für einen weißen Turm

somit kann ein Spieler verschiedene Sets an Figuren mit dem autonomen Schachtisch verwenden.

### 5.3 Schaltungsentwurf

Durch die zuvor durchgeführte Validierung der verwendeten Technologien konnte ein Blockdiagramm 5-5 der verwendeten elektrischen Komponenten angefertigt werden. Dieses enthält zum einen die zwei Schrittmotor-Treiber und zum anderen die Komponenten zur Ansteuerung der beiden Elektromagnete sowie das PN532 Modul zum Auslesen der NFC Tags.

Die wichtigsten Komponenten in der Schaltung sind das eingebettete System und die beiden Schrittmotortreiber TMC5160-B0B. Diese sind direkt über einen SPI Bus miteinander verbunden. Zusätzlich zu den Schrittmotoren selbst ist an jedem Treiber der Endschalter zur Durchführung der Referenzfahrt der Achse angeschlossen. Die Treiber bieten dabei Eingänge für zwei Endschalter, jedoch wird nur ein Endschalter für die minimale Position (Home Position) benötigt. Die Treiber sind direkt mit der Eingangsspannung verbunden, werden jedoch durch eine 5A Glassicherung geschützt. Da der SPI Bus und die Treiber mit dem 3.3V Logikpegel des eingebetteten Systems kompatibel sind, können diese direkt miteinander verbunden werden. Dieser Bus ist in einer Stern-Konfiguration aufgebaut, was zur Folge hat, dass jeder Treiber ein zusätzliches

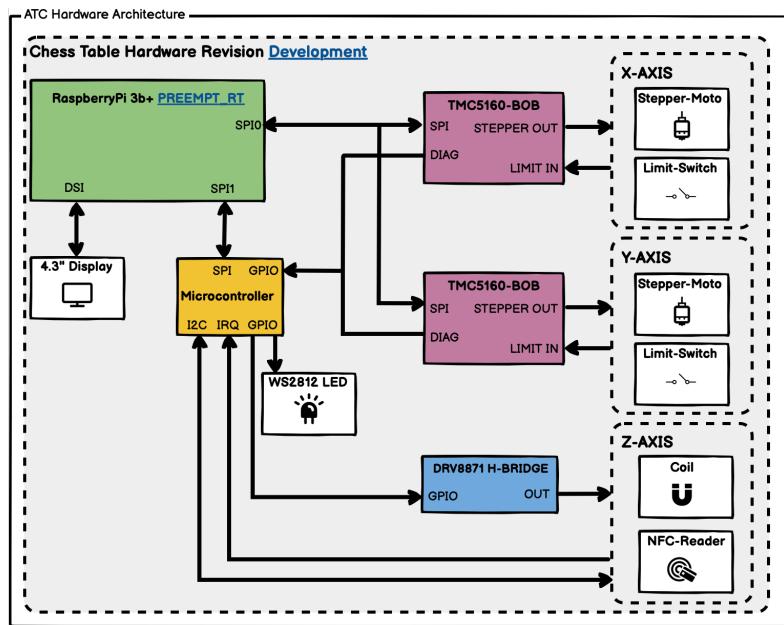


Bild 5-5: Prototyp Hardware: Blockdiagramm

Chip-Select Signal benötigt. Diese wurden ebenfalls mit dem eingebetteten System verbunden.

Zusätzlich sind Spannungswandler nötig, um die erforderlichen Spannungen von 12V für die Elektromagnete und 5V für das eingebettete System zu erzeugen. Die Schrittmotoren werden direkt mit der Versorgungsspannung von 14-24V betrieben. Alle weiteren verwendeten Komponenten, zu denen unter anderem auch das [PN532](#) NFC Modul und die [WS2811](#) LED Module gehören, werden ebenfalls über die 5V Schiene versorgt.

Für den Betrieb der beiden Elektromagnete wurde kein N-Channel Mosfet o.ä. verwendet, da hier maximale Flexibilität bei der Ansteuerung ausschlaggebend ist und bisher nicht ausreichend Erfahrung mit dem Verhalten dieser im Zusammenspiel mit den magnetischen Schachfiguren gesammelt werden konnte. Deshalb wurde hier eine H-Brücke [DRV8871H](#) verwendet. Somit kann auch die Polarität im Nachhinein per Software geändert werden und nicht nur die Spannung über ein Pulse Width Modulation (PWM) Signal. Der verwendete Treiber besitzt darüber hinaus zwei Ausgänge, was den Nutzen dieser Module besonders ausweitet.

Für die Erzeugung der PWM Signale für die H-Brücke wurde ein zusätzlicher Mikrokontroller [Atmega328p](#) benötigt, da hier die Steuersignale nicht direkt vom eingebetteten System erzeugt werden sollen, sondern nur die Zustandsinformationen über den SPI Bus übertragen werden sollen. Dies spart zusätzliche GPIO Anschlüsse und somit sind alle Komponenten über einen einzigen zentralen Bus kontrollierbar, welches einen möglichen Tausch des eingebetteten Systems in späteren Revisionen vereinfacht.

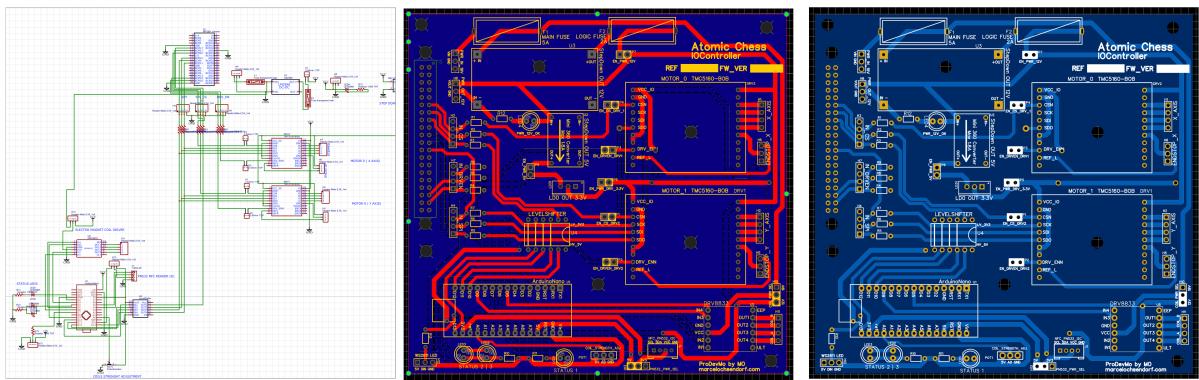


Bild 5-6: Prototyp Hardware: Schaltplan und finaler PCB Entwurf

Der zusätzliche Mikrocontroller übernimmt auch die Kommunikation mit dem [PN532](#) Modul, da dieses sonst über seine I2C Schnittstelle mit einem entsprechenden Host-System kommunizieren müsste. Der Mikrocontroller übernimmt somit ebenfalls die Konversion des I2C Bus hin zum zentralen SPI Bus. Zu beachten ist, dass nun ein zusätzlicher Chip-Select GPIO zum Ansteuern der Elektromagnete und des [PN532](#) Moduls benötigt wird. Dies wird durch die Firmware, welche auf dem Mikrocontroller ausgeführt wird, realisiert, die je nach empfangenem Kommando die entsprechende Komponente ausgewählt.

Nach der Festlegung der zu verwendenden Komponenten wurde ein entsprechender Schaltplan 5-6 nach den zuvor erörterten Vorgaben entworfen. Hierbei wurden die Vorgaben der Datenblätter[29][2] und der Application Notes [20][14] in diesen integriert. Da es sich hier um einen ersten Funktionsentwurf handelte, wurde zusätzliche Testpunkte in das Design eingefügt.

Somit war es während der weiteren Entwicklung möglich, zusätzliches Testequipment wie einen Logic-Analyser direkt an den SPI Bus oder ein Oszilloskop an die Ausgänge der H-Brücke dauerhaft anzuschliessen. Des Weiteren war es möglich, die Bus- und Spannungsversorgung über Jumper zu trennen, um einen Funktionstest einzelner Komponenten durchführen zu können.

Allgemein verwenden alle Komponenten 3.3V als Logik-Pegel. Trotzdem wurde ein Level-Shifter eingesetzt, welcher den SPI Bus des eingebetteten Systems von dem der Mikrocontroller trennt.

Durchgeführte Tests mit dem verwendeten [Atmega328p](#) ergaben, dass dieser nicht direkt mit 3.3V und einer Taktfrequenz von 16MHz betrieben werden kann und es somit zu einem nicht kontrollierbaren Verhalten dieses kommt. Dieses Verhalten machte sich durch eine gestörte Kommunikation mit dem [PN532](#) Modul bemerkbar und ein Auslesen

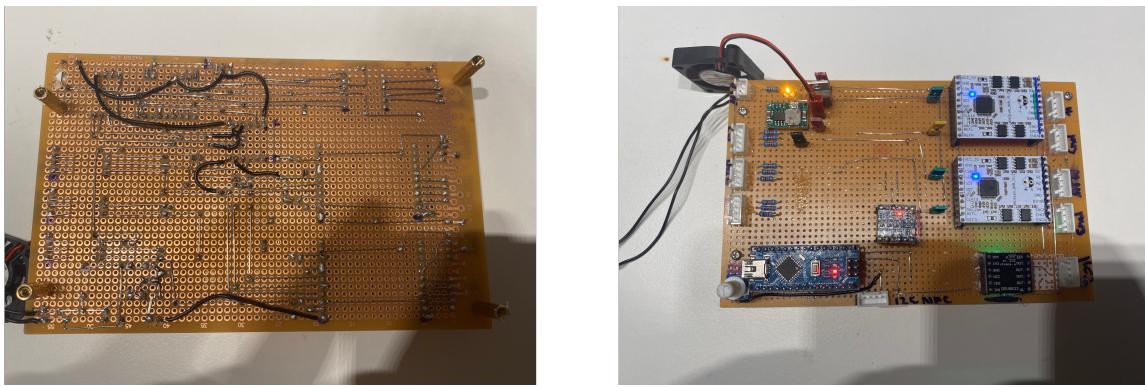


Bild 5-7: Prototyp Hardware: Aufbau der Lochrasterplatine

von NFC Tags war nur in 60% der Fälle fehlerfrei möglich.

Im Anschluss wurde die Versorgungsspannung auf 5V erhöht, was zur Folge hatte, dass die Ein- und Ausgänge ebenfalls mit diesem Pegel arbeiteten; der Einsatz des zusätzlichen Level-Shifter wurde zum Schutz des eingebetteten Systems und dessen GPIO Schnittstelle notwendig.

Der Schalplan und dessen Funktionalität wurden anschließend durch den Aufbau der vollständigen Schaltung auf einer Lochrasterplatine 5-7 im Eurokartenformat manuell aufgebaut und getestet.

Aus diesem Design wurde ein Printed Circuit Board (PCB) Layout für eine einfache 2-lagige Platine erstellt. Dieses orientierte sich an der zuvor umgesetzten Lochrasterplatine und spiegelte deren Layout wider. Auch hier wurde nicht auf den Platzverbrauch geachtet. Es wurden zusätzliche Steckverbindungen für die externen Komponenten eingefügt und passende Bohrungen an den Ecken sowie in der Mitte zur Montage vorgesehen. Auf dem obersten Layer wurde der Bestückungsdruck erhöht und mit zusätzlicher Information über die Pin-Belegungen der einzelnen Stecker erweitert.

## 5.4 Implementierung HAL

Der HAL stellt das Verbindungsglied zwischen der Hardware und der Benutzer-Software dar. In diesem Fall übernimmt er die Übersetzung der Befehle der Controller-Software in für die Hardware verständliche Befehle. Dabei geschieht dies über den zentralen SPI Bus, welcher im Linux-System als Datei unter dem Pfad `/dev/spidev0.0` eingebunden wird und über Dateioperation (lesen, schreiben) mittels `ioctl` konfiguriert werden kann.

## 5 Erstellung erster Prototyp

---

Weiterhin können Daten über das File-Handle gelesen und geschrieben werden. Somit ist eine Kommunikation mit der Hardware-Ebene möglich.

Diese Funktionalität wird von dem für dieses Projekt implementierten HAL in Form einer C++ Klasse abgebildet und ermöglicht einen einfachen Zugriff auf die elektrisch verbundenen Komponenten. Zusätzlich wird in dieser Klasse auch das Hardware-Versions-Management abgebildet.

Da im Verlauf der Entwicklung mehrere Hardware-Revisionen gebaut wurden und die Controller-Software weiterhin mit allen Revisionen kompatibel sein soll, ermittelt der HAL vor dem Start die entsprechende Revision. Dazu wird die Prozessor-Identifikator (ID) (welche mittels des `cat /proc/cpuinfo | grep Serial | cut -d ',' -f 2` Kommandos abgefragt werden kann) des Systems abgefragt und mittels einer statischen Liste die entsprechende Revision ermittelt. Hierbei enthält die Tabelle nur Revisionsinformationen über die während der Entwicklung entstandenen Revisionen. Sollte die Prozessor-ID nicht hinterlegt sein, geht das System von der aktuellsten Revision aus. So ist keine manuelle Pflege der Tabelle während einer möglichen Produktion notwendig.

```
1 //HardwareInterface.h
2 //...
3 class HardwareInterface
4 {
5     enum HI_HARDWARE_REVISION {
6         HI_HWREV_UNKNOWN = 0,
7         HI_HWREV_DK      = 1, //FIRST 55x55cm ATC TABLE WITH TWO
                           COILS
8         HI_HWREV_PROD    = 2, //SECONDS GENERATION BASED ON SKR1
                           .3 3D PRINT CONTROLLER
9         HI_HWREV_PROD_V2 =3, //THIRD GENERATION WITH SKR 1.4
                           WITH CORE XY MECHANIC
10        HI_HWREV_VIRT=4, //SIMULATED HW FOR TESTING USING THE
                           DOCKERFILE
11    };
12
13    enum HI_COIL
14    {
15        HI_COIL_A      = 0,
16        HI_COIL_B      = 1,
17        HI_COIL_NFC   = 2
18    };
19    //....
20    //MOTOR CONTROL FUNCTIONS
21    void enable_motors();
22    void disable_motors();
23    bool is_target_position_reached();
24    void move_to_postion_mm_absolute(const int _x, const int _y,
                           const bool _blocking);
```

## 5 Erstellung erster Prototyp

---

```
25 void home_sync();
26 //...
27 //LED CONTROL FUNCTIONS
28 bool setTurnStateLight(const HI_TURN_STATE_LIGHT _state);
29 //NFC CONTROL FUNCTIONS
30 ChessPiece::FIGURE ScanNFC();
31 //MAGNET CONTROL FUNCTIONS
32 bool setCoilState(const HI_COIL _coil, const bool _state);
33 //...
```

Je nach ermittelter Revision werden die erforderlichen Hardwarekomponenten initialisiert. Bei allen über den SPI Bus angeschlossenen Komponenten werden nach der Initialisierung des SPI Bus auf der Betriebssystem-Ebene zusätzliche Versionsregister der einzelnen Komponenten abgefragt. Dies stellt sicher, dass alle Komponenten mit dem System verbunden sind.

Allgemein kann eine Datentransfer über den SPI drei Mal fehlschlagen, bevor die Software mittels eines Fehlers abbricht. Gerade bei der Kommunikation mit dem Mikrokontroller kam es bei Testläufen zu Fehlern bezüglich der SPI Kommunikation, sofern das NFC-Modul aktiv war. Um ein direktes Beenden der Software zu verhindern, wurde diese Art der Fehlerbehandlung eingeführt.

```
1 //SPICommunications.cpp
2 //...
3 int SPICommunication::spi_write_ack(SPICommunication::
4                                     SPI_DEVICE _device, uint8_t* _data, int _len)
5 {
6     uint8_t* buffer_r{ new uint8_t[_len] { 0 } };
7     uint8_t* buffer_w{ new uint8_t[_len] { 0 } };
8
9     volatile int res = -1;
10    volatile int c = 0; //RETRY COUNTER
11    while (true)
12    {
13        //RECREATE COMMAND BUFFER
14        //WILL BE OVERWRITTEN AFTER spi_write / spi_read
15        for (size_t i = 0; i < _len; i++)
16        {
17            buffer_w[i] = _data[i];
18            buffer_r[i] = 0;
19        }
20
21        //WRITE COMMAND
22        res = SPICommunication::getInstance()->spi_write(
23            _device, buffer_w, _len);
24        //WAIT
25        std::this_thread::sleep_for(std::chrono::milliseconds(
26            SPI_RW_DELAY));
```

```
24     //READ RESULT BACK
25     res = SPICommunication::getInstance() -> spi_write(
26         _device, buffer_r, _len);
27     //PARSE RESULT; CHECK FOR READ SUCCESS
28     if(buffer_r[0] == MAGIC_ACK_BYT)
29     {
30         break;
31     }
32     //INCREASE ERROR COUNTER
33     c++;
34     if (c > SPI_RW_ACK_RETRY)
35     {
36         break;
37     }
38     return res;
39 }
40 //...
```

Der HAL und dessen benötigte Softwarekomponenten zur Buskommunikation und Hardware-Revisionsbestimmung wurden für die Verwendung innerhalb von mehreren Threads angepasst und somit ist deren Verwendung threadsafe. Diese Optimierung wurde jedoch nicht verwendet, da jegliche Funktionsaufrufe, welche die Hardware betreffen, aus dem Main Thread der Controller-Software getätigigt werden.

### 5.4.1 TMC5160 SPI Treiber

Der Treiber für die verwendeten TMC5160 Schrittmotor-Treiber ist ebenfalls ein Bestandteil des HAL. Die verwendeten Bausteine bieten mitunter sehr komplexe Konfigurationsmöglichkeiten und je nach Betriebsart sind mehrere Lese- und Schreiboperationen über den SPI Bus notwendig. Diesbezüglich wurde die komplette Ansteuerung auf der Softwareseite in ein eigenes Modul geschachtelt. Dieses stellt verschiedene Funktionen zum Verfahren eines Motors bereit. Hierzu benötigt jeder verwendete Hardware-Treiber eine Instanz des Moduls zur Ansteuerung; so ist es zusätzlich möglich, für jede Achse verschiedene Parameter 5.1 setzen zu können in Bezug auf Beschleunigung und Positioniergeschwindigkeit des Motors.

Tabelle 5.1: TMC5160 Beschleunigungskurve / RAMP Parameter

Parameter	Value
V_START	1
A1	25000
V1	250000
A_MAX	5000
V_MAX	1000000
D_MAX	5000
D1	50000
V_STOP	10

Der Treiber unterstützt dabei zwei verschiedene Funktionsmodi:

- Position-Mode
- Velocity-Mode

Der Treiber wird hierbei nur im [Position-Mode](#) betrieben, da hier eine Ziel-Position für den Motor vorgegeben werden kann. Hierbei kann über ein Register eine Zielposition in Schritten vorgegeben werden. Der Treiber ermittelt daraufhin die passende Beschleunigungskurve und verfährt den Motor an diese Position. Über ein entsprechendes Register kann der Status der Operation abgefragt werden und ob der Motor seine Position erreicht hat bzw. ob Fehler auftraten. Somit muss nicht auf das Erreichen der Zielposition gewartet werden und andere Aufgaben können währenddessen ausgeführt werden. Die Beschleunigungskurve kann zusätzlich manuell angepasst werden. Hier wurden jedoch die Standardwerte aus dem Datenblatt verwendet, welche sich bei mehreren Tests als optimal im Bezug auf Geräuschemission des Motors herausstellten.

## 5 Erstellung erster Prototyp

---

Beim dem [Velocity-Mode](#) hingegen, kann der Motor in einer definierten Geschwindigkeit in eine Richtung verfahren werden. Dabei kann diese Bewegung in einem Dauerlauf durchgeführt werden, welches im [Position-Mode](#) nicht geht. Hierbei wird dies durch den maximalen möglichen Wert des Ziel-Positions-Register bestimmt.

```
1 // /TMC5160.cpp
2 TMC5160::TMC5160(MOTOR_ID _id) {
3     //...
4     //CHECK SPI INIT
5     if (!SPICommunication::getInstance() -> isInitialised()) {/*...
6         */
7     //REGISTER SPI CS PIN FOR SELECTED MOTOR ID
8     if (_id == MOTOR_ID::MOTOR_0) {
9         const SPI_CS_DEVICE = SPICommunication::SPI_DEVICE::
10        MOTOR_0; //TODO CAST
11        SPICommunication::getInstance() -> register_cs_gpio(
12            SPI_CS_DEVICE, CS_GPIO_NUMBER_MOTOR_0);
13    }
14 }
15 //...
16 void TMC5160::default_settings()
17 {
18     // ENABLE STEALTH-CHOP
19     write(REGDEF_GCONF, 0x0000000C);
20     // SET SPREAD CYCLE PWM
21     write(REGDEF_CHOPCONF, 0x000100C3);
22     // SET MAX MOTOR CURRENT
23     write(REGDEF_IHOLD_IRUN, 0x00080F02);
24     // SET MOTOR AUTO POWER OF TO 10 SEC
25     write(REGDEF_TPOWERDOWN, 0x0000000A);
26     // SET MAX VELOCITY IN STEALTH-CHOP MODE
27     write(REGDEF_A1, 0x000001F4);
28     // SET RAMP PARAMETERS
29     reset_ramp_defaults();
30     // SET DRIVER STATE TO POSITION MODE
31     write(REGDEF_RAMPmode, 0);
32     // SET CURRENT POSITION TO 0
33     write(REGDEF_XACTUAL, 0);
34     // SET TARGET POSITION TO 0
35     write(REGDEF_XTARGET, 0);
36 }
37
38 int TMC5160::write(const int _address, const int _data)
39 {
40     const size_t DATA_LEN = 5;
41     //POPULATE WRITE DATA BUFFER
42     uint8_t write_buffer[] = { _address | 0x80, 0, 0, 0, 0 };
43     write_buffer[1] = 0xFF & (_data >> 24);
```

## 5 Erstellung erster Prototyp

```
44     write_buffer[2] = 0xFF & (_data >> 16);
45     write_buffer[3] = 0xFF & (_data >> 8);
46     write_buffer[4] = 0xFF & _data;
47     //WRITE DATA OVER SPI
48     return SPICommunication::getInstance()->spi_write(
49         SPI_CS_DEVICE ,write_buffer , DATA_LEN);
50 }
51 int TMC5160::read(const int _address)
52 {
53     //POPULATE WRITEBUFFER = READ REGISTER ADDRESS
54     const size_t DATA_LEN = 5;
55     uint8_t write_buffer[] = { _address & 0x7F , 0, 0, 0, 0 };
56     uint8_t read_buffer[] = { _address & 0x7F , 0, 0, 0, 0 };
57     //FIRST WRITE REGISTER ADRESS TO READ
58     int res = SPICommunication::getInstance()->spi_write(
59         SPI_CS_DEVICE ,write_buffer , DATA_LEN);
60     //READ RESULT
61     res = SPICommunication::getInstance()->spi_write(
62         SPI_CS_DEVICE , read_buffer , DATA_LEN);
63     //PARSE RESULT INTO INT
64     int value = read_buffer[1];
65     value = value << 8;
66     value |= read_buffer[2];
67     value = value << 8;
68     value |= read_buffer[3];
69     value = value << 8;
70     value |= read_buffer[4];
71     return value;
72 }
73 //EXAMPLE USAGE, GOTO POSITION
74 void TMC5160::go_to(const int _position) {
75     write(REGDEF_RAMPMODE , 0);
76     //SET XTARGET REGISTER = TARGET POSITION
77     //NON BLOCKING
78     write(REGDEF_XTARGET , _position);
79     //USE move_to_positon_mm_relative FOR A BLOCKING VARIANT
80 }
81 void TMC5160::atc_home_sync()
82 {
83     enable_motor(); //ENABLE MOTOR
84     enable_switch(TMC5160::REF_SWITCH::REF_L , true , true , true
85         ); //ENABLE LIMIT SWICHT => ENABLE HARD ENDSTOP
86     move_velocity(TMC5160::VELOCITY_DIRECTION::NEGATIVE ,
87         HOME_SPEED_VELOCITY , 1000); //MOVE NEGATIVE TO LIMIT
88     SWITCH
89     //WAIT TO REACH THE ENDSTOP
90     while(!get_ramp_stauts().status_stop_l) {
91         std::this_thread::sleep_for(std::chrono::microseconds
92             (1));
93 }
```

```
89     }
90     //STOP MOTOR
91     hold_mode();
92     //SAVE LATCHED POSITION
93     int offset = get_position() - get_latched_position();
94     write(REGDEF_XACTUAL, offset);
95     //SAVE OFFSET
96     int currpos = get_position();
97     set_postion_offset(currpos);
98     //RESET RAMP
99     write(REGDEF_RAMPSTAT, 4);
100    //GOTO THE NEW ZERO POSITION
101    set_AMAX(RAMP_AMAX);
102    set_VMAX(RAMP_VMAX);
103    go_to(0);
104    //DISABLE HARD ENDSTOP
105    enable_switch(TMC5160::REF_SWITCH::REF_L, true, false,
106                  true);
106    disable_motor(); //DISABLE MOTOR
107 }
108 //...
```

Eine zusätzliche Besonderheit stellt der Referenzfahrt dar. Nach dem Start des Systems ist es möglich, dass sich der Schlitten einer Achse nicht an der Null-Position befindet, sondern an einer unbekannten Position auf der Achse. Deswegen muss diese Achse zuerst an die Home-Position gefahren werden. Dazu besitzt das System zwei Endschalter, welche jeweils mit einem Schrittmotor-Treiber verbunden sind. Diese besitzen zwei solche Taster-Eingänge `REF_L/REF_R`.

Bei einer wechselnden Flanke an diesem Eingang kann der Motor-Treiber verschiedene Funktionen ausführen. In diesem Fall wurde die Motor-Stopp Funktion mittels Register-eintrag gewählt, welche den Motor stoppt, sobald der Schalter betätigt wird. Dies stellt schlussendlich die Home-Position dar. Dies kann jedoch nicht im `Position-Mode` des Treibers umgesetzt werden, da das Ziel-Positionsregister auf Null gesetzt wird. Hierzu muss der Treiber in den `Velocity-Modus` geschaltet werden, welches ein Verfahren des Motors in eine Richtung ohne Zeitbegrenzung erlaubt. Dies wird so lange in negativer Bewegungsrichtung ausgeführt, bis der Endschalter erreicht wurde. Somit ist die Achse an ihrer Home-Position angekommen und kann anschließend im Positions-Modus normal verfahren werden.

## 5.5 Fazit bezüglich des ersten Prototyps

Im Hinblick auf den Umsetzungsprozess des autonomen Schachtisches stellt die Fertigstellung des ersten Prototyps einen ersten großen Erfolg dar. Dennoch konnten nicht alle zuvor gestellten Requirements mit diesem Design umgesetzt werden.

Zu den Defiziten zählte zum einen der Bewegungsspielraum der einzelnen Achsen. Dieser wurde bereits während der Entwicklung durch die Verwendung von zwei Elektromagneten künstlich verlängert. Nach einem Langzeittest stellte sich jedoch diese Methode als zu fehleranfällig heraus. Die Parkpositionen, welche sich an den zwei Seiten des Spielbrettes befinden, konnten nicht durchgängig zuverlässig angefahren werden und boten nur Platz für 14 ausgeschiedene Figuren pro Spielerfarbe. Somit war ein komplettes Abräumen des Spielfeldes nicht möglich, auch wenn dieses in der Praxis eher selten vorkommt.

Zum anderen war der Aufbau und die anschließende Kalibrierung der Mechanik und der entsprechenden Offset-Werte in der Software nicht trivial und benötigte einiges an Zeit. Durch die Verwendung der Tischplatte und des hölzernen Grundrahmens konnte jedoch ein robustes Design in einem kleinen Formfaktor umgesetzt werden, welches zusätzlichen Platz für Erweiterungen bietet.

Gerade die Verwendung von verschraubten Holzplatten machte jedoch eine Vervielfältigung mit gleicher Qualität schwierig. Ein Re-Design der inneren Komponenten gestaltete sich schwierig, da hier bereits mehrere Iterationen durchgeführt wurden, um eine maximalen möglichen Verfahrweg zu ermöglichen.

Auf Seiten der Elektronik arbeitete diese eher zuverlässig und bereitete keinerlei Probleme. Jedoch stellten die verwendeten Motortreiber einen größeren Kostenfaktor dar und der Zeitaufwand für den Zusammenbau und Überprüfung dieser war substantiell. Die verwendeten Elektromagnete sind für 9V Betriebsspannung ausgelegt, mussten jedoch über ihren Spezifikationen mit 12V betrieben werden, was bei einem Dauerbetrieb zu stark erhöhten Temperaturen führte.

Allgemein war hier die Entscheidung, die Außenmaße des Tisches zu optimieren, nicht ideal und führte zu diversen Problemen. Diese konnten jedoch mit verschiedenen Workarounds behoben werden können. Ein Schachspiel ist mit diesen Prototypen mit Einschränkungen möglich und bildet bis auf das Fehlen der nicht funktionstüchtigen Parkpositionen die zuvor festgelegten Requirements ab.

## **5 Erstellung erster Prototyp**

---

Im weiteren Verlauf der Entwicklung stand jedoch die Verbesserung der Zuverlässigkeit und die fehlerfreie Umsetzung der Parkposition für ausgeschiedene Figuren an. Ein einfacherer Zusammenbau auch für Dritte sollte ebenfalls ins Auge gefasst werden.

Hierzu würde ein komplettes Re-Design der Mechanik sowie der Elektronik nötig sein. Anpassungen der Software würden dadurch ebenfalls nötig, stellten jedoch durch den modularen Aufbau dieser kein Hindernis dar. Die durch den ersten Prototypen gewonnenen Erkenntnisse konnten somit direkt in das neue Design einfließen.

# 6 Aufbau des zweiten Prototypen

## 6.1 Modifikation der Mechanik

### 6.1.1 Gehäuse und Design

Mit der Entscheidung, auf die hölzerne Struktur des Systems gänzlich zu verzichten, wurden massive Veränderungen des Designs des Schachtischs bestehend aus Gehäuse, Dimensionen und allen Außenelementen nötig.

Zuvor bestand der Quader-förmige Schachttisch aus einem Lack-Tisch als Deckel, welcher mit einem selbsterstellten Untergestell bestehend aus Rahmen und Boden verschraubt wurden. Nun musste der Quader selbst konstruiert werden.

Die Wahl des neuen Materials war jedoch simpel; aufgrund der langjährigen Bewährtheit, der Stabilität und der einfachen Möglichkeit der Anpassung wurden als Basis des neuen Systems Aluminium-Profilstangen gewählt. Da der Tisch keine größeren Kräfte aufnehmen muss, wurde ein Stangengrundmaß von 20 x 20 mm gewählt. Diese Stangen sind dennoch stabil genug, um möglichen Außeneinwirkungen wie Stößen oder Drücken standzuhalten.

Als Außenmaße wurden 620 x 620 x 170 mm (Länge, Breite, Höhe) gewählt. Das Außenmaß ergab sich aus der Berechnung der benötigten Spielfeldgröße, der Parkpositionen und der gegebenen Stangenbreite. Die Schachfiguren besitzen einen maximalen Durchmesser von 22 mm. Damit Figuren aneinander ohne Berührung vorbeigeführt werden können, ist somit eine Größe von mindestens 44 mm für ein Feld nötig. Da eine Distanz eingerechnet werden muss, um ein magnetisches Anziehen der Figuren zu verhindern und Fehler bei der mittigen Positionierung der Figuren möglich sind, wurde hierfür eine zusätzliche Toleranz von 13 mm ergänzt und somit ein Idealmaß von 57 mm Seitenlänge pro Feld errechnet. Bei einem vollständigen Schachttisch ergibt sich daraus



**Bild 6-1:** Production Hardware: Finaler autonomer Schachtisch

eine Schachfeldgröße von 456 x 456 mm. Für die Parkpositionen wurden zusätzlich noch einmal 30 mm berechnet mit einem Abstand zum Feld von weiteren 37 mm. Somit ergibt sich, wenn man das Feld quadratisch auslegt, eine Seitenlänge von 590 mm. Als Plattenmaß wurde 620mm gewählt, um eine Toleranz für die Befestigung zu berücksichtigen und zudem möglichen Einschränkungen der Mechanik vorzubeugen.

Die Platte wurde dann in Alu-Profilstangen eingelassen; die Stangen sollen die Platte umrahmen. Mit einem Grundmaß von 20 x 20 mm für die Profilstangen ergab sich somit ein Gesamtmaß von 660 mm x 660 mm. Benötigt wurden 8 Profilstangen der Länge 620 mm und 4 der Länge 170 mm, welche zu einem quadratischen Kasten zusammengesetzt wurden. Für die X-Achsen wurden zudem zwei Profilstangen der Länge 610 mm und für die Y-Achse noch eine weitere der Länge 620 mm benötigt.

Die für die Montage üblicherweise verwendeten Winkel wurden jedoch aufgrund der Größe nicht verwendet. Es wurden eigene Winkel mittels 3D-Druck erstellt oder auch Komponenten zur Befestigung direkt als Winkel-Elemente integriert. So wird der obere Quadrant des Korpus von 4 Winkeln gehalten, welche zum einen als Auflage für die Tischplatte fungieren und zum anderen das Befestigen der beiden X-Achsen Profilstangen ermöglichen.

Da die Tischplatte nur aufliegt, ist es zusätzlich möglich, den Raum unterhalb der Profilstangen als Konstruktionsraum zu verwenden. Dabei ist zu beachten, dass an allen Seiten des Tisches noch Seitenelemente, bestehend aus 620 x 130 x 5 mm Acrylglas-Platten, eingelassen werden. Somit beträgt die exakte Länge der für die Konstruktion nutzbaren Seiten 650 mm. Lediglich die Ecken, welche die Höhenelemente der Aluprofile beinhalten, bieten nur eine Länge von 620 mm.

Um das Design optisch zu verbessern, wurden die Acrylglas-Elemente in weiß gewählt. Transparente Elemente ermöglichen zwar eine Sicht auf die Mechanik im Inneren, jedoch wurde hierbei insbesondere Wert auf den Gesamteindruck gelegt, welcher durch die weiße Struktur des Glases aufgewertet wird. Zudem wurden im Inneren noch zusätzlich LED-Streifen verlegt, welche dank des verwendeten TMC-Boards einfach angeschlossen werden konnten. Die weißen Glaselemente streuen das Licht günstiger und ermöglichen so ein unterschwelliges Leuchten.

Um das System vollständig zu verschließen und somit auch besser zu schützen, wurde zudem eine Bodenplatte mit identischen Maßen zur Tischplatte eingelassen und mit Stützelementen verschraubt.

Nachteil der verwendeten Aluminium-Profilstangen und der weißen Acrylglas-Elemente sind die höheren Kosten und der Aufwand der Montage. Der verwendete Lack-Tisch und auch das selbsterstellte Untergestell als solches waren preisgünstig und leicht erhältlich. Zudem ist die Tischplatte, welche aus einem einzelnen Tisch bestand, stabil und robust. Aluminiumstangen hingegen, die zusätzlich noch bestellt, selbstständig an die gewünschte Länge angepasst und mit weiteren Komponenten verschraubt werden müssen, sind dementsprechend deutlich kostenintensiver.

Zudem wurde die Tischplatte nun durch eine simple Holzplatte ersetzt. Eine Höhe von 3 mm darf aufgrund des Magnetismus zwischen Schlitten und Schachfigur nicht überschritten werden. Um ein Durchbiegen der Platte zu verhindern, wurden in den Profilstangen horizontale Vorsprünge ergänzt, die die Platte auf einer Ebene mit der Oberkante der Alu-Profilstangen halten.

Die Beine des zuvor verwendeten Lack-Tisches wurden erneut verwendet. Diese konnten für die zweite Revision verwendet werden und so zusätzlich die gleiche Montagehöhe zwischen der ersten und der zweiten Revision des Tisches erreicht werden. Da selbst die Höhe der Quader der Schachtisch-Revisionen identisch ist, sind beide Tische nun gleich hoch. Alternative Lösungen wären der Erwerb von simplen Hohlleisten der gleichen Länge oder aber das Integrieren weiterer Profilstangen, welche man optimalerweise auch klappbar lagern könnte. Derzeit sind die Beine verschraubt und nicht klappbar. Der daraus resultierende Nachteil der Tischbeine ist, dass man den gesamten Tisch nun schwerlich auf einen anderen Tisch stellen kann, um die Montage zu erleichtern oder ein Schachspiel auf einer anderen, eventuell bequemeren Höhe durchzuführen. Der Tisch benötigt allerdings auch keine Unterlage mehr und kann ohne Probleme im offenen Raum platziert werden. Die aktuell verwendeten Beine können je nach Bedarf auch entfernt werden, sodass der Schachtisch wieder als simpler Quader einfach zu

handhaben ist.

Insgesamt überwiegen die Vorteile der Universalität dank der gegebenen Normungen, der einfachen Anpassungsmöglichkeiten in der Länge und dank des einfachen Ergänzens und Verschiebens von Komponenten. Im Holzrahmen verschraubte Elemente hinterlassen Löcher, die zu Beeinträchtigungen führen können. Zudem ist das Ergänzen von anderen Komponenten im Aluminium-Profil einfacher.

### 6.1.2 3D-Komponenten

Die Masse an selbst-erstellten 3D-Komponenten wurde aufgrund des selbst erstellten Korpus in der zweiten Revision erhöht. Aluprofile bieten die Möglichkeit der einfachen Montage von zusätzlichen Komponenten. Mittels sogenannter Nutensteine, welche in das Profil geschoben werden, ist es möglich, diese Komponenten einfach an das Profil zu schrauben, indem man in das im Nutenstein befindliche Gewinde schraubt. Dank der Schienen-ähnlichen Gestaltung der Profile sind die Positionen dieser Nutensteine individuell anpassbar. Ausgehend von den gewählten Maßen der Stangen wurden Nutensteine vom „Typ 6“ mit einem M5 Gewinde gewählt.

Zudem wurde nur ein einziges 3D-Design angefertigt, indem eine simple Platte erstellt wurde, auf welcher zwei Vorsprünge extrudiert wurden und eine Durchführung des Durchmessers 6,5 mm. Mittels der Durchführung konnte die Platte mit einem Nutenstein verschraubt werden, mittels der Vorsprünge, welche in die Profilschienen ragen, wird ein Drehen der Platte verhindert. Dieses 3D-Design wurde im Folgenden als Grundlage für alle neuen Komponenten verwendet. Oftmals wurden bestehende Modelle der ersten Revision mit diesem neuen Design verbunden und als eine Komponente gedruckt, was eine Wiedernutzung von etablierten Komponenten ermöglicht.

Der Grundrahmen als solcher wurde einmalig erstellt und nur in einfachen Strukturen wie der Winkel-Sehnenlängen oder Höhenparametern von Flächen angepasst. Alle weiteren Komponenten im Inneren bedurften mehrerer Revisionen, um die verwendete Mechanik optimal umzusetzen und mehr Kräfte für Riemenspannungen zu ermöglichen.

Insgesamt benötigt der gesamte Schachttisch 26 verschiedene mittels FDM 3D-gedruckte Komponenten, durch Mehrfachnutzung werden insgesamt 75 Elemente gedruckt. Dies ergibt bei den verwendeten Drucken und dem gewählten Filament eine Druckzeit von 25 Stunden und rund 450 Gramm Filament.

Zusätzlich zu diesen Komponenten ist es möglich, 32 Schachfiguren mittels SLA 3D-Druck zu erzeugen.

### 6.1.3 Positions-Mechanik

Die Mechanik des ersten Prototypen wurde für die Erstellung des zweiten Prototypen gänzlich verändert.

In der ersten Revision wurde noch jede Achse über einen separaten Riemen gesteuert, sodass ein Schrittmotor die Bewegung des Schlittens entlang der Y-Achse und ein weiterer die Bewegung der gesamten Y-Achse, bestehend aus Motor, Riemen, Schlitten und Führungsschiene, entlang der X-Achse ermöglichte. Die Führung entlang der X-Achse erfolgte in der Mitte des Tischs, die Y-Achse wurde links und rechtsseitig rollbar gelagert und in der Mitte über einen Riemen gezogen. Dies hatte zur Folge, dass bei entstehender Unwucht, welche durch die Bewegung des Schlittens auf der Y-Achse auftreten kann, die Y-Achse in ihren Lagerungen nicht mehr parallel verlief, sondern beim Betätigen des Motors der X-Achse die Y-Achse in einem unerwünschten Winkel bewegt wurde.

Die Konsequenz dessen war, dass die Schachfiguren nicht mehr rein parallel zu X oder Y-Achse bewegt werden konnten, sondern immer ein unvorteilhafter Winkel in den Bewegungsablauf integriert wurde. Das hatte wiederum zur Folge, dass Figuren nicht richtig positioniert wurden oder zu dicht an unbewegten Figuren vorbeigeführt wurden.

Eine Lösungsmöglichkeit wäre die Ergänzung eines zweiten Motors für die X-Achse gewesen, so dass linksseitig und rechtsseitig unmittelbar an der Lagerung gezogen werden könnte. Dies erwies sich jedoch als unpraktikabel und hätte einen zusätzlichen Kostenfaktor dargestellt. Ein stabileres Befestigen der Y-Achse in ihren Lagerungen hätte zur Folge gehabt, dass der Widerstand der Lagerungen zugenommen hätte und die Bewegung der Achse nur unter erhöhten Kräften möglich gewesen wäre.

Deswegen wurde ein völlig anderes System für die zweite Revision des Schachtisches gewählt, welches auf beide Achsen die gleichen Kräfte ausübt und beide Achsen nicht mehr unabhängig voneinander betrachtet.

CoreXY basiert auf der Idee alter Zeichentische und wird heute für verschiedene Anwendungen wie den 3D-Druck oder das Computerized Numerical Control (CNC)-Fr

genutzt, bei dem ein Objekt oder Werkzeug in mehreren Dimensionen bewegt werden soll.

Ein Objekt wird von zwei Enden desselben gespannten Riemens gehalten; wenn ein Ende des Riemens kürzer wird, wird das andere Ende länger. Der Riemen wird über Lagerungen in den gegenüberliegenden Ecken eines Rechtecks so angeordnet, dass die Bewegung eines Motors eine Bewegung des Werkzeugs in einem 45-Grad-Winkel bewirkt. Werden nun zwei Riemen in das System gebracht und alle vier Enden an dem Objekt befestigt, so ist es möglich, durch das Bewegen eines Riemens mittels einer Bewegung des anderen Riemens den 45-Grad Winkel zu einer geraden Strecke zu glätten. Das bedeutet, dass man beide Motoren bewegen muss, um in einer geraden Linie zu fahren.

Einer der größten Vorteile des CoreXY-Systems ist die hohe Bewegungsgeschwindigkeit. Dies ist insbesondere dadurch möglich, dass es keine beweglichen Teile von nennenswerter Masse gibt. In der ersten Revision wurden beim Bewegen der X-Achse alle Komponenten bewegt, zu denen auch der höher gewichtige Schrittmotor zählte. Im neuen System ist die einzige Belastung der Schlitten und dessen Lagerung, die beiden Motoren sind in gegenüberliegenden Ecken des Tisches dauerhaft befestigt und dienen jeweils als ein Lagerpunkt (und Antriebspunkt) eines Riemens. Nur die Lagerung der Y-Achse hat leichte Reib-Einflüsse. Das bedeutet, dass der Schlitten der einzige Teil des Systems ist, der mit einer nennenswerten Geschwindigkeit und Masse bewegt wird, und dass daher viel weniger Vibrationen auftreten.

Da die Riemen des Systems dauerhaft auf Spannung gehalten werden, ist kein Spiel mehr im System festzustellen. Positionen der Schachfiguren werden millimetergenau und mit einer hohen Wiederholgenauigkeit angefahren.

Ein weiterer Vorteil ist, dass CoreXY das gleiche Bauvolumen bei geringeren Gesamtabmessungen bieten kann. Der Fahrweg der Schachfiguren konnte somit ausgeweitet werden, ohne den Bauraum des Tisches zu ändern, da bei einem CoreXY-System jeder Punkt der gesamten Bauplatte angefahren werden kann, ohne zusätzlichen Platz zu benötigen. Bei einem Außenmaß des Tisches von 620x620 mm wies der erste Prototyp einen Fahrweg von 480x480 mm auf, während die zweite Revision mit selben Außenmaßen einen Fahrweg von 580x580 mm erreicht. Der fehlende Raum der ersten Version ist insbesondere auf die Lagerung der Motoren zurückzuführen, die jeweils ihre eigene Achse verkürzten. Nun liegen beide Motoren auf der x-Achse und dienen sogar als Bremse vor den Steuerkomponenten.

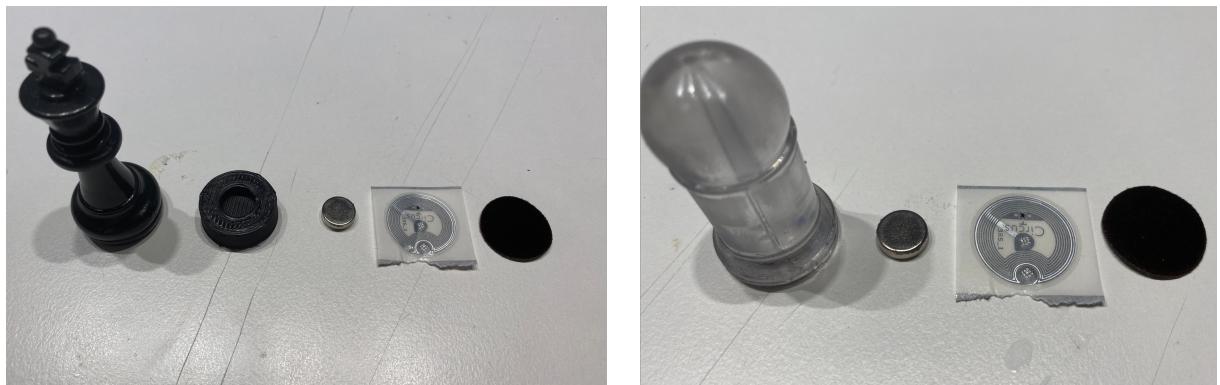


Bild 6-2: Schachfiguren im Vergleich

Die Konstruktion ist zudem stabiler, es erleichtert das Einschließen und Aufstellen im ausgeschalteten Zustand.

Zudem ist die Steuerung des CoreXY-Systems bereits in der Firmware Marlin integriert. Sie ist lediglich mittels des Parameters CoreXY zu aktivieren und die Schrittweite auszurechnen. Wird anschließend eine zu fahrende Strecke vorgegeben, berechnet Marlin selbstständig, wie schnell und in welche Richtung jeder der beiden Motoren bewegt werden muss.

In der Komplexität des Aufbaus und dem Zeitaufwand der Montage war kein Unterschied zwischen der ersten und zweiten Revision zu erkennen. Für Anfänger im Bereich CoreXY ist das Verlegen und insbesondere das stramme Spannen der Riemen eine Herausforderung, die sich aber durch die Konstruktion der Bauteile vereinfachen lässt. Insbesondere die Verbindung an der Lagerung der Y-Achse ist so erstellt worden, dass die Riemen nach der Durchführung nur in eine definierte Richtung gespannt werden können.

Das Resultat übertrifft sogar die Erwartungen. Die Mechanik ist robust und es konnten keine Fehler mehr im Betrieb festgestellt werden. Einzelne Fehler durch nachgiebige 3D-Konstruktionen wurden ausgebessert und so ein optimales und möglichst beständiges X-Y-System erzeugt.

## 6.2 Optimierung der Spielfiguren

Die bisher genutzten vorgefertigten Figuren funktionierten grundsätzlich gut mit dem ersten Prototyp. Allerdings weisen sie eine zu hohe Fehleranfälligkeit, in Bezug auf das

gegenseitige Beeinflussen (Abstoßen, Anziehen) durch die verwendeten Magnete auf. Zusätzlich stellt der Fertigungsprozess einer Figur einen zeitlichen Aufwand dar, da diese jeweils aus fünf **6-2** Einzelteilen bestehen:

- Figur
- Basisplatte
- Magnet
- NFC Tag
- Filzgleiter

Die Größe der Figuren kann durch die fest definierte Schachfeldgröße von 57mm und die verwendeten NFC Tags nicht verändert werden. Nach einigen Testdurchläufen mit dem ersten Prototyp war zu erkennen, dass sich die Figuren je nach aktueller Situation auf dem Spielfeld weiterhin magnetisch anziehen. Um diesen Fehler zu beheben wurden verschiedene Bewegungsgeschwindigkeiten getestet. Es ergaben sich allerdings für diesen Anwendungsfall keine merklichen Verbesserungen.

Dies führt je nach Spielverlauf zu Komplikationen, sodass die Figuren manuell vom Benutzer wieder mittig auf den Felder platziert werden müssen. Um dies zu verhindern, wurden einige Figuren zusätzlich mit einer 20mm Unterlegscheibe am Boden beschwert. Dies behob das Problem, jedoch erwies sich das NFC Tag nicht mehr als lesbar.

Die aktuell verwendeten Figuren des ersten Prototyps wiegen zwischen 8 Gramm für die Bauern und 10 Gramm für die restlichen Figuren. Der Test mit der Unterlegscheibe ergab, dass diese mit zusätzlichen 5 Gramm genug Gewicht hinzufügt, um die magnetische Beeinflussung zu unterbinden.

Testweise wurden einige Figuren mittels 3D Druck erstellt, um so das Gewicht zu erhöhen. Nach einem erfolgreichen Test wurde das CAD Modell so angepasst, dass sich der Magnet direkt in den Boden der Figur einkleben lässt. Des Weiteren wurden bei den Bauern die Magnete ausgetauscht. Die zuerst verwendeten 10x3mm Neodym-Magnete wurden bei diesen Figuren gegen 6x3mm Magnete getauscht. Somit sind im Design zwei verschiedene Arten von Magneten notwendig. Jedoch traten in den anschließend durchgeführten Testläufen keine Beeinflussungen mehr auf.

Durch diese Veränderungen am Figur-Design konnte zusätzlich die Zeit für den Zusammenbau einer einzelnen Figur gesenkt werden. Es werden nur noch vier **6-2** Einzelteile (Figur, Magnet, NFC-Tag, Filzgleiter) verwendet und zusätzliches Verkleben der Teile ist nicht mehr notwendig. Diese können direkt in den Fuß der Figur mittels einer

Presspassung eingelegt werden.

## 6.3 Änderungen der Elektronik

Mit einem relevanter Kritikpunkt, welcher bereits während des Aufbaus des ersten Prototyps zu erkennen war, war die Umsetzung der Elektronik. Diese wurde im ersten Prototyp manuell aufgebaut und enthielt viele verschiedene Komponenten.

Die verwendeten Motortreiber stellten sich während der Entwicklung als sehr flexibel heraus, stellten aber auch einen signifikanten Kostenfaktor dar. Nach Aufbau und Erprobung des ersten Prototyps wurde ersichtlich, dass hier nicht alle zuerst angedachten Features der Treiber benötigt wurden und so auch Alternativen in Betracht gezogen werden konnten. Zusätzlich konnte die Elektronik nur beschränkt mit anderen Systemen verbunden werden, was insbesondere der verwendeten SPI Schnittstelle geschuldet war.

All diese Faktoren erschweren einen einfachen Zusammenbau des autonomen Schachttischs. Die Lösung stellte die Verwendung von Standardhardware dar. Nach der Minimierung der elektrischen Komponenten und des mechanischen Aufbaus war zu erkennen, dass der autonome Schachttisch einer CNC-Fr bzw. einem 3D Drucker stark ähnelt. Insbesondere die XY-Achsen Mechanik sowie die Ansteuerung von Schrittmotoren wird in diesen Systemen verwendet. Mit dem Durchbruch von 3D Druckern im Consumer-Bereich sind auch kleine und preisgünstige Steuerungen 6.1 erhältlich, welche 2-3 Schrittmotoren und diverse zusätzliche Hardware ansteuern können.

Hierbei existiert eine große Auswahl solcher Steuerungen mit den verschiedensten Ausstattungen. Bei der Auswahl wurde vor allem auf die Möglichkeit geachtet, sogenannte Silent-Schrittmotortreiber verwenden zu können, um die Geräuschemissionen durch die Motoren so weit wie möglich zu minimieren. Im ersten Prototyp wurden unter anderem aus diesem Grund die [TMC5160-BOB](#) Treiber ausgewählt. Die meisten Boards bieten austauschbare Treiber, so dass es auch im Nachhinein möglich ist, diese auszuwechseln.

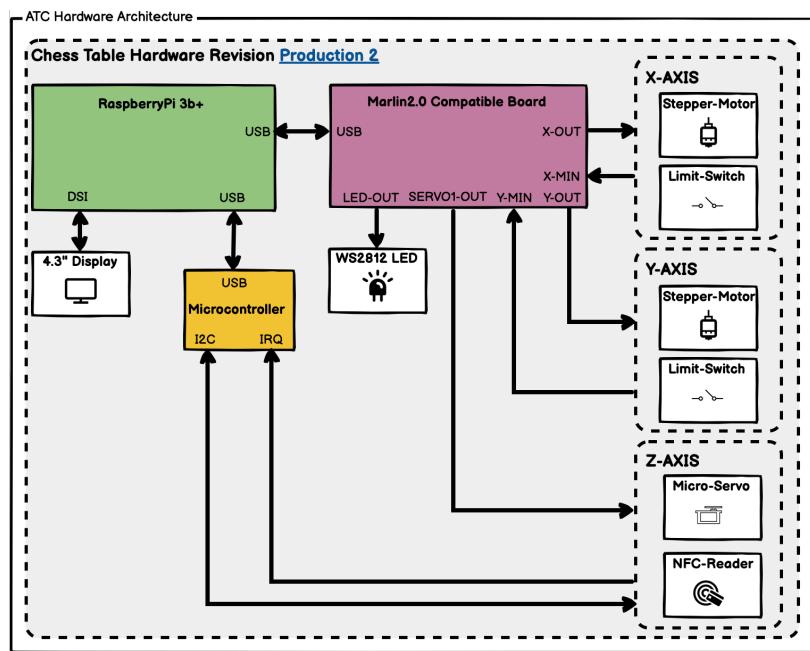


Bild 6-3: Production Hardware: Blockdiagramm

Tabelle 6.1: Standardhardware 3D Drucker Steuerungen

	SKR 1.4 Turbo	Ramps 1.4	Anet A8 Mainboard
Stepper Driver	TMC2209	A4988 / TMC2209	A4988
LED Strip Port	WS2811 / RGB	-	-
Firmware	Marlin-FW 2.0	Marlin-FW 1.0	Proprietary

Hierzu wurde der Schrittmotor-Treiber [TMC2209](#) gewählt, welcher diese Features ebenfalls unterstützt und in der Variante als Silent-Step-Stick direkt in die meisten 3D Drucker Steuerungen eingesetzt werden kann. Hierbei ist es wichtig, dass auf der gewählten Steuerung die Treiber-ICs nicht fest verlötet sind, sondern getauscht werden können.

Ein weiterer Punkt ist die Kommunikation der Steuerung mit dem Host-System. Hierbei setzten alle untersuchten Steuerungen auf die USB Schnittstelle und somit ist eine einfache Kommunikation gewährleistet. Der verwendete [Raspberry Pi](#) als eingebettetes System im autonomen Schachttisch bietet vier freie USB Anschlüsse, somit ist eine einfache Integration gewährleistet.

Nach einer gründlichen Evaluation der zur Verfügung stehenden Steuerungen wurde die SKR 1.4 Turbo-Steuerung ausgewählt, da diese trotz des geringfügig höheren Markt- preises genug Ressourcen auch für spätere Erweiterung bietet und eine Unterstützung für die neuste Version der Marlin-FW[32] bereitstellt. Somit wurde die Elektronik durch die verwendete Plug&Play Komponenten stark vereinfacht 6-3.

## 6.4 Anpassungen HAL

### 6.4.1 Implementierung GCODE-Sender

Durch die durchgeführten Änderungen an der Elektronik, insbesondere durch die Verwendung einer Marlin-FW[32] fähigen Motorsteuerung, war eine Anpassung des HAL notwendig. Diese Steuerung unterstützt die Ansteuerung der Motoren und anderer Komponenten (z.B. Spindeln, Heizelemente) mittels G-Code und wird typischerweise in 3D Druckern und CNC-Fr eingesetzt.

G-Code ist eine Programmiersprache, welche mittels einfacher textbasierter Befehle 6.2 Komponenten diese Maschinen kontrollieren kann. Dabei können einzelne Achsen verfahren werden oder die Drehzahl einer Spindel kontrolliert werden. Der G-Code wird von der Steuerung interpretiert. In der Regel wird dieser zuvor von einem CAD Programm erzeugt und zeilenweise übertragen. Bei einem 3D Drucker wird dieser vom Slicer generiert und enthält die Wegpunkte, welche vom Hotend angefahren werden sollen.

Im Falle des autonomen Schachtischs werden die G-Code Anweisungen on-the-fly durch den HAL erzeugt und die Motorsteuerung verfährt die Achsen an die jeweils gewünschten Positionen.

Marlin-FW bietet dabei einen großen Befehlssatz an G-Code Kommandos. Bei diesem Projekt werden jedoch nur einige wenige G-Code Kommandos verwendet 6.2, welche sich insbesondere auf die Ansteuerung der Motoren beschränken.

Tabelle 6.2: Grundlegende verwendete G-Code Kommandos

	G-Code Command	Parameters
Move X Y	G0	X <sub>dest_pos_x_mm</sub> Y <sub>dest_pos_y_mm</sub>
Move Home Position	G28	-
Set Units to Millimeters	G21	-
Set Servo Position	M280	P <sub>servo_index</sub> S <sub>servo_position</sub>
Disable Motors	M84	X Y

Die erforderlichen Kommandos wurden auf ein Minimum beschränkt, um eine maximale Kompatibilität mit verschiedenen G-Code-fähigen Steuerungen zu gewährleisten. Die Software unterstützt jedoch weitere Kommandos wie zum Beispiel [M150](#), mit welchem spezielle Ausgänge für LEDs gesteuert werden können. Dieses Feature bietet sowohl die verwendete [Marlin-FW](#) als auch die verwendete Steuerung an. Sollte die verwendete Steuerung solch ein optionales Kommando nicht unterstützen, so wird dieses ignoriert, was zur Folge hat, dass auch preisgünstige Steuerungen verwendet werden können.

Die Kommunikation zwischen Steuerung und eingebettetem System geschieht über eine USB Verbindung. Die Steuerung meldet sich als virtuelle serielle Schnittstelle im System an und kann über diese mit der Software kommunizieren. Auch werden so keine speziellen Treiber benötigt, da auf nahezu jedem System ein Treiber [USB-CDC](#) für die gängigsten USB-zu-Seriell Wandler bereits installiert ist. Die Software erkennt anhand der zur Verfügung stehenden USB-Ger sowie deren Vendor und Product-ID-Informationen die verbundene Steuerung und verwendet diese nach dem Start automatisch. Hierzu wurde zuvor eine Liste 6.3 mit verschiedenen getesteten Steuerungen sowie deren USB-Vendor und Product-ID angelegt.

Tabelle 6.3: Hinterlegte G-Code Steuerungen

Product	Vendor-ID	Product-ID	Board-Type
Bigtreetech SKR 1.4 Turbo	1d50	6029	Stepper-Controller
Bigtreetech SKR 1.4	1d50	6029	Stepper-Controller
Bigtreetech SKR 1.3	1d50	6029	Stepper-Controller

Damit die Software mit der Steuerung kommunizieren kann, wurde eine G-Code Sender Klasse implementiert, welche die gleichen Funktionen wie die HAL-Basisklasse bereitstellt. Nach Aufruf einer Funktion zum Ansteuern der Motoren wird aus den übergebenen Parametern das passende G-Code Kommando in Form einer Zeichenkette zusammengesetzt und auf die serielle Schnittstelle geschrieben.

```

1 //GCodeSender.cpp
2 bool GCodeSender::setServo(const int _index,const int _pos) {
3     return write_gcode("M280 P" + std::to_string(_index) + " S
4     " + std::to_string(_pos));
5 }
6 bool GCodeSender::write_gcode(const std::string _gcode_line,
7     bool _ack_check) {
8     //...
9     //FLUSH INPUT BUFFER
10    port->flushReceiver();
11    //APPEND NEW LINE CHARACTER IF NEEDED
12    if (_gcode_line.rfind('\n') == std::string::npos)
13    {
14        _gcode_line += '\n';
15    }
16    //WRITE COMMAND TO SERIAL LINE
17    port->writeString(_gcode_line.c_str());
18    //WAIT FOR ACK
19    return wait_for_ack();
20 }
21 bool GCodeSender::wait_for_ack() {
22     int wait_counter = 0;
23     //...
24     while (true) {
25         //READ SERIAL REONSE
26         const std::string resp = read_string_from_serial();
```

```

27      // ...
28      //PROCESS RESPONSE
29      if (resp.rfind("ok") != std::string::npos)
30      {
31          break;
32      }else if(resp.rfind("echo:Unknown") != std::string::
33          npos) {
34          break;
35      }else if(resp.rfind("Error:") != std::string::npos) {
36          break;
37      }else if (resp.rfind("echo:busy: processing") != std::
38          string::npos) {
39          wait_counter = 0;
40          LOG_F("wait_for_ack: busy_processing");
41      }else {
42          //READ ERROR COUNTER AND HANDLING
43          wait_counter++;
44          if (wait_counter > GCODE_ERROR_RETRY_COUNT)
45          {
46              break;
47          }
48      }
49      //...
50  }
```

Die Steuerung verarbeitet diese und bestätigt die Ausführung mit einer Acknowledgement-Antwort. Hierbei gibt es verschiedenen Typen. Der einfachste Fall ist ein `ok`, welches eine erfolgreiche Abarbeitung des Kommandos signalisiert. Ein weiterer Fall ist die Busy-Antwort `echo:busy`. Diese signalisiert, dass das Kommando noch in der Bearbeitung ist und wird im Falle des autonomen Schachtisches bei langen und langsam Bewegungen der Mechanik ausgegeben. Das System wartet diese Antworten ab, bis eine finale `ok`-Antwort zurückgegeben wird. Erst dann wird das nächste Kommando aus der Warteschlange bearbeitet.

### 6.4.2 I2C-Seriell Umsetzer

Durch den Wegfall der zuvor eingesetzten Elektronik und den Austausch durch die [SKR 1.4 Turbo](#) Steuerung ist jedoch ein Anschluss des [PN532 NFC Moduls](#) nicht mehr direkt möglich, da dieses mittels I2C Interface direkt mit dem eingebetteten System verbunden war. Dieses Interface entfällt nun. Dennoch besteht weiterhin eine Anschlussmöglichkeit, jedoch wurde auch hier auf eine USB Schnittstelle gewechselt. So ist es möglich, das System auch an einem anderen Host-System zu betreiben, wie zum Beispiel an einem

handelsüblichen Computer.

Dazu wurde ein Schnittstellenwandler entwickelt, welcher die I2C Schnittstelle zu einer USB seriell wandelt. Dabei wurde ein [Atmega328p](#) Mikrokontroller eingesetzt, da dieser weit verbreitet und kostengünstig zu beschaffen ist. Die Firmware des Mikrokontrollers stellt ein einfaches kommandobasiertes Interface bereit. Die Kommunikation ist mit der Kommunikation und der Implementierung des G-Code Senders vergleichbar und nutzt die gleichen Funktionen zur Kommunikation mit der seriellen Schnittstelle.

```
1 //userboardcontroller.cpp Atmega328p Firmware
2 //simplyfied version
3 char scan_nfc_tag(){
4     //...
5     if (nfc.tagPresent())
6     {
7         //READ TAG CONTENT
8         NfcTag tag = nfc.read();
9         //READ NDEF PAYLOAD
10        NdefMessage msg = tag.getNdefMessage();
11        if(msg.getRecordCount() > 0){
12            //READ FIRST RECORD
13            NdefRecord record = msg.getRecord(0);
14            const int payloadLength = record.getPayloadLength
15                ();
16            byte payload[payloadLength];
17            //...
18            record.getPayload(payload);
19            //...
20            //...
21            //RETURN FIGURE ID
22            if(payloadLength == 6){
23                return payload[3];
24            }
25        }
26    }
27    return 0; //VALID TAGS FROM 1-127
28 }
```

In diesem Falle wird nur ein Befehl zum Auslesen des NFC Tags benötigt. Das Host-System sendet die Zeichenkette `_readnfc_` zum Mikrokontroller und dieser versucht über das [PN532](#) Modul ein NFC Tag zu lesen. Wenn dieses erkannt wird und einen passenden Payload enthält, antwortet dieser mit dem String `_readnfc_res_PICTURE-ID_ok_` oder wenn kein Tag gefunden wurde mit `_readnfc_res_empty_`. Auch hier wird wie bei der G-Code Sender Implementierung auf Fehler bei der Kommunikation bzw. einem Abbruch durch einen Timeout reagiert. Das System initialisiert die serielle Schnittstelle neu und resettet das System durch Setzen des `DTR`-GPIO am USB-Seriell Wandler IC (falls vorhanden).

```

1 //UserBoardController.cpp HOST - SYSTEM
2 ChessPiece::FIGURE UserBoardController::read_chess_piece_nfc()
{
3     ChessPiece::FIGURE fig;
4     fig.type = ChessPiece::TYPE::TYPE_INVALID;
5     //...
6     //READ SERIAL RESULT
7     const std::string readres = send_command_blocking(
        UBC_COMMAND_READNFC);
8     //...
9     //SPLIT STRING -
10    const std::vector<std::string> re = split(readres,
        UBC_CMD_SEPARATOR);
11    //READ SECTIONS
12    //...
13    const std::string figure = re.at(3);
14    const std::string errorcode = re.at(4);
15    //CHECK READ RESULT
16    if(errorcode == "ok"){
17        if(figure.empty()){
18            break;
19        }
20        //...
21        //DETERMINE FINAL READ FIGURE
22        const char figure_charakter = figure.at(0);
23        fig = ChessPiece::getFigureByCharakter(
            figure_charakter);
24    }
25    //...
26    return fig;
27 }
```

Das System erkennt den Anschluss der Hardware beim Start auf die gleiche Art und Weise wie der G-Code Sender. Dafür wurden einige verschiedene Mikrokontroller im System hinterlegt 6.4, auf welchen die Firmware getestet wurde.

Tabelle 6.4: Hinterlegte Mikrokontroller

---

Product	Vendor-ID	Product-ID	Board-Type
Arduino Due [Programming Port]	2341	003d	User-Move-Detector
Arduino Due [Native SAMX3 Port]	2341	003e	User-Move-Detector
CH340	1a86	7523	User-Move-Detector

Product	Vendor-ID	Product-ID	Board-Type
HL-340	1a86	7523	User-Move-Detector
STM32F411	0483	5740	User-Move-Detector

## 6.5 Fazit bezüglich des finalen Prototyps

Der in der zweiten Iteration entstandene Prototyp wurden viele Elemente aus der ersten Iteration grundlegend überarbeitet. Dabei endstand ein völlig neues Design, welches sich auf einfach zu beschaffende Komponenten und Materialien stützt. Dies ermöglicht einen simpleren und zeitlich effektiveren Zusammenbau des vollständigen autonomen Schachttischs und bietet die Möglichkeit einer einfachen Erweiterung des Systems.

Aus der Verwendung des CoreXY Aufbaus resultiert eine nahezu spiel- und verlustfreie Mechanik (+-1mm), welche für diesen Zweck überaus geeignet ist. Somit konnten die mechanischen Probleme des ersten Prototyps vollständig eliminiert werden und es wird somit eine zuverlässige Spielführung erzielt. Diese Zuverlässigkeit wurde im mehreren Testläufen verifiziert und ein abschliessender sechs Stunden Dauertest bestätigte diese zusätzlich. Auch konnte die Bewegungsgeschwindigkeit des Schlittens erhöht werden, was zu einem schnelleren Platzieren der Figuren führt.

Ein großer Kritikpunkt des ersten Prototyps waren die nicht vollständig funktionsfähigen Park-Positionen für die ausgeschiedenen Figuren. Durch die Vergrößerung des Bewegungsspielraums der Achsen und die Anpassungen in der Software ist es nun möglich, alle Figuren vom Spielbrett entfernen zu können. Das System ist darauffolgend auch in der Lage, diese wieder in das Spielgeschehen zurückholen zu können. Somit ist kein manuelles Eingreifen durch den Benutzer mehr notwendig, wenn ein neues Spiel gestartet werden soll.

Zusätzlich wurde durch das transparente Design eine neue Art der Benutzerinteraktion geschaffen. Durch die visuellen Hinweise, welche der Tisch mittels der LED Beleuchtung geben kann, ist der Nutzer nicht mehr auf das GUI angewiesen und erfährt visuell, ob der gegnerische Spielzug beendet wurde. Der Nutzer kann ohne Aufwand erkennen, in welchem Zustand sich der autonome Schachttisch befindet.

Zudem konnte eine reibunglose Erkennung des getätigten Schachzug umgesetzt werden, welches bei der vorherigen Version nicht vollständig umsetzbar war. Durch den modularen Aufbau des HAL und des erweiteren Revisions-Managements ist es zudem möglich, die Software auf allen bisher erstellen Prototypen ausführen zu können.

Somit ist festzuhalten, dass mit der zweiten Revision alle zuvor geforderten Eigenschaften 7.1 zufriedenstellend umgesetzt werden konnten. Die erstellte Hard- und Software bietet zusätzlich zahlreiche Erweiterungsmöglichkeiten.

Tabelle 6.5: Eigenschaften die finalen Prototypen

---

### ATC – autonomous Chessboard

---

Feldabmessungen (LxBxH)	57x57mm
Abmessungen (LxBxH)	620x620x170mm
Gewicht	5.7kg
Konnektivität	WLAN, USB
Automatisches Bewegen der Figuren	ja
Erkennung Schachfigurstellung	ja
Spiel Livestream	ja
Cloudanbindung (online Spiele)	ja
Parkposition für ausgeschiedene Figuren	ja
Stand-Alone Funktionalität	ja
Besonderheiten	visuelle Hinweise per Beleuchtung

---

Dennoch ist zu beachten, dass dieser Stand des Projekts noch nicht vollständig ausgereift ist und noch Verbesserungspotential bietet, welches zum Beispiel vor einem

kommerziellen Verkauf des Produktes notwendig wären. Dabei besonders markant ist die Erkennung der vom Benutzer getätigten Schachzüge. Durch die Verwendung des NFC Moduls und den Scavorgang des Schachfeldes muss eine gewisse Wartezeit von ca. 20 Sekunden in Kauf genommen werden, bevor das System einen Zug erkennt. Somit sind keine schnellen Partien möglich wie zum Beispiel bei Schachformen wie dem Schnellschach, bei denen die Zugzeit begrenzt ist.

# 7 Entwicklung der Cloud Infrastruktur

Die erste Phase der Entwicklung des Gesamtsystems (autonomer Schachtisch + Cloud-Anbindung) bestand in der Auslegung und Erstellung der Cloud-Infrastruktur und der darauf ausgeführten Services. Diese stellt dabei die Funktionalität der Kommunikation zwischen den einzelnen Schachtischen bereit, sodass mit mehreren, Ortsunabhängig gegeneinander gespielt werden kann. Zusätzlich stellt diese auch optionale Computer-Spieler als Service bereit, sodass Nutzer auch die Möglichkeit haben gegen den Computer spielen zu können. Dazu stellt die Cloud-Infrastruktur zusätzliche Mechanismen bereit um Spieler, welche auf der Suche nach einem Spiel sind mit anderen Suchenden zu verbinden.

Die “Cloud” stellte in diesem Zusammenhang einen Server dar, welcher aus dem Internet über eine feste IPv4 und IPv6-Adresse erreichbar war und frei konfiguriert werden konnte. Auf diesem System wird der ATC-Cloud Stack 7-1 installiert, welcher zum einen aus der Schach-Software bestand, welche in einem Docker-Stack ausgeführt wurde und zum anderen, weitere Dienste, welche bei der Entwicklung des Systems genutzt werden. Diese sind jedoch kein Bestandteil des ATC-Cloud-Systems.

## 7.1 API Design

Das System soll so ausgelegt werden, dass zu einem späteren Zeitpunkt verschiedene Client-Devices mit diesem kommunizieren können. Dazu zählen zum einen der autonome Schachtisch, aber zum Beispiel auch ein Web-Client, welcher die Funktionalität eines Schachtisches im Browser abbilden kann. Hierzu muss das System eine einheitliche Representational State Transfer (REST)-Schnittstelle bereitstellen.

Die RESTful Application Programming Interface (API) stellt verschiedene Ressourcen bereit, welche über eine URI 7-2 eindeutig identifizierbar sind. Auf diese können mittels verschiedenster HTTP Anfragemethoden (unter anderem: [GET](#), [POST](#)) zugegriffen werden.

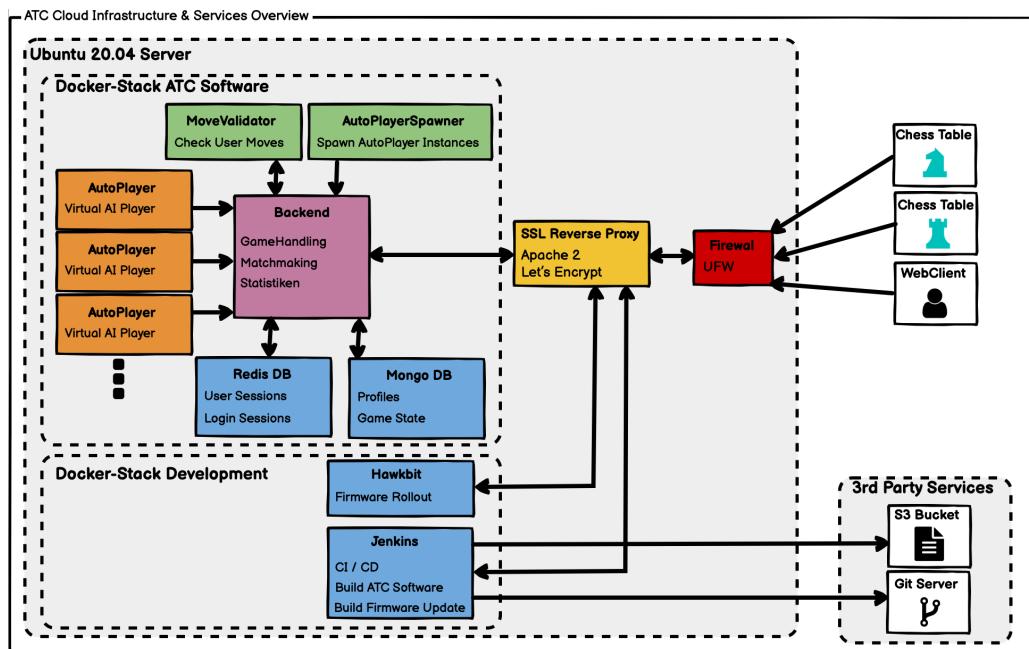


Bild 7-1: Gesamtübersicht der verwendeten Cloud-Infrastruktur



Bild 7-2: Cloud-Infrastruktur: Aufbau einer URI

Jede dieser Methoden stellt einen anderen Zugriff auf die Ressource dar und beeinflusst somit das Verhalten und die Rück-Antwort dieses Zugriffs.

Eine URI besteht dabei aus mehreren Teilen. Das Schema gibt an, wie die nachfolgenden Teile interpretiert werden sollen. Dabei werden bei einer RESTful Schnittstelle typischerweise das Hypertext Transfer Protocol (HTTP) Protokoll sowie Hypertext Transfer Protocol Secure (HTTPS) verwendet. Dabei steht HTTPS für eine verschlüsselte Verbindung.

Somit stellt die RESTful API eine Interoperabilität zwischen verschiedenen Anwendungen und Systemen bereit, welche über ein Netzwerk miteinander kommunizieren. Dieser Ansatz ist somit geeignet, um den verschiedenen Client Systemen (Schachtisch, Webclient) eine Kommunikation mit dem Server zu erlauben.

## 7.2 Service Architektur

Der komplette Software-Stack, welcher zum Betrieb der Schach-Cloud notwendig ist, wurde in einer sehr vereinfachten Mikroservice-Architektur angelegt. Dies bedeutet, dass hier zum Betrieb notwendige Softwarekomponenten in mehrere kleinere Bestandteile ausgelagert wurden **7-3**, die jeweils auf wenige Funktionalitäten fokussieren. Durch dieses modulare Design ist es zusätzlich möglich, die eigentliche Schach-Logik auslagern und in der Theorie auch andere Spiele implementieren zu können.

Diese einzelnen Komponenten sind eigenständig ausführbar und erst die Vernetzung dieser in einem gemeinsamen privaten Netzwerk bilden eine funktionfähige Schachcloud. Somit setzt sich diese aus den folgenden Komponenten zusammen:

- Backend
- MoveValidator
- AutoPlayer

Da jeder dieser Services stateless ist und keine eigenen Daten speichert, werden zwei Datenbank-Services benötigt, um die Spieldaten zu speichern:

- Mongo NoSQL Datenbank
- Redis In-Memory Key Value Datenbank

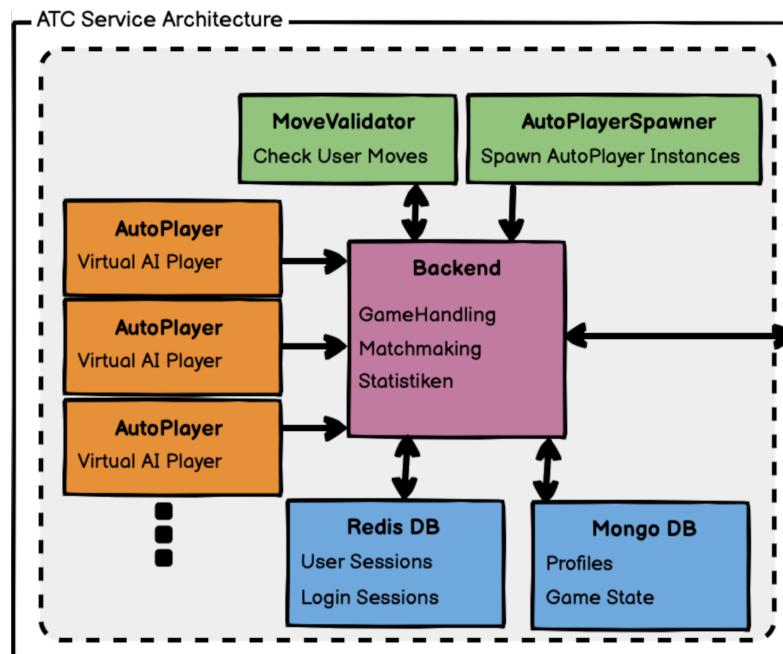


Bild 7-3: Cloud-Infrastruktur: Aufbau der Service Architecture

Hierbei wurde auf zwei verschiedene Datenbanken gesetzt, was im Folgenden erläutert wird.

Die [Redis](#) [24] Datenbank wird ausschließlich für die Speicherung der aktiven Sessions der einzelnen verbundenen Clients verwendet. Durch das verwendete Sessionsystem, bei dem jeder Client in kurzen Intervallen seine Aktivität bestätigen muss, bietet diese Datenbank den Vorteil, dass diese durch ihre Architektur sehr schnell auf die angeforderten Datensätze zugreifen kann. Auch wird hier nur der Datensatz gespeichert, welcher die notwendigen Informationen zu der aktiven Session des Clients speichert. Dieser werden durch die ID des Clients abgefragt. Ferner wird der Zeitstempel der Anmeldung sowie die letzte Anfrage des Clients in Form eines JavaScript Object Notation (JSON) Dokuments gespeichert.

```

1  {
2    "client_hwid": "h34724",
3    "login_ts": 1622128754,
4    "heartbeat_ts": 1622158754
5  }

```

Durch den Key-Value-Ansatz sowie den hohen Verbrauch an Arbeitsspeicher eignet sich diese Datenbank jedoch nicht zum Speichern der Spieldaten. Hierzu wurde ein zusätzlicher [Mongo](#) [?] Datenbank Service erstellt, in welchem diese Daten abgelegt werden. Zusätzlich zu den Spieldaten (Spiele, Spielstände, Statistiken) werden auch die Nutzerprofile speichert. Ein Profil wird beim ersten Anmeldevorgang erstellt und

enthält neben den Profilinformationen te-ID, Namen, Spielertyp) auch die Referenzen auf die gewonnenen und verlorenen Spiele. Diese Referenzen können später für die Visualisierung verwendet werden.

Alle aufgelisteten Services werden in separaten Containern betrieben. Die Containervirtualisierung geschieht mittels der Software [Docker](#) [5]. Diese stellt ein einfaches Interface zur Erstellung und Verwaltung von Containern bereit. Um einen Container auf dem System starten zu können, muss dieser zunächst aus einem Image heraus erstellt werden. Dieses Image wird mittels eines [Dockerfile](#) beschrieben. Das [Dockerfile](#) besteht dabei aus einer Reihe an Kommandos, welche den Aufbau des Images beschreiben.

Bei diesem Projekt besteht ein Image in der Regel aus einem vorgefertigten [Ubuntu 20.04](#) Image, in welchem zusätzliche Software installiert wird, die zur Ausführung der eigentlichen Software benötigt wird. Auch existieren bereits vorgefertigte Images, welche bereits Software für einen spezifischen Anwendungsfall enthalten.

```
1 # Dockerfile for ATC_AutoPlayer
2
3 FROM golang:latest #USE golang AS BASE IMAGE
4 RUN mkdir /app
5 ADD . /app/ # COPY SOURCE FILE OVER
6 WORKDIR /app
7 RUN ls
8 RUN cd ./stockfish-11-linux/src/ && make clean && make build
    ARCH=autodetect
9 RUN go mod init AutoPlayer ; exit 0
10 RUN CGO_ENABLED=0 GOOS=linux go build -a -installsuffix cgo -o
     main .
11 CMD ["/app/main"] # START APP
```

Da die Architektur aus mehr als einem Container besteht, gestaltet sich ein manuelles Management dieser Container als nicht praktikabel. Zu diesem Zweck existieren mehrere Tools und Systeme, um solche Aufgaben zu automatisieren. Ein weiterer nicht zu vernachlässigender Punkt sind die Abhängigkeiten, welche unter den Containern bestehen. In diesem Fall benötigt der Backend-Service die beiden Datenbanken, um starten zu können. Somit ist es essentiell, dass diese bereits zuvor erfolgreich gestartet werden. Solche Funktionalitäten deckt das sehr leichtgewichtige Tool [docker-compose](#) [6] ab. Durch eine entsprechende Konfigurationsdatei kann ein so genannter Stack aus mehreren Containern aufgebaut werden.

```
1 # docker-compose.yml STACK CONFIGURATION src_server
2 version: "3"
3 services:
4   AtomicChessBackend:
```

```
5      container_name: atcbackend
6      depends_on:
7          - AtomicChessRedisDatabase
8          - AtomicChessMongoDatabase
9          - AtomicChessMoveValidator
10     links:
11         - "redisdb:AtomicChessRedisDatabase"
12         - "mongodb:AtomicChessMongoDatabase"
13         - "movevalidator:AtomicChessMoveValidator"
14     image: atcbackend:latest
15     build:
16         context: ../ATC_Backend/
17     restart: always
18     ports:
19         - 3000:3000
20     environment:
21         - PRODUCTION=1
22
23 AtomicChessMoveValidator:
24     container_name: atcmovevalidator
25     build:
26         context: ../ATC_MoveValidator/
27     image: atcmovevalidator:latest
28     restart: always
29     ports:
30         - 5000:5000
31     environment:
32         - PRODUCTION
33
34 AtomicChessRedisDatabase:
35     image: redis:latest
36     restart: always
37     container_name: atcredis
38     ports:
39         - 6379:6379
40
41 AtomicChessMongoDatabase:
42     image: mongo:latest
43     container_name: atcmongo
44     restart: always
45     environment:
46         - MONGO_LOG_DIR=/dev/null
47     volumes:
48         - ./data/db:/data/db
49     ports:
50         - 27017:27017
51     command: mongod --logpath=/dev/null --quiet
52
53 AtomicChessAutoPlayer:
54     depends_on:
55         - AtomicChessBackend
56     links:
```

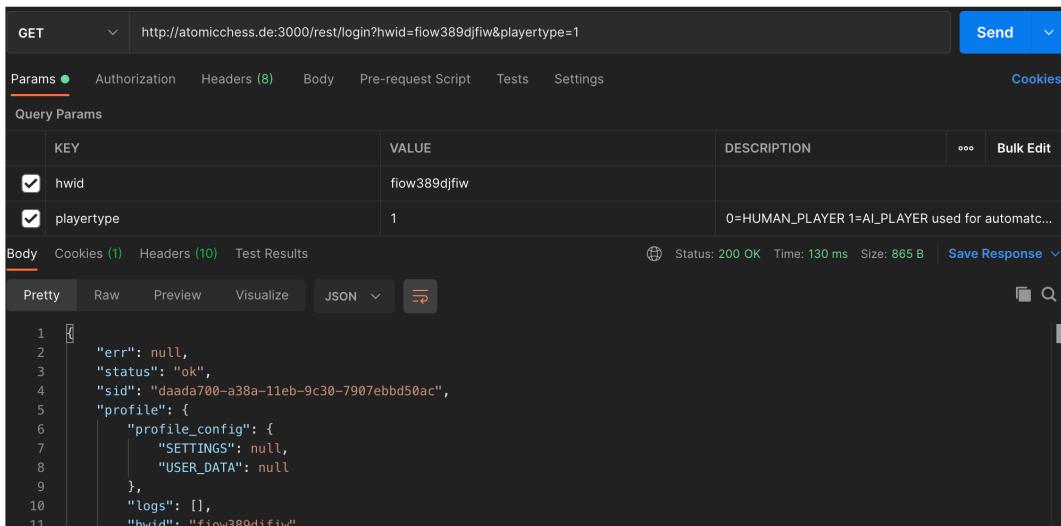


Bild 7-4: Cloud-Infrastruktur: Backend Login-Request und Response

```

57      - "backend:AtomicChessBackend"
58 build:
59   context: ../ATC_AutoPlayer/
60 image: atcautoplayer:latest
61 restart: always
62 scale: 5 # SPAWN THREE INSTANCES
63 environment:
64   - PRODUCTION=1
65   - BACKEND_IP=backend:3000 #HOST IP:PORT OF BACKEND
66   #- USE_HOSTNAME_HWID=TRUE # USE THE MACHINE HOSTNAME AS
67   # HWID
     #- PLAYER_TYPE_HUMAN=1 # SIMULATE A HUMAN PLAYER
  
```

## 7.3 Service: Backend

Das Backend, welches den zentralen Teil der Service-Architektur bildet, stellt den Zugriffspunkt für die autonomen Schachtische und den Webclient (s.u.) dar. Dieses stellt die API zur Außenwelt bereit, mit dem sich die einzelnen Clients verbinden.

Dies geschieht zusätzlich durch einen Transport Layer Security (TLS)-Reverse Proxy, welcher eine verschlüsselte Verbindung HTTPS bereitstellt. Diese funktioniert sowohl mit einem self-signed Certificate als auch mit einem Zertifikat der Lets Encrypt Organisation[19]. Somit sind die vom Backend bereitgestellte API und die später erstel-

len Webclients (s.u.) für alle modernen Webbrowser vertrauenswürdig.

Bei dem eingerichteten Reverse-Proxy werden alle Verbindungen aus dem öffentlichen Internet mit einem Service verbunden, welcher im lokalen Netzwerk betrieben wird. In diesem Fall ist dies der lokale Server bzw `localhost`, auf dem der Backend-Service auf dem Port 3000 ausgeführt wird.

```
1 # APACHE 2 REVERSE PROXY CONFIGURATION
2 <IfModule mod_ssl.c>
3 <VirtualHost *:443>
4     ServerName atomicchess.de
5     ProxyPreserveHost On
6     DocumentRoot /var/www/html
7     ProxyPass /.well-known !
8     ProxyPass / http://127.0.0.1:3000/
9     ProxyPassReverse / http://127.0.0.1:3000/
10    ServerAdmin webmaster@atomicchess.de
11
12    ErrorLog ${APACHE_LOG_DIR}/error.log
13    CustomLog ${APACHE_LOG_DIR}/access.log combined
14
15    SSLCertificateFile /etc/letsencrypt/live/atomicchess.de/
16        fullchain.pem
17    SSLCertificateKeyFile /etc/letsencrypt/live/atomicchess.de/
18        /privkey.pem
19    Include /etc/letsencrypt/options-ssl-apache.conf
20 </VirtualHost>
21 </IfModule>
```

Durch diese Methode wird eine sichere Verbindung zwischen dem Service und dem Nutzer-Device hergestellt. Der Vorteil ist, dass die Services im privaten Netzwerk keine TLS Zertifikate benötigen, um in diesem Netz miteinander kommunizieren zu können. Lediglich bei einer Verbindung zum öffentlichen Internet wird eine sichere Verbindung durch die Forward-Proxy Funktion des Apache 2 Webservers hergestellt.

Der Backend-Service stellt die grundlegenden Funktionen bereit, welche die Clients benötigen. Dazu zählen unter anderem:

- Profilverwaltung
- Matchmaking
- Spielstatus
- Authentifizierung der Clients

Jeder Client meldet sich mittels der `/rest/login` Route an. Das Backend prüft, ob bereits ein Spielerprofil in der Datenbank angelegt wurde und erstellt ggf. ein neues

für das Device. Dabei werden der Spieler-Typ (Artificial Intelligence (AI), autonomer Schachtisch, Webclient) als auch die Geräte-(id) festgehalten. Nach einem erfolgreichen Login 7-4 erhält der Client ein Session-Token. Nur mit diesem Token können weitere Funktionen des Backends verwendet werden. Dieses Token ändert sich nach jedem Login-Prozess. Somit kann nur ein Client Token-Inhaber sein und die Tokens anderer, zuvor angemeldeter Clients werden ungültig.

Nach einem erfolgreichen Login kann der Client den Spielstatus abfragen, in welchem er sich befindet:

- Idle: kein Spiel aktiv und nicht auf der Suche nach einem Spiel
- Matchmaking: Spieler sucht aktiv nach einem Spiel
- Game-Running: Client ist einem aktiven Spiel zugewiesen

Der `Idle`-Status, wird direkt nach einem Login gesetzt. Somit wird der Client nicht automatisch Spielen zugewiesen. Dies kann durch die `/rest/set_player_state` API Route geändert werden. Diese wird vom Client aufgerufen, wenn dieser ein Spiel starten möchte. Dazu wird ein Eintrag in der Lobby-Tabelle der Datenbank erzeugt. In dieser befinden sich alle Spieler, welche auf der Suche nach einem Spiel sind. Dabei wird zusätzlich der Zeitpunkt des Eintretens gespeichert.

Wenn mindestens zwei Clients auf der Suche nach einem Spiel sind und sich somit in der Lobby-Tabelle befinden, wird der Matchmaking-Algorithmus aktiv. Dieser sortiert die Clients nach Zeitpunkt des Eintretens und nach dem Spieler-Typ. Der Spieler-Typ kann dabei einer der folgenden Clienten sein:

- autonomer Schachtisch `Human`
- Webclient `Human`
- AutoPlayer `AI`

Das System sortiert die Liste der suchenden Clients nach deren Typ. Somit wird sichergestellt, dass zuerst alle menschlichen Spieler zusammen ein Spiel beginnen und erst im letzten Schritt ein Mensch gegen den Computer spielen muss.

Zum Beispiel besteht die Spielerliste welche auf der Suche nach einem Match sind aus den folgenden Typen:

- 1. Webclient A `Human`

- 2. autonomer Schachtisch A [Human](#)
- 3. autonomer Schachtisch B [Human](#)

Alle Spieler sind vom Typ [Human](#) somit versucht das System mit jeweils zwei Spielern ein neues Spiel zu starten. Da der [Webclient A](#) und der [autonome Schachtisch A](#) bereits am längsten gewartet haben, werden diese zuerst ausgewählt. Das System entscheidet hierbei nicht die beiden autonomes Schachtisch Clienten zu verbinden, da hier zuerst auf die Wartezeit der Spieler rücksicht genommen wird. Nach dem Matchmaking sieht die Liste folgendermaßen aus:

- 1. autonomer Schachtisch B [Human](#)

Somit steht nur noch ein wartender Spieler auf der Liste, da dieser vom Typ [Human](#) ist, wartet das System auf einen weiteren Spieler. Sollte sich in einer definierten Zeit von circa 20 Sekunden kein weiterer Spieler vom Typ [Human](#) hinzukommen, wird automatisch ein [AI](#)-Spieler gestartet.

- 1. autonomer Schachtisch B [Human](#)
- 2. AutoPlayer [AI](#)

Somit wird mit diesen beiden Spielern ein weiteres Spiel gestartet. Somit wird sicher gestellt das jeder Spieler welcher mit einem autonomen Schachtisch oder Webclient spielt, auch zuerst gegen einen menschlichen Spieler spielen kann. Erst zum Schluss kommt ein Match gegen den Computer zustande, damit kein Spieler ewig lange auf einen Spielpartner warten muss.

```

1 //ATC_BACKEND matchmaking_logic.js
2 var matchmaking_job = new CronJob('*/' + CFG.getConfig().
3   matchmaking_runner_interval + ' * * * *', function () {
4     //GET ALL PLAYERS WITH SEARCHING FOR A NEW GAME IS ENABLED
5     LH.get_player_for_matchmaking(function (gpfm_err, gpfm_res
6       ) {
7       //...
8       //CHECK IF MORE THEN TWO PLAYERS ARE SEARCHING (HUMAN
9       + AI)
10      if (!gpfm_res || gpfm_res.combined_player_searching.
11        length <= 1) {
12        return;
13      }
14      // 1 HUMAN AND 1 AI SEARCHING => DIRECT MATCH

```

```
11     if (CFG.getConfig().matchmaking_ai_enable === true &&
12         gpfm_res.player_searching_human.length === 1 &&
13         gpfm_res.player_searching_ai.length >= 1) {
14             //...
15             //START A MATCH FOR THESE TWO PLAYERS => REMOVE
16             // LOBBY ENTRY FROM DB AND CREATE A NEW GAME IN
17             // THE GAME DATABASE
18             GH.start_match(gpfm_res.player_searching_human[0].hwid,
19                           gpfm_res.player_searching_ai[0].hwid,
20                           function (sm_err, sm_res) {
21                 if (sm_err) {
22                     //ON ANY ERROR THE CLIENT WILL RESET THE
23                     // FAULTY STATE ITSELF AND RELOGIN TO THE
24                     // SYSTEM
25                     console.error(sm_err);
26                 }
27             });
28         //MORE THAN 1 HUMAN PLAYER WAITING
29     } else if (gpfm_res.player_searching_human.length > 1)
30     {
31         //THEN MAKE A MATCH BEWEEN THE TWO HUMAN PLAYER
32         //SORT PLAYER WITH THE LONGEST WAIT TIME IN THE
33         // LOBBY
34         gpfm_res.player_searching_human.sort(
35             player_sort_function_swt);
36         //SELECT THE MOST WAITING PLAYER
37         const p1 = gpfm_res.player_searching_human[0];
38         //SELECT A RANDOM OTHER PLAYER
39         const p2 = gpfm_res.player_searching_human[
40             HELPER_FUNCTIONS.randomInteger(1, gpfm_res.
41                 player_searching_human.length - 1)];
42         //...
43         //START A MATCH
44         GH.start_match(p1.hwid, p2.hwid, function (sm_err,
45                         sm_res) {
46             if (sm_err) {
47                 //ON ANY ERROR THE CLIENT WILL RESET THE
48                 // FAULTY STATE ITSELF AND RELOGIN TO THE
49                 // SYSTEM
50                 console.error(sm_err);
51             }
52         });
53     }
54 }, true);
```

Kommt ein Match zustande, werden die Spielereinträge aus der Lobby-Tabelle entfernt und es wird ein neues Spiel in der Game-Tabelle der Datenbank angelegt.

Diese Tabelle enthält alle Spiele und deren aktuellen Status:

- aktuelles Spielbrett
- welcher Spieler aktuell am Zug ist
- Anzahl Schachzüge
- Spieler-(id)s
- Spielerfarbe
- Spiel-Status (abgebrochen, beendet)

Diese Einträge fragen die Clients in regelmäßigen Intervallen über die `/rest/player_state` Route ab. Somit kennen sie das aktuelle Spielfeld und ob sie gerade am Zug sind. Ein Zug wird mittels der `/rest/make_move` Route übermittelt. Das Backend überprüft diesen mittels des `MoveValidator`-Services und speichert das Ergebnis in dem passenden Datenbank-Record zum Spiel ab.

Nach Beendigung eines Spiels werden die Clients wieder in den `Idle`-Status zurückgesetzt. Somit können diese ein neues Spiel beginnen. Nach einem Sieg ermittelt das Backend einen Score für den Client, welcher gewonnen hat. Dieser wird in dem Profil-Record gespeichert und kann abgefragt werden. Somit wurde ein einfaches Profil-System implementiert.

Ein Client muss sich außerdem in regelmäßigen Abständen über die `/rest/heartbeat` Route zurückmelden. Somit weiß der Backend-Service, dass der Client noch existiert. Bleibt ein Request innerhalb einer bestimmten Zeit aus, werden alle aktuellen Spiele beendet und der Client wird aus dem System entfernt. Somit ist sichergestellt, dass beide Parteien bei einem gestarteten Spiel noch aktiv sind, auch wenn diese keine Schachzüge ausführen.

## 7.4 Service: MoveValidator

Der MoveValidator-Service bildet im System die eigentliche Schachlogik ab. Die Aufgabe ist es, die vom Benutzer eingegebenen Züge auf Korrektheit zu überprüfen und daraufhin einen neuen Spiel-Status zurückzugeben. Dazu zählen unter anderem das neue Schachbrett und ob ein Spieler gewonnen oder verloren hat. Bevor ein Spiel begonnen wird, generiert der MoveValidator das initiale Spielfeld und bestimmt den Spieler, welcher als erstes am Zug ist.

Der Backend-Service fragt ein neues Spiel an oder übergibt einen Schachzug inkl. des aktuellen Spielbrett-Aufbaus an den Service.<sup>7-5</sup> Der Response wird dann vom Backend

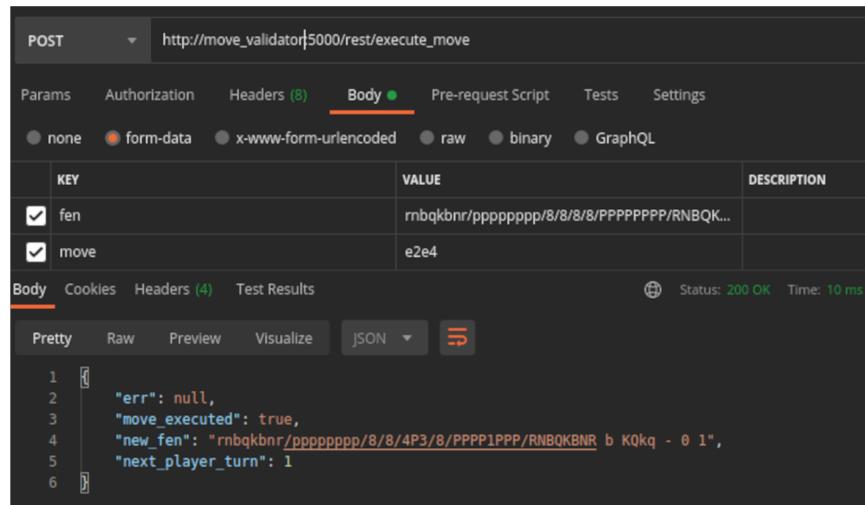


Bild 7-5: MoveValidator: Beispiel Request zur Ausführung eines Zuges auf einem gegebenen Schachbrett

in der Datenbank gespeichert und weiter an die Client-Devices verteilt.

Tabelle 7.1: MoveValidator-Service API Overview

MoveValidator-Function	API-Route	Method	Form-Data
Check Move	/rest/check_move	POST	fen, move, player
Execute Move	/rest/execute_move	POST	fen, move
Validate Board	/rest/validate_board	POST	fen
Init Board	/rest/init_board	GET	

Allgemein geschieht die Kommunikation über vier API Calls, welche vom MoveValidator-Service angeboten werden 7.1. Als erstes wird vom Backend der `/rest/init_board` Request verwendet, welcher ein neues Spielbrett in der Forsyth-Edwards-Notation (FEN) Notation zurückgibt, welches zum Start der Partie verwendet wird. Allgemein wurde das gesamte System so umgesetzt, dass es mit einem Spielfeld in einer Zeichenkettenrepräsentation arbeitet. Dies hat den Vorteil, dass die Spielfeld-Notation leicht angepasst werden kann. Mit diesem Design ist es möglich, auch andere Spielarten im System zu implementieren, da nur an dieser Stelle die initialen Spielfelder generiert werden und Züge der Spieler validiert werden müssen.

Tabelle 7.2: Vergleich FEN - X-FEN

FEN-TYPE	FEN-String
FEN	rnbqbnr/pp1ppppp/8/2p5/4P3/5N2/PPPP1PPP/RNBQKB1R
X-FEN	rnbqbnr/pp1ppppp/8/2p5/4P3/5N2/PPPP1PPP/RNBQKB1R b KQkq - 1 2
SCHEMA	Board Player-Color Rochade En-Passant Halfturn Turn-Number

Alle gängigen Schachprogramme und Bibliotheken unterstützen das Laden von Spielbrettern in der FEN bzw X-FEN Schreibweise, ebenso die für den MoveValidator Service verwendete [Python-Chess](#)[7] Bibliothek. Diese unterstützt zusätzlich die Generierung der für den Benutzer möglichen Schachzüge, welche auf dem aktuellen Brett möglich sind.

Diese Liste wird vom System dazu verwendet, um sicherzustellen, dass der Benutzer nur gültige Züge tätigen kann. Diese Funktion lässt sich zusätzlich abschalten, falls das Spiel nicht nach den allgemeinen Schachregeln verlaufen soll. Bei der Generierung der möglichen Schachzüge muss zwischen den [Legal-Moves](#) und den [Pseudo-Legal](#) Schachzügen unterschieden werden. Die [Legal-Moves](#) beinhalten nur die nach den Schachregeln möglichen Züge, welche von Figuren des Spielers ausgeführt werden können. Die [Pseudo-Legal](#) Schachzüge sind alle Schachzüge, welche von den Figuren auf dem aktuellen Schachbrett möglich sind. Somit sind hier zum Beispiel auch die möglichen Schachzüge aller Figuren enthalten, auch wenn der König im Schach steht und so eigentlich nur Züge möglich sind welche diese Situation beseitigen.

Wenn ein Spieler an der Reihe ist und einen Zug getägt hat, wird sein getätigter Zug mittels der `/rest/check_move` API überprüft und festgestellt, ob dieser gemäß der Legal-Moves durchführbar war. Ist dies der Fall, wird der Zug auf dem online-Spielbrett angewendet. Dies geschieht durch die `/rest/execute_move` API. Diese führt den Zug aus, ermittelt anschließend das neue Spielbrett und überprüft zusätzlich, ob das Spiel gewonnen oder verloren wurde.

Hat der Benutzer jedoch einen ungültigen Zug ausgeführt, wird dieser vom System gestrichen und der Client des Benutzers stellt den Zustand des Spielbretts vor dem



Bild 7-6: WebClient: Spielansicht

getätigten Zug wieder her. Danach hat der Benutzer die Möglichkeit, einen alternativen Zug auszuführen.

## 7.5 Service: WebClient

Der WebClient wurde primär dazu entwickelt, um das System während der Entwicklung zu testen. Dieser simuliert einen autonomen Schachtisch und verwendet dabei die gleichen HTTP Requests. Um das zu ermöglichen, wurde dieser vollständig in JavaScript (JS) sowie Hypertext Markup Language (HTML) und Cascading Style Sheets (CSS) umgesetzt und ist somit komplett im Browser ausführbar.

Ausgeliefert werden die statischen Dateien zur Einfachheit durch den Backend-Service; es wurde kein gesonderter Frontend-Service angelegt. Durch die Implementierung des Webclients in JS ist dieser sogar lokal über einen Browser ausführbar, ohne dass die benötigten Dateien über einen Webserver ausgeliefert werden müssen.

Zusätzlich zu dem verwendeten Vanilla-JS wurde [jQuery](#)[22] als zusätzliche JS Bibliothek verwendet, was eine Manipulation der HTML Elemente stark vereinfacht. jQuery bietet insbesondere einfach zu nutzende HTTP-Request Funktionen bzw. Asynchronous JavaScript and XML (AJAX) an, welche für die Kommunikation mit dem Backend-Service verwendet werden. Diese werden im Hintergrund eingesetzt, so dass der WebClient auto-

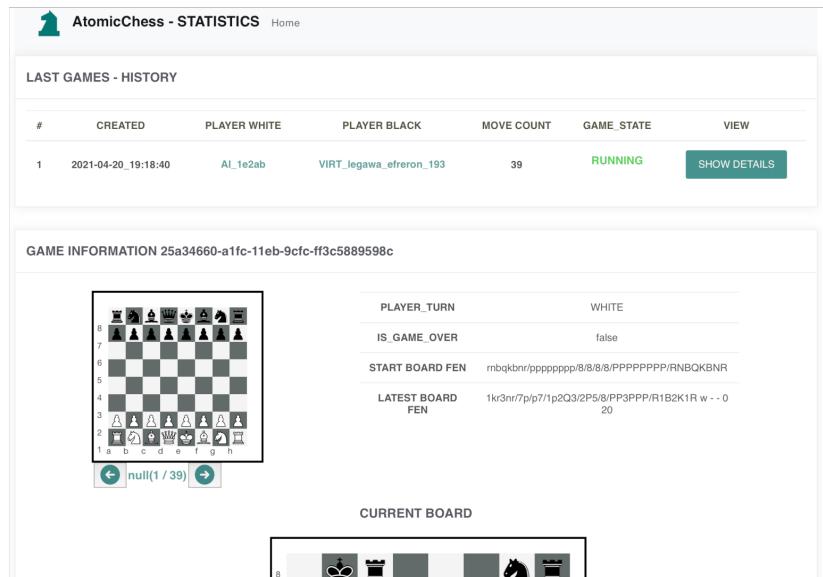


Bild 7-7: Webclient: Statistiken

matisch den neuen Spielzustand dem Benutzer anzeigt. Dies geschieht mittels [polling](#), bei dem der Webbrower in zyklischen Abständen die aktuellen Spiel-Informationen vom Backend-Service abfragt. Diese Methode wurde verwendet, um eine maximale Kompatibilität mit verschiedenen gegebenenfalls älteren Web-Browsern sicherzustellen. Eine moderne Alternative ist die Verwendung von WebSockets, bei welchen der Web-Browser eine direkte Transmission Control Protocol (TCP)-Verbindung zum Webserver (in diesem Fall zum Backend-Service) aufnimmt und so eine direkte Kommunikation stattfinden kann ohne Verwendung der [polling](#)-Methode.

Der Hauptanwendungsfall des Webclients 7-6 während der Entwicklung war es, weitere Spieler zu simulieren und so ein Spiel mit nur einem autonomen Schachtisch testen zu können. Durch den Webclient ist zusätzlich möglich, gezielt Spiele und Spielzüge zu simulieren. Hierzu gehören vor allem Sonderzüge wie die Rochade oder der En-Passant Zug. Auch können durch den Webclient ungültige Züge simuliert werden, welche durch die verwendete Schach-AI nicht getätigten werden.

Während der Implementierung wurde der Webclient weiter ausgebaut und es wurden weitere Eigenschaften ergänzt. Dazu zählt zum einen eine Übersicht über vergangene und aktuell laufende Spiele. In dieser können Spiele Zug um Zug nachvollzogen werden und weitere Informationen über den Spielstatus angezeigt werden.<sup>7-7</sup> Auch ist es möglich, aktuell laufende Spiele in Echtzeit anzeigen zu lassen; somit wurde eine Livestream-Funktionalität implementiert.

## 7.6 Service: AutoPlayer

Der AutoPlayer-Service stellt den Computerspieler bereit.

Jede Service-Instanz stellt einen virtuellen Spieler bereit, welcher die gleichen Schnittstellen wie der Webclient oder der autonome Schachtisch verwendet. Die einzige Änderung an den verwendeten REST-Calls ist der Login-Request. Hier wird das `playertype` Flag gesetzt, welches den Spieler als Computerspieler gegenüber dem System authentifiziert. Daraus resultierend wird dieser während des Matchmaking-Prozesses erst für ein Match ausgewählt, wenn keine menschlichen Spieler mehr zur Verfügung stehen. Dieser digitale Gegenspieler ist vom Typ Webclient oder autonomer Schachtisch. Dieser Prozess gewährleistet zudem, dass immer zuerst die menschlichen Spieler ein Spiel beginnen und die digitalen nur Alternativen darstellen.

Eine weitere Modifikation ist die Verwendung einer Schach-AI, da dieser Service als Computerspieler agieren soll. Hierzu kam die Open-Source Chess Engine `Stockfish`[28] in der Version 11 zum Einsatz. Die Stockfish-Engine bietet noch weitere Features als nur die besten Züge zu einem gegebenen Schachbrett zu ermitteln.

Die AutoPlayer-Instanz kommuniziert über das Universal Chess Interface (UCI) Protokoll[27] mit der Engine. Dieses Protokoll wird in der Regel von Schach-Engines verwendet, um mit einem GUI zu kommunizieren.

Um das aktuelle Spielbrett in der Engine zu setzen, wird dieses in der X-FEN Notation mit dem Präfix `position fen` als Klartext an die Engine übergeben und diese sendet daraufhin eine List möglicher Züge zurück. Die erste Indexposition dieser Liste ist dabei der von der Engine am besten bewertete Zug.

Im Kontext des AutoPlayer-Service wird der Engine nur das aktuelle Spielbrett übermittelt und der beste Zug auslesen. Dies wird ausgeführt, wenn der AutoPlayer am Zug ist. Nachdem die Engine einen passenden Zug gefunden hat, wird das Ergebnis über den `make_move (+rest)`-API Call übermittelt.

Wenn das Match beendet wird, beendet sich auch die Service-Instanz. Diese wird jedoch wieder gestartet, wenn die Anzahl der zur Verfügung stehenden Computerspieler unter einen definierten Wert fällt. Somit ist dafür gesorgt, dass das System nicht mit ungenutzten AutoPlayer-Instanzen gebremst wird. Diese Anzahl ?? ist in der Konfiguration des Backend-Service frei wählbar und kann je nach zu erwarteten Aufkommen angepasst werden.

Allgemein skaliert das System durch diese Art der Ressourcenverwaltung auch auf kleinen Systemen sehr flexibel. Durch die Art der Implementierung, dass sich der AutoPlayer-Service wie ein normaler Spieler verhält, sind auch andere Arten des Computerspieler möglich. So ist es zum Beispiel möglich, die Spielstärke je Spieler anzupassen oder einen Computerspieler zu erstellen, welcher nur zufällige Züge zieht.

Ein weiterer Anwendungsfall für den AutoPlayer-Service ist das Testen des weiteren Systems, insbesondere des Backend-Service. Durch das Erstellen eines Spiels mit zwei AutoPlayer-Instanzen können automatisierte Schachpartien ausgeführt werden, um die Funktionsfähigkeit des restlichen Systems zu testen. Dieses Feature wurde insbesondere bei der Entwicklung des Webclients und der Steuerungssoftware für den autonomen Schachtisch verwendet.

# 8 Embedded System Software

Die Embedded System Software ist die Hauptsoftware, welche auf dem eingebetteten System ausgeführt wird. Als Basis-System dient das über das [Buildroot](#) erstellte Linux-System, in welchem die Software nach dem Start ausgeführt wird.

Um eine einfache Integration in das Linux-System zu gewährleisten, wurde ein [Buildroot](#)-Paket erstellt **8-1**, welches über den Konfigurations-Dialog ausgewählt werden kann. Somit kann ein komplettes System-Image erstellt werden, welches die Software für den autonomen Schachttisch und dessen eingebettetes System enthält.

Hierbei ist die Software in zwei Teile aufgeteilt:

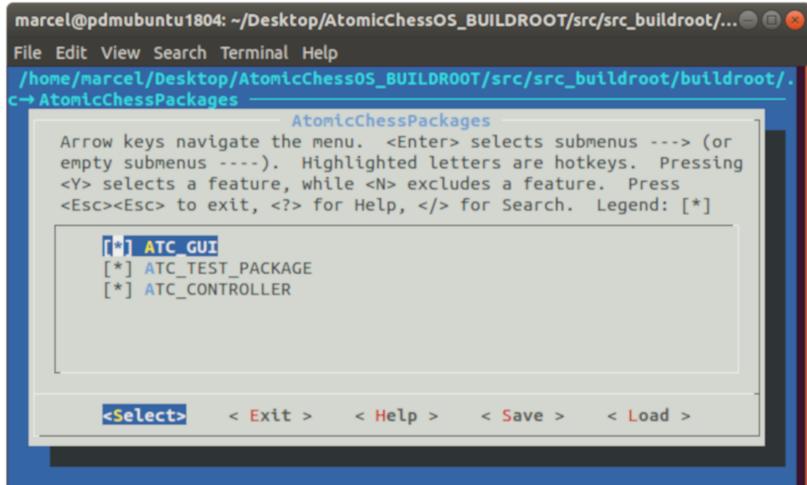
- Controller
- GUI

Dabei stellt die Controller-Software die Hauptsoftware zur Ansteuerung der Mechanik dar. Auch übernimmt diese die Kommunikation mit der Cloud-Infrastruktur und die Berechnung der Figur Positionen.

Die GUI Anwendung stellt dabei die Schnittstelle mit dem Benutzer dar. Diese generiert alle visuellen Elemente, welche über das Display am autonomen Schachttisch dargestellt werden, soll und leitet Eingaben des Nutzers an die Controller-Software weiter.

## 8.1 Ablaufdiagramm

Nach dem Start der Controller-Software folgt diese einem fest vorgegebenen Ablauf **8-2**. Dieser wird mittels einer State-Machine in der Controller-Software abgebildet. Nachdem die Software gestartet ist, wird zuerst eine Verbindung mit dem Cloud-Server aufgenommen. Da der Tisch eine Art Thin-Client darstellt, bei dem die eigentliche Spiellogik



**Bild 8-1:** Embedded System Software: Buildroot Pakete

auf dem Server ausgeführt wird, muss die Controller-Software nur das vom Server vorgegebene Schachfeld mittels der Mechanik synchronisieren und entsprechende Schachzüge des Benutzers an diesen übermitteln.

Nach erfolgreicher Anmeldung am Cloud-Server kann der Benutzer ein Spiel starten, welches lediglich zu einem entsprechenden Request an den Server führt. Die nachfolgende Dauerschleife überprüft, ob ein Spiel gestartet wurde. Dazu wird in zyklischen Intervallen die `/rest/get_playerstate` API aufgerufen. Diese stellt Informationen bereit, ob und in welchem Status sich das Spiel für den anfragenden Client befindet.

Wurde das Spiel gerade erst gestartet, beginnt die Sync-Phase. Bei dieser müssen beide Clients die Figuren in die vorgegebene Ausgangsstellung bringen und dies bestätigen. Erst dann gilt das Spiel für den Server als begonnen und der aktive Spieler wird ausgewählt. Ist der Client am Zug, wartet dieser auf einen Zug in Form einer Benutzereingabe. Dies kann entweder durch manuelles Eintippen des Schachzugs über das GUI geschehen oder über eine manuelle Bewegung der Figuren auf dem Schachtisch. Auch hier hat der Client keine Informationen darüber, ob der getätigte Zug gültig ist. Die Zuginformationen werden über die entsprechende API Route `/rest/make_move` an den Server übermittelt, welcher diesen Zug auf dem Schachbrett ausführt. Wenn der Zug ungültig ist, muss der Client den Benutzer informiert werden, diesen Zug rückgängig zu machen. Ist der Zug jedoch gültig, wird dieser vom Server an den anderen Client übermittelt und dieser muss anschließend wie bei der Sync-Phase das Spielbrett aufbauen bzw. verändern.

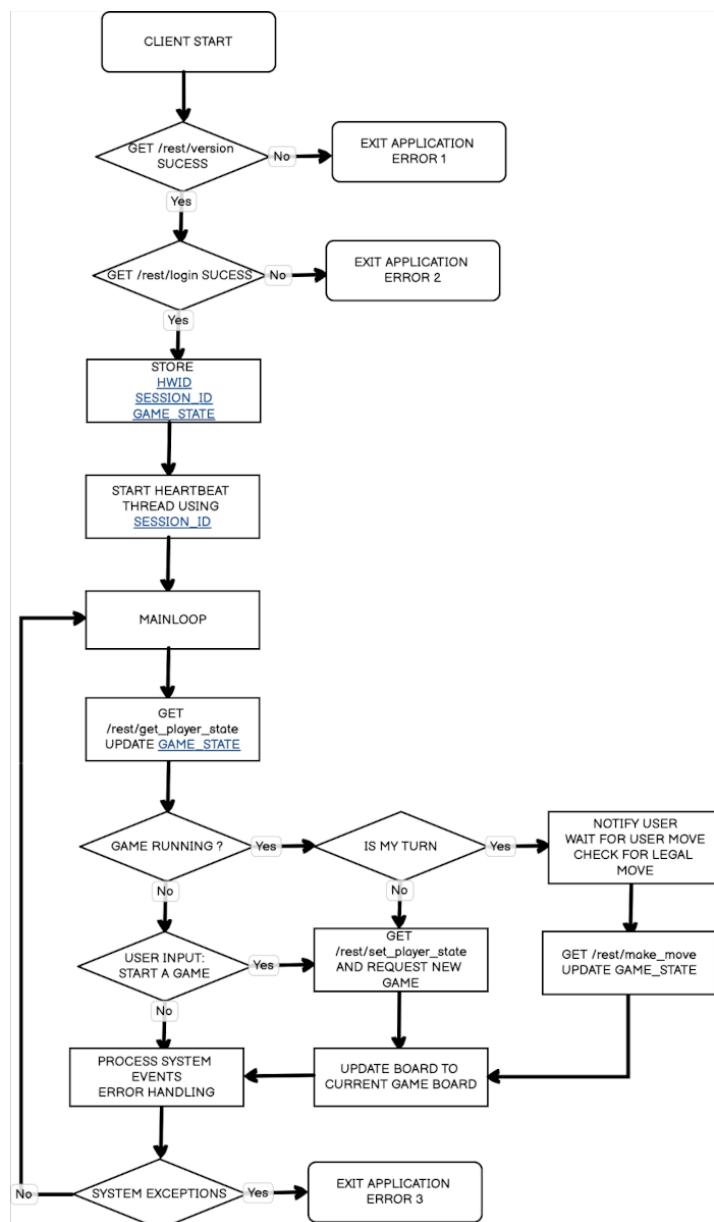


Bild 8-2: Embedded System Software: Ablaufdiagramm

Nach einem Abbruch oder einem Gewinn oder Verlust des Spiels wartet der Client wieder, bis ein neues Spiel vom Server aus gestartet wird, oder der Benutzer manuell ein Spiel startet. Dieser Zyklus wird dauerhaft ausgeführt. Der Client bietet jedoch noch weitere Einstellungsmöglichkeiten für den Benutzer über das GUI an. Diese Benutzer-Events werden separat verarbeitet und sind vom Spielablauf getrennt. Hierzu zählen unter anderem der Kalibrierungs-Dialog sowie eine Informationsansicht über den aktuellen Status des Systems.

## 8.2 Figur Bewegungspfadberechnung

Nach dem Start der Software werden durch das Abscannen jedes einzelnen Feldes die Anzahl und Typen der Figuren ermittelt. Dies stellt sicher, dass sich die erforderliche Anzahl der Figuren beim Systemstart auf dem Spielbrett befindet, ansonsten in ein Start des Programms nicht möglich.

Während der Sync-Phase muss die Software das vorgegebene Schachfeld erstellen. Dazu hält die Software den aktuellen Brett-Zustand vor und vergleicht diesen mit dem Ziel-Schachbrett. Durch den Vergleich können die Figuren mit geänderter Position identifiziert werden. Dadurch dass immer Ziel und aktuelles-Spielbrett miteinander verglichen werden, können mehrere Züge auf einmal durchgeführt werden. Hierbei ist es auch möglich, auf Spielbrettern in einem beliebigen Zustand wieder die Ausgangsposition herstellen zu können. Somit kann ein beliebiges Spielfeld vorgegeben werden, welches der Tisch dementsprechend aufbaut.

Um dies zu ermöglichen, wird aus dem Vergleich der beiden Spielbretter die Differenz in Form einer Liste gebildet. In dieser sind alle erforderlichen Änderungen der einzelnen Felder vermerkt. Eine Änderung besteht aus der Figur, welche sich aktuell auf dem Brett befindet, und dem Ziel-Zustand.

```
1 //ChessBoard.cpp
2 std::vector<ChessBoard::FigureFieldPair>
3 ChessBoard::compareBoards(ChessPiece::FIGURE *_board_a,
4     ChessPiece::FIGURE *_board_b, bool _include_park_pos) {
5     std::vector<ChessBoard::FigureFieldPair> diff_list;
6     ChessPiece::FIGURE *board_current = get_board_pointer(
7         ChessBoard::BOARD_TPYE::REAL_BOARD);
8     ChessPiece::FIGURE *board_target = get_board_pointer(
9         ChessBoard::BOARD_TPYE::TARGET_BOARD);
10    //...
11    //NOW CHECK BOARD DIFFERENCES
```

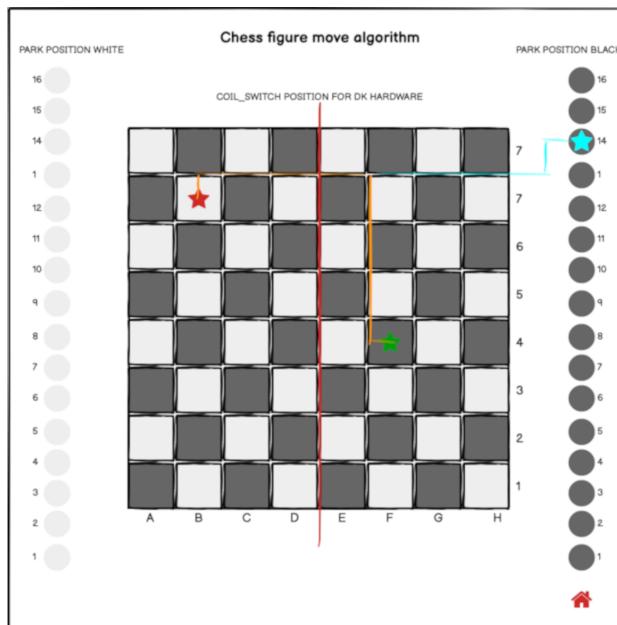
```

9     for (int i = ChessField::field2Index(ChessField::
10         CHESS_FILEDS::CHESS_FIELD_A1);
11             i < ChessField::field2Index(ChessField::CHESS_FILEDS
12                 ::CHESS_FIELD_PARK_POSITION_WHITE_1); i++) {
13     ChessPiece::FIGURE tmp_curr = getFigureOnField(
14         board_current, ChessField::Index2Field(i));
15     ChessPiece::FIGURE tmp_target = getFigureOnField(
16         board_target, ChessField::Index2Field(i));
17     //CHECK IF EQUAL FIGURES
18     if (ChessPiece::compareFigures(tmp_curr, tmp_target))
19     {
20         continue;
21     }
22     ChessBoard::FigureFieldPair tmp;
23     tmp.field_curr = ChessBoard::FigureField(ChessField::
24         ::Index2Field(i), tmp_curr);
25     tmp.field_target = ChessBoard::FigureField(ChessField
26         ::Index2Field(i), tmp_target);
27     tmp.processed = false;
28     diff_list.push_back(tmp);
29 }
30 return diff_list;
31 }
```

Aus dieser Liste können anschließend einzelne Figur-Bewegungen abgeleitet werden. Dazu wird zu einer Änderung des Start-Feldes in der Liste ein weiteres Listenelement gesucht, bei welchem die Änderung im Zielfeld liegt. Somit können Start- und Zielfeld für eine Figur bestimmt werden. Anzumerken ist, dass die errechneten Züge nicht die logischsten oder kürzesten darstellen müssen, da hier die Reihenfolge der Änderungen nach Vorkommen in der Liste entscheidend ist. Somit entsteht eine weitere Liste an Feld-Operationen, bei denen Figuren hinzugefügt, bewegt, entfernt werden können.

- überschüssige Figuren entfernen
  - wenn allgemein zu viele Figuren auf dem Feld sind
  - wenn bei dem auszuführenden Zug eine Figur geschlagen wird
- möglichen Zug ausführen
  - falls Figuren fehlen diese hinzufügen
  - sonst Figur an Zielposition bewegen
- Figur fehlt
  - Figur aus Park-Position auf das Spielbrett holen

Dieser Vorgang wird rekursiv solange ausgeführt, bis es keinen Änderungsbedarf auf dem Spielbrett mehr gibt. Der rekursive Ansatz ist notwendig, da Figuren ihren



**Bild 8-3:** Embedded System Software: Figur Wegpunkte

Bestimmungsort noch nicht einnehmen können, wenn dieser noch von einer anderen Figur belegt ist. Diese muss dann zuerst auf deren Ziel-Feld geschoben werden.

Aus den Start und Ziel-Feldern werden im letzten Schritt Wegpunkte **8-3** generiert. Diese beschreiben den Weg, welchen die Figur von Start zum Zielfeld ablaufen muss. Das Spielbrett wurde so designt, dass zwischen Figuren auf benachbarten Feldern immer noch eine weitere Figur Platz hat. Somit ist es möglich, dass die sich bewegenden Figuren zwischen zwei auf ihren Feldern stehenden hindurchbewegt werden können. Der Algorithmus berechnet genau diese Wegpunkte. Nachdem die Figur aus der Mitte des Feldes an dessen Rand bewegt wurde, kann die Figur ungehindert an den anderen Figuren vorbei bewegt werden. Die Figur wird anschließend in Richtung der X-Achse auf die Höhe des Zielfeldes bewegt, um darauffolgend auf der Y-Achse an die Kante des Zielfeldes bewegt zu werden. Der letzte Wegpunkt liegt im inneren des Zielfelds, sodass sich die Figur in der Mitte von diesem befindet.

Anzumerken ist, dass dieser Algorithmus nicht weiter optimiert wurde. Somit führen die Figuren gegebenenfalls Zick-Zack-Bewegungen aus, auch wenn das Zielfeld direkt neben dem Start-Feld liegt.

## 8.3 Schachfeld Scan Algorithmus zur Erkennung von Schachzügen

Ein weiterer wichtiger Teil der Controller-Software ist die Erfassung der Schachzüge, welche vom Benutzer getätigt wurden. Das System bietet dem Benutzer hier zwei Möglichkeiten, welche im Folgenden erläutert werden.

Über das User Interface (UI) des autonomen Schachtisches kann der Benutzer, wenn dieser am Zug ist, seinen Zug manuell eingeben. Hierbei wird das Start- und Ziel-Feld angegeben, woraus das System automatisch den gewünschten Zug ermittelt.

Dies ist jedoch bei einer Schachpartie nicht praktikabel. Der Benutzer muss eine Möglichkeit haben, die Schachfiguren händisch bewegen zu können. Das System muss aus den geänderten Figuren den getätigten Schachzug ermitteln können.

Da das Schachbrett in beiden Revisions-Varianten über keine Sensoren unter den einzelnen Schachfeldern verfügt, wurde der existierende NFC Scanner verwendet. Mit dem ist es möglich, gezielt Figuren auf zuvor bestimmten Feldern zu ermitteln. Der Nachteil dieser Methode ist die Wartezeit, welche aufgrund des Scan-Prozesses nötig ist. Ein Scan aller 64 Felder ist nicht praktikabel, da jeder Scan und die Bewegung der Mechanik ca 3 Sekunden benötigt. Zusätzlich verlängert sich die Scandauer durch ein leeres Schachfeld, da der Scanner mehrere Versuche unternimmt, dort ein gültiges NFC Tag zu erkennen.

Somit muss mittels eines Algorithmus **8-4** entschieden werden, welche Felder als mögliche Kandidaten in Fragen kommen. Hinweise auf diese Felder bietet der aktuelle Spiel-Status, welcher vom System über den Cloud-Service abgefragt wird. Dieser liefert nicht nur das aktuelle Schachbrett, sondern auch die möglichen Schachzüge, welche vom Benutzer ausgeführt werden können.

Durch diese Auflistung an möglichen Zügen wird anschließend eine Liste mit den möglichen Start-Feldern der Figuren erstellt. Anhand dieser Liste werden die Felder mittels des NFC Moduls auf Veränderungen überprüft. Stellt das System eine Änderung fest, wird ermittelt, auf welche Felder die Figur auf dem ursprünglichen Feld ziehen konnte. Anschließend werden alle Ziel-Feld-Positionen der Figur abgescannt, bis auch hier eine Änderung detektiert wurde. Aus diesen beiden Informationen lässt sich der getätigte Zug ableiten. Dieser wird anschliessend an der Cloud-Service zur Überprüfung weitergeleitet.

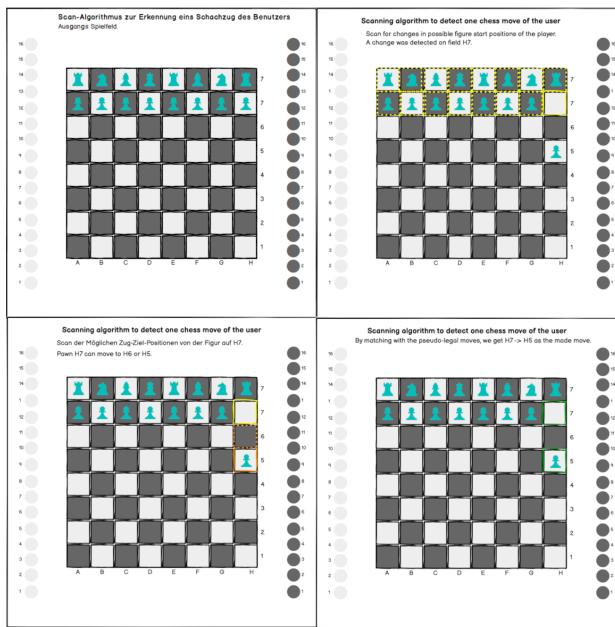


Bild 8-4: Embedded System Software: Schachfeld Scan Algorithmus Ablauf

Sollte kein Zug bestimmt werden können, gibt es zwei Möglichkeiten für das System. Zum einen kann der Benutzer informiert werden, dass sein getätigter Zug ungültig ist und zum anderen ist es möglich, alle Schachfelder auf einen möglichen alternativen Zug abzuscannen. Daraufhin kann der autonome Schachtisch den getätigten Zug manuell zurücksetzen. Dies kann vom Benutzer in den Einstellungen eingestellt werden, da ein manuelles Zurücksetzen wesentlich schneller durchgeführt werden kann. Danach hat der Benutzer die Möglichkeit, einen weiteren Zug durchzuführen, solange bis der getätigte Zug gültig ist.

Der gesamte Prozess des Scavorgangs dauert je nach Anzahl der Möglichkeiten welche der Spieler hat, um die 20 Sekunden bis das System den getätigten Zug ermittelt hat.

## 8.4 Inter Prozess Communication

Bei der Entwicklung des Systems wurde darauf geachtet, dass das User-Interface auswechselbar bleibt. Somit ist es auch möglich, ein web-basiertes User-Interface zu integrieren. Dazu wurde ein zusätzlicher Interprocess Communication (IPC) Layer hinzugefügt, welcher eine Abstraktion der von der User-Interface Software verwendeten Funktionen auf der Controller-Software Ebene bereitstellt.

Desweiteren wurde eine einfache IPC Bibliothek implementiert, welche sowohl dem Controller- als auch dem User-Interface als Shared-Library zur Verfügung steht. Diese stellt einfache Funktionen zum Senden und Empfangen von Events bereit und erzeugt nach der Initialisierung einen separaten Thread, in welchem die Kommunikation mit den anderen IPC Instanzen verwaltet wird.

Der Hauptthread des Programms kann anschließend über eine First In – First Out (FIFO) Message Queue die von den anderen Instanzen empfangenen Events in einer Polling-Loop abfragen und Events an die anderen Instanzen absetzen. Diese können mit der gleichen Vorgehensweise Events der jeweils anderen Instanzen empfangen, sowie Events erstellen und senden.

Die Kommunikation zwischen den IPC Instanzen geschieht hierbei über eine TCP Socket-Verbindung. Es wurde keine Shared Memory (speicherbasierte) Implementierung verwendet, da hier nur eine Kommunikation auf Betriebssystemebene möglich ist.

Durch die Socket-basierte Implementierung ist es möglich, die anderen IPC Instanzen auszulagern und auf verschiedenen Endgeräten ausführen zu lassen.

```
1 {
2   "event":12, //BEGIN_BTN_SCAN
3   "type":2, //CLICKED
4   "dest_process_id":"ui_qt_01",
5   "origin_process_id":"controller_sw_01",
6   "is_ack":false //Qos
7 }
```

Über die TCP Verbindung werden ausschließlich Daten im JSON Format übertragen. Dies macht ein einfaches Debugging und Steuerung über einen Webbrowser möglich, was die Implementierung während der Entwicklungsphase vereinfachte.

Zusätzlich kann über die Acknowledgement-Funktionalität sichergestellt werden, dass die anderen IPC Instanzen dieses Event erhalten haben. Diese müssen nach Erhalt das empfangene Event quittieren, was mittels des `is_ack` Flag zurückgemeldet wird.

```
1 //IPC guicommuicator.cpp
2 //SIMPLIFIED EXAMPLE USAGE
3
4 //INIT IPC SERVER
5 guicommuicator gui;
6 gui.start_recieve_thread();
7 //CHECK OTHER PROCESS REACHABLE
8 while (!gui.check_guicommuicator_reachable()){
```

```

9     gui_wait_counter++;
10    if (gui_wait_counter > GUI_WAIT_COUNTER_MAX){
11        break;
12    }
13 }
14 //...
15 //CHECK OTHER PROCESS VERSION NUMBER
16 if(gui.check_guicomunicator_version()){
17     LOG_F(WARNING, "guicomunicator version check failed");
18 }
19
20 //SWITCH MENU ON SCREEN TO PLEASE WAIT SCREEN
21 gui.createEvent(guicomunicator::GUI_ELEMENT::SWITCH_MENU,
22                 guicomunicator::GUI_VALUE_TYPE::PROCESSING_SCREEN);
22 //FLIP SCREEN ORIENTATION
23 gui.createEvent(guicomunicator::GUI_ELEMENT::
24                 QT_UI_SET_ORIENTATION_180, guicomunicator::GUI_VALUE_TYPE
25                 ::ENABLED);
26
26 //GET EVENT FROM OTHER PROCESSES STORED IN EVENT QUEUE
27 guicomunicator::GUI_EVENT ev = gui.get_gui_update_event();
28 if (!ev.is_event_valid){
29     gui.debug_event(ev, true);
30     continue;
31 }
31 //CHECK EVENT QUEUE FOR USER INPUT
32 if(ev.event == guicomunicator::GUI_ELEMENT::BEGIN_BTN_SCAN &&
33     ev.type == guicomunicator::GUI_VALUE_TYPE::CLICKED) {}
```

## 8.5 User Interface

Das User-Interface ist eines der zentralen Elemente, mit welchem der Benutzer interagiert. Hierbei soll dieses nur die nötigsten Funktionen bereitstellen, welche zur Bedienung des Schachttisches nötig sind. Aufgrund der kleinen Abmessungen des Displays mit 4.3 Zoll wurden alle Bedienelemente in ihrer Größe angepasst, sodass der Benutzer auch von einer weiter entfernten Position den Zustand direkt erkennen kann. Auch wurde die maximale Anzahl an Bedienelementen in einer Ansicht auf drei begrenzt. Die Spielansicht stellt zudem nur die eigene Spielerfarbe sowie welcher Spieler gerade am Zug ist dar. Somit soll der Spieler nicht vom Spiel abgelenkt werden. Nach dem Spielstart findet keine weitere Interaktion mit dem User-Interface mehr statt.

Trotz der Einfachheit der Bedienung und dem meist nur also Informationsquelle über den Spielstand dienenden User-Interface bietet dieses viele Möglichkeiten der Konfigu-

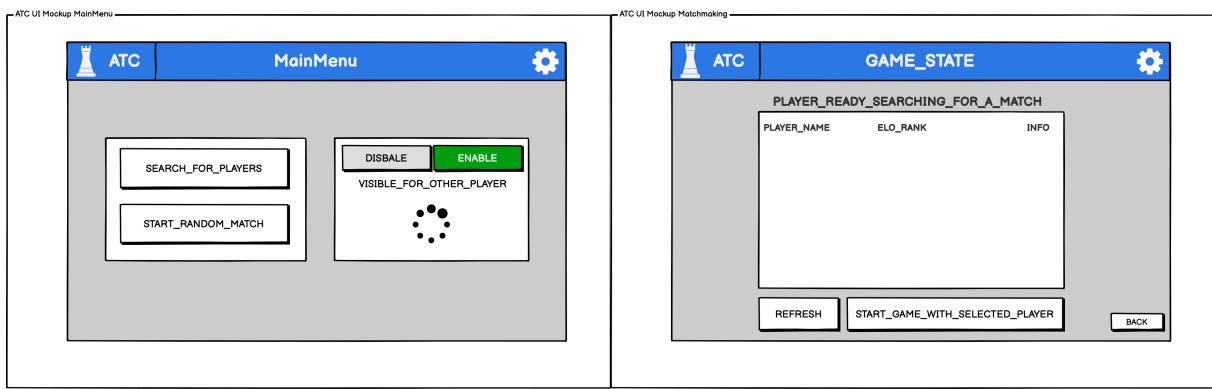


Bild 8-5: Embedded System Software: User-Interface Mockup

ration des Systems. Somit kann auf ein weiteres Eingabegerät, wie zum Beispiel ein Mobiltelefon, verzichtet werden, da alle relevanten Einstellungen im Optionen-Menu vorgenommen werden können.

Als Framework wurde hier das [Qt](#)[9] verwendet, da dieses bereits in der Version 5.12 im Buildroot-Framework hinterlegt ist. Somit musste kein anderes derartiges Framework aufwändig in das Buildroot-Framework integriert werden.

Das User-Interface wurde gegen Ende der Entwicklung der Controller-Software begonnen. Somit waren alle benötigten Ansichten und Funktionen definiert. Trotzdem wurden im Vorfeld bereits mögliche Ansichten und Menüstrukturen mittels Wireframing 8-5 festgehalten und konnten anhand dieser schnell umgesetzt werden.

Das [Qt](#) bietet dazu einen separaten Editor [Qt Design Studio](#) an, in den die zuvor erstellten Wireframe-Grafiken importiert wurden und anschliessend mit den Bedienelementen ersetzt werden konnten. Dieser Prozess gestaltete sich als sehr effizient und so konnte das komplette UI mit moderatem Zeitaufwand umgesetzt werden.

```

1 // WINDOW.qml User-Interface ATC
2 import QtQuick 2.15
3 import QtQuick.Controls 2.15
4 //...
5 Rectangle {
6     id: window
7     objectName: "window"
8     width: 800
9     height: 480
10    //BACKEND LOGIC INIT => CREATES INSTANCE OF THE
11    //      MenuManager CLASS
12    MenuManager{
13        id:main_menu
14        objectName: "mainmenu"
15    }
16}

```

```

15     //...
16     // MAIN MENU CONTAINER
17     Rectangle {
18         id: mm_container
19         objectName: "mm_container"
20         property var headline_bar_name:"Main Menu"
21         //START AI MATCH BUTTON
22         Button {
23             id: mm_start_random_btn
24             x: 40
25             y: 183
26             width: 207
27             height: 55
28             text: qsTr("START AI MATCH")
29             //CONNECT BUTTON EVENTS TO BACKEND LOGIC
30             Connections {
31                 target: mm_start_random_btn
32                 function onClicked(_mouse){
33                     //CALL A FUNCTION IN BACKEND LOGIC
34                     INSTANCE
35                     main_menu.
36                     mm_search_for_players_toggled(true)
37                 }
38             }
39         }
40     }
41 
```

Die anschließende Implementierung der Backend-Logik des Unter-Interfaces bestand in der Verbindung der in Qt Modeling Language (QML) erstellten Bedienelemente durch den [Qt Design Studio](#)-Editor mit der User-Interface Backend Logik. Diese beschränkt sich auf die Initialisierung des Fensters und das anschließende Laden und Darstellen des QML Codes. Die Backend-Logik Funktionalitäten sind in einem QML Typ [MenuManager](#) angelegt, welcher vor dem Laden des eigentlichen User-Interface QML-Codes registriert werden muss.

```

1 // main.cpp User-Interface ATC
2 #include <QGuiApplication>
3 #include <QQmlApplicationEngine>
4 #include "menumanager.h" //BACKEND LOGIC
5 int main(int argc, char *argv[])
6 {
7     QCoreApplication::setAttribute(Qt::AA_EnableHighDpiScaling);
8     //...
9     //CREATE WINDOW
10    QWindow window;
11    window.setBaseSize(QSize(800,480));
12
13    //REGISTER MainMenu COMPONENT
14    qmlRegisterType<MenuManager>("MenuManager",1,0,"MenuManager") 
```

```

        );
15 //LOAD User-Interface QML
16 QQuickView view;
17 //...
18 view.engine()->addImportPath("qrc:/qml/imports");
19 view.setSource(QUrl("qrc:/qml/WINDOW.qml"));
20 view.engine()->rootContext()->setContextProperty("app", &app
    );
21 //...
22 //IMPORTANT STEP: AFTER INIT THE MainMenu COMPONENT HAS NO
    PARENT
23 //SO WE NEED TO SET IT MANUALLY TO MAKE C++ -> QML FUNCATION
    CALLS WORKING
24 QObject *object = view.rootObject();
25 QObject *rect = object->findChild<QObject*>("mainmenu");
26 if (rect){
27     rect->setParent(object);
28 }
29 //FINALLY SHOW MENU ON SCREEN
30 view.show();
31 }

```

Da das User-Interface ein separates Programm ist, welches auf dem System ausgeführt wird, muss dieses in der Lage sein, mit der Controller-Software zu kommunizieren. Hierzu wurde die zuvor erstellte IPC Bibliothek in das Projekt importiert, jedoch wurde im Makefile das `USES_QT` Define-Flag gesetzt. Wenn dieses Flag gesetzt ist, wird die Bibliothek in den Client-Modus versetzt und stellt somit das Gegenstück zu der Instanz dar, welche in der Controller-Software läuft. Somit werden auch die Funktionen zum Senden von `gui.createEvent()` umgekehrt, sodass ein Event in der Controller-Software ausgelöst wird. Dies kann zum Beispiel durch eine Benutzereingabe geschehen oder wenn das User-Interface den von der Controller-Software geforderten Zustand angenommen hat.

```

1 // menumanager.cpp User-Interface ATC
2 #include "menumanager.h"
3
4 MenuManager::MenuManager()
5 {
6     //START IPC THREAD
7     guiconnection.start_recieve_thread();
8     //...
9 }
10
11 //METHOD CALLED FROM QML ELEMENT ss_calboard_btn
12 void MenuManager::ss_calboard_btn(){
13     //SEND EVENT TO CONTROLLER SOFTWARE
14     guiconnection.createEvent(guicommunicator::GUI_VALUE_TYPE
        ::START_CALBOARD_PROC);

```

```
15 }
16
17 //PROCESSES EVENTS COMMING FROM THE INTER PROCESS
18 // COMMUNICATION AND SHOWS MENUS OR SET IMAGES/LABES
19 // MenuManager::updateProgress() CALLED BY SPERATE THREAD
20 void MenuManager::updateProgress()
21 {
22     //GET LATEST EVENT FROM IPC
23     const guicomunicator::GUI_EVENT ev = guiconnection.
24         get_gui_update_event();
25     if(!ev.is_event_valid){return;}
26     //PROCESS EVENTS
27     //SWITCH MAIN MENU REQUEST
28     if(ev.event == guicomunicator::GUI_ELEMENT::SWITCH_MENU){
29         switch_menu(ev.type);
30     }
31 }
```

## 9 Fazit

Zusammenfassend lässt sich feststellen, dass das Ziel der Arbeit erreicht wurde. Die Kernfrage, welche die Überprüfung der Ausführbarkeit inklusive Erstellung und Umsetzung eines eingebetteten Systems und einer Cloud-Infrastruktur umfasst, konnte abschließend positiv beantwortet werden.

Es wurde iterativ ein autonomer Schachtisch **9-1** entwickelt, welcher alle zuvor gestellten Anforderungen erfüllt. Die Positionen der Schachfiguren können mittels NFC-Tags in den Füßen der Figuren und eines NFC-Lesers unterhalb des Schachfelds umgesetzt werden. Die Mechanik zur Bewegung des NFC-Lesers und eines Magneten in dessen Mitte ermöglicht zudem durch gegenpolige Magnete in den Füßen der Figuren ein automatisches Bewegen der Figuren ohne manuelle Interaktion. Die Größe der Felder des Schachtisches ist so ausgelegt, dass Figuren ohne Kontraktionen aneinander vorbeigeführt und am Rand des Spielfeldes positioniert werden können, sofern sie aus dem Spiel ausgeschieden sind. Dadurch war eine kleinere Revision des Tisches nicht anwendbar, dennoch konnten mittels der größeren Dimensionen der letzten Revision diese Funktion und weitere, wie die Mechanik zur Bewegung, optimiert und adäquat umgesetzt werden.

Der Tisch verfügt über eine Stand-Alone Funktionalität, welche das Starten und Auswerten eines Spiels ohne Anbindung zum Internet oder zu externen Geräten wie Smartphones ermöglicht. Dennoch ist es über eine Verbindung zum Internet mittels eines Cloud-Services möglich, zusätzliche Funktionen wie das Spiel gegen andere reale Spieler oder gegen eine Schachlogik zu nutzen oder einen Livestream zum Darstellen von gegenwärtigen Spielen aufzubauen.

Anhand der Durchführung von iterativen Verbesserungen und einzelnen Revisionen des Modells lassen sich definierte Aussagen zu Fortschritten und Veränderungen des Gesamtsystems treffen. Der Schachtisch als eingebettetes System besteht aus einem modularen Aufbau, welcher sich im Verlauf der Iterationen durch das Anpassen einzelner Komponenten aufwerten ließ. Derartige Systeme erfordern in der Vorbereitung ein



**Bild 9-1:** Erstellten Prototypen

hohes Verständnis der Zusammenhänge von Komponenten, da spätere Veränderungen komplexer sind. Da hierbei in den verschiedenen Iterationen aber nur einzelne Modulgruppen verändert wurden, musste lediglich die Verbindung des jeweiligen Moduls neu berücksichtigt werden und keine gänzliche Aufwertung aller Komponenten erfolgen. Anders als Mehrzweksystemen, welche oftmals wenige einzelne Module beinhalten, konnte hier eine separierte Betrachtung der Komponenten erfolgen.

Nicht nur der Aufbau der Komponenten erfolgte modular, sondern auch das Erstellen aller Softwarekomponenten und die Konstruktion des Schachttischs. Letzteres führt zu einem einfachen Aufbau des Tisches, welcher auch von versierten oder motivierten Anwendern ausgeführt werden kann. Eine aussagekräftige Versuchsführung dazu liegt jedoch nicht vor. Zudem sind nahezu alle verwendeten Materialien mühelos erhältlich, nur 3D-Komponenten müssen separat gedruckt werden. Alle Ressourcen einschließlich der CAD -Modelle werden online zur Verfügung gestellt.

Die Bedienung des Systems mittels des verbauten Bildschirms ist ebenfalls auf eine simple Benutzerführung spezifiziert. Auch unerfahrene Spieler können ein Spiel beginnen, ohne eine differenzierte Einweisung erhalten zu haben. Diese Funktionalität wurde an einzelnen Probanden getestet, eine aussagekräftige Statistik liegt jedoch nicht vor.

Grundsätzlich ist festzuhalten, dass es sich beim Resultat der Arbeit um kein finalisiertes Produkt, sondern um einen strukturellen Prototyp handelt. Weitere Prüfungen wie Nutzungsstatistiken oder Sicherheitsprüfungen müssten durchgeführt werden, ehe der

Schachtisch als kommerzielles Produkt betrachtet werden kann. Auch fehlen diverse Langzeittests. Ein Betriebstest mit einer Dauer von 6 Stunden verlief ohne erkennbare Zwischenfälle, für aussagekräftige Verschleiß- und Fehlererkennungen bedarf es jedoch noch weiterer und längerer Untersuchungen.

Der Prototyp lässt sich jedoch mit kommerziell erhältlichen und open-source verfügbaren Schachttischen vergleichen. Das Ziel, alle wünschenswerten Funktionen und Implementationen dieser Tische in den Prototypen zu integrieren, konnte erfolgreich umgesetzt werden. Darüber hinaus wurde weitere Funktionalitäten eingegliedert, wie eine Stand-Alone Funktionalität oder eine Schnittstelle zum Erstellen weiterer Erweiterungen.

Das System und insbesondere der implementierte Cloud-Service sind online erreichbar und erweiterbar. Dies ermöglicht unter anderem das Bauen eines eigenen Tisches unter Verwendung des ATC-Systems, aber auch die Integration weiterer Komponenten. Erfahrene Entwickler können somit das Spiel beliebig ausweiten oder sogar andere Spiele ergänzen. Die für das Projekt entworfene Mechanik und Spielführung kann dementsprechend auch für diverse andere Tischbrettspiele wie zum Beispiel Mühle verwendet werden.

Neben diversen im Studium erlernten Fähigkeiten wurden im Laufe des Projekts noch diverse andere Leistungen erforderlich, wie die Erstellung einer Mechanik oder das Konstruieren von Komponenten, was das Aneignen von zusätzlichem Wissen erforderte. Die resultierende Mechanik ist ungeachtet dessen fehlerlos und nahezu spielfrei, welches einen reibunglosen Spielablauf ermöglicht.

Abschließend lässt sich feststellen, dass ein funktionstüchtiger Schachtisch konstruiert wurde, der basierend auf einem eingebetteten System und einer Cloud-Infrastruktur als gelungener Abschluss eines Informatikstudiums bezeichnet werden kann.

## **9.1 Persönliches Fazit**

Im Verlauf dieser Arbeit bin ich persönlich weit über mich hinausgewachsen und ich bin sehr stolz, mein fertiges Projekt präsentieren zu können. Ich konnte nicht nur theoretisch die Machbarkeit meines Konzepts beweisen, sondern auch einen physischen Prototypen erstellen.

Natürlich hat das System noch Schwächen und bedarf diverser Ergänzungen, bevor es als fertiges Produkt betrachtet werden kann. Dazu gehören auch diverse Untersuchungen wie ein Langzeittest oder schleifenbasierte Hardwareüberprüfungen. Es wäre von Vorteil gewesen, das System durch einen erfahrenen Schachspieler bewerten zu lassen oder Rückmeldung über die Struktur und Menüführung von Anfängern zu erhalten. Diese Option blieb aufgrund von Versammlungs- und Zusammentreffens-Reglementierungen während der COVID-19 Pandemie jedoch verwehrt. Für eine vollständige Analyse fehlen daher wichtige statistische Daten.

Doch genau solche Erkenntnisse geben mir einen Anreiz, das Projekt auch nach Abschluss des Studiums nicht zu beenden, sondern weitere Ideen umzusetzen und anschließend auch mit Testpersonen zu erproben.

Der iterative Prozess der Erstellung des Schachtisches ist zeitaufwändig und kostspielig, ermöglicht allerdings das frühzeitige Erkennen von Schwachstellen und das Ausbessern der jeweiligen modularen Komponente, noch eher eine Etablierung im System erfolgen kann. Ebenso ermöglicht es ein einfacheres Verständnis über alle Komponenten, lediglich die Zusammensetzung und Beziehung dieser muss rechtzeitig überdacht werden.

Insgesamt ist das Projekt selbst recht umfangreich und umfasst in verschiedenen Facetten diverse Themenbereiche meines Studiums, was mir von Beginn an ein Anliegen war. Es manifestiert meinen Studienschwerpunkt, die technische Informatik, und hat mich dazu motiviert, noch tiefgründiger in die Materie zu schauen. Zudem waren noch weitere Kompetenzen gefordert, welche zuvor gar nicht oder nur teilweise gegeben waren, wie das Konstruieren von 3D-Komponenten oder das Gestalten von Produkten. Umso beeindruckter bin ich selbst von der Bewegungsmechanik des Systems, welche sich im Entwicklungsprozess sehr stark verändert hat und letztlich zu einem fehlerlosen Resultat führte.

Neben diesen projektspezifischen Kompetenzen ist es zudem möglich gewesen, weitere Erfahrungen im Bereich der Projektplanung und Organisation zu sammeln. Im optimalen Verlauf wäre ein fertiger Prototyp bereits früher möglich gewesen, jedoch erforderte die Veränderung der Mechanik vom XY-System zu CoreXY und verschiedene unvorhergesehene Schwächen mit den verwendeten Magneten weitere Umsetzungsiterationen, die rückblickend nötig und zielführend waren.

Auch die Erstellung der Thesis und das Formatieren von Vorlagen und Befehlen in Markdown und LaTeX erwies sich als lehrreich und innovativ.

Insgesamt bin ich sehr zufrieden mit meinem Ergebnis, aber auch mit mir selbst und meiner erbrachten Leistung.

## **9.2 Ausblick**

Wie jedes Produkt und insbesondere jeder Prototyp nach Abschluss eines Projekts lassen sich auch hierbei Aussagen zu Verbesserungen und Ergänzungen tätigen.

In erster Linie besteht die Möglichkeit, diverse Modifikationen des traditionellen Schachs in das System zu integrieren. Diese Option wurde im Laufe des Projekts bewusst ausgelassen. Doch auch Erweiterungen auf diverse andere Brettspiele, wie beispielsweise Mühle, Dame oder Mensch-ärgere-dich-nicht wären optional möglich. Mittels eines bildgebenden Systems oberhalb des Tisches oder Lichtelementen und Strukturen unterhalb einer durchsichtigen Tischplatte wäre es möglich, unterschiedliche Strukturen anzuzeigen und als Spielbrett darzustellen.

Auch weitere Hardware kann mittels Porterweiterungen wie USB zum Tisch ergänzt werden. So wäre zum Beispiel eine Schachuhr optional, welche bei öffentlichen Turnieren erforderlich ist. Eine Uhr würde zudem erkenntlich machen, wann ein Spieler den eigenen Zug beendet hat. Derzeit geschieht dies über eine Nutzereingabe oder bei Ablauf eines Timers.

Zudem ermöglicht die Verbindung zum Internet unzählige Kapazitäten, wie die Einbindung anderer, existierender Schach-Clouds wie beispielsweise „[licchess.org](#)“.

Ebenso könnten Sprachassistenten ergänzt werden, die das Spiel somit auch für körperlich eingeschränkte Personen möglich macht. Diese könnten zudem auch über aktuelle Zustände auditiv informieren oder weitere Befehle berücksichtigen.

Der derzeitige Systemzustand ermöglicht ein autonomes Schachspiel mit möglichst äquivalentem Spielerlebnis im Vergleich zu einem konventionellen Schachspiel. Auf Basis dessen ist der Anzahl der möglichen Ergänzungen und Ideen zur Verbesserung keine Grenzen gesetzt. Erfahrene Entwickler wird die Möglichkeit geboten, diese mittels der diversen Schnittstellen eigenständig zu integrieren. Das System als solches ist dennoch ein abgeschlossenes Projekt mit einem präsentablen Resultat.

# Literaturverzeichnis

- [1] ASSOCIATION, Buildroot: *Buildroot Making Embedded Linux Easy*. <https://buildroot.org>. Version: 28.03.2021
- [2] ATMEL: *ATmega328P 8-bit AVR Microcontroller with 32K Bytes ISP Flash*. [https://ww1.microchip.com/downloads/en/DeviceDoc/Atmel-7810-Automotive-Microcontrollers-ATmega328P\\_Datasheet.pdf](https://ww1.microchip.com/downloads/en/DeviceDoc/Atmel-7810-Automotive-Microcontrollers-ATmega328P_Datasheet.pdf). Version: 28.03.2021
- [3] DGT: *DGT Bluetooth Wenge*. <https://www.topschach.de/bluetooth-wenge-eboard-figuren-p-3842.html>. Version: 28.03.2021
- [4] DGT: *DTG Smart Board*. <https://www.topschach.de/smart-board-p-3835.html>. Version: 28.03.2021
- [5] DOCKER, Inc: *Docker*. <https://docker.com>. Version: 28.03.2021
- [6] DOCKER, Inc: *Docker-Compose*. <https://docs.docker.com/compose/>. Version: 28.03.2021
- [7] FIEKAS, Niklas: *Python-Chess Library*. <https://github.com/niklasf/python-chess>. Version: 28.03.2021
- [8] FOUNDATION, Linux ; PROJECT, Yocto: *THE YOCTO PROJECT*. <https://www.yoctoproject.org>. Version: 28.03.2021
- [9] GROUP, Qt: *Qt*. <https://www.qt.io>. Version: 28.03.2021
- [10] GUERERO, Michael: *Automated Chess Board*. <https://create.arduino.cc/projecthub/maguerero/automated-chess-board-50ca0f>. Version: 28.03.2021

- [11] INC., Infivent: *Square Off - Chess App.* <https://apps.apple.com/app/square-off-chess/id1267805783>. Version: 28.03.2021
- [12] INC., SquareOff: *Grand Kingdom Set.* <https://squareoffnow.com/product/gks>. Version: 28.03.2021
- [13] INC., SquareOff: *Kingdom Set.* <https://squareoffnow.com/product/kds>. Version: 28.03.2021
- [14] INSTRUMENTS, Texas: *DRV8833 Dual H-Bridge Motor Driver.* [https://www.ti.com/lit/ds/symlink/drv8833.pdf?ts=1622999216265&ref\\_url=https%253A%252F%252Fwww.bing.com%252F](https://www.ti.com/lit/ds/symlink/drv8833.pdf?ts=1622999216265&ref_url=https%253A%252F%252Fwww.bing.com%252F). Version: 28.03.2021
- [15] JUST, Tim: *US CHESS FEDERATION'S OFFICIAL RULES OF CHESS - Free Version.* [https://www.peoriachess.com/files/US\\_Chess\\_Rule\\_Book\\_Chapters\\_1\\_2\\_11\\_v7.0.pdf](https://www.peoriachess.com/files/US_Chess_Rule_Book_Chapters_1_2_11_v7.0.pdf). Version: 04.04.2016
- [16] MACHINES, DIY: *DIY Super Smart Chessboard.* <https://www.instructables.com/DIY-Super-Smart-Chessboard-Play-Online-or-Against-/>. Version: 28.03.2021
- [17] McEVOY, Brian: *PRINT CHESS PIECES, THEN DEFEAT THE CHESS-PLAYING PRINTER.* <https://hackaday.com/2021/03/06/print-chess-pieces-then-defeat-the-chess-playing-printer/>. Version: 28.03.2021
- [18] McEVOY, Brian: *Ultimaker Cura Slicer.* <https://github.com/Ultimaker/Cura>. Version: 28.03.2021
- [19] NON-PROFIT-ORGANISATION, Let's E.: *Let's Encrypt.* <https://letsencrypt.org/de/>. Version: 28.03.2021
- [20] NXP: *AN133910.* <https://www.nxp.com/docs/en/nxp/application-notes/AN133910.pdf>. Version: 28.03.2021
- [21] NXP: *NXP NTAG 21.* <https://www.nxp.com/products/rfid-nfc/nfc-hf/ntag-for-tags-labels/>. Version: 28.03.2021

- [22] OPENJS FOUNDATION, jQuery c.: *jQuery*. <https://jquery.com>. Version: 28.03.2021
- [23] RAVICHANDRAN, Akash: *Automated Chess Board*. <https://create.arduino.cc/projecthub/automaters/automated-chess-5dbd7a>. Version: 28.03.2021
- [24] SALVATORE SANFILIPPO, Redis c.: *Redis*. <https://redis.io>. Version: 28.03.2021
- [25] SEMICONDUCTOR, Nordic: *NFC library and modules: Generating messages and records*. [https://infocenter.nordicsemi.com/index.jsp?topic=%2Fcom.nordic.infocenter.sdk5.v12.0.0%2Fnfc\\_ndef\\_format\\_dox.html](https://infocenter.nordicsemi.com/index.jsp?topic=%2Fcom.nordic.infocenter.sdk5.v12.0.0%2Fnfc_ndef_format_dox.html). Version: 28.03.2021
- [26] SEMICONDUCTOR, Nordic: *NFC library and modules: NDEF message and record format*. [https://infocenter.nordicsemi.com/index.jsp?topic=%2Fcom.nordic.infocenter.sdk5.v12.0.0%2Fnfc\\_ndef\\_format\\_dox.html](https://infocenter.nordicsemi.com/index.jsp?topic=%2Fcom.nordic.infocenter.sdk5.v12.0.0%2Fnfc_ndef_format_dox.html). Version: 28.03.2021
- [27] STEFAN-MEYER KAHLEN, ShredderChess: *UCI protocol*. <http://wbec-ridderkerk.nl/html/UCIProtocol.html>, <https://www.shredderchess.com/download/div/uci.zip>. Version: 28.03.2021
- [28] TORD ROMSTAD, Joona K. Marco Costalba C. Marco Costalba: *Stockfish - Strong Open Source Chess Engine*. <https://stockfishchess.org/>. Version: 28.03.2021
- [29] TRINAMIC: *TMC5160 / TMC5160A DATASHEET*. [https://www.trinamic.com/fileadmin/assets/Products/ICs\\_Documents/TMC5160A\\_Datasheet\\_Rev1.14.pdf](https://www.trinamic.com/fileadmin/assets/Products/ICs_Documents/TMC5160A_Datasheet_Rev1.14.pdf). Version: 28.03.2021
- [30] V., Deutsche I. e.: *DIN EN ISO 9241-220 Ergonomie der Mensch-System-Interaktion - Teil 220: Prozesse zur Ermöglichung, Durchführung und Bewertung menschzentrierter Gestaltung für interaktive Systeme in Hersteller- und Betreiberorganisationen*. <https://www.din.de/de/mitwirken/normenausschusse/naerg/veroeffentlichungen/wdc-beuth:din21:289443385>. Version: 04.04.2021
- [31] YOUSIFNIMAT: *Chess Robot*. <https://www.instructables.com/Chess-Robot/>. Version: 28.03.2021
- [32] ZALM, Erik van d.: *Marlin Firmware*. <https://marlinfw.org/>, <https://github.com/MarlinFirmware/Marlin>. Version: 28.03.2021



# Akronyme

**AI** Artificial Intelligence. 71, 78, 79

**AJAX** Asynchronous JavaScript and XML. 77

**API** Application Programming Interface. 63, 65, 69, 71, 75, 76, 79, 82

**ATC** Atomic Chess Table. 15, 61, 63, 97

**CAD** Computer-Aided Design. 25, 27, 51, 54, 96

**CNC** Computerized Numerical Control. 48, 52, 54

**CSS** Cascading Style Sheets. 77

**DHCP** Dynamic Host Configuration Protocol. 17

**DSI** Display Serial Interface. 22, 23

**FDM** Fused Deposition Modeling. 25

**FEN** Forsyth-Edwards-Notation. 75, 76

**FIFO** First In – First Out. 89

**GDB** GNU Debugger. 18

**GPIO** General Purpose Input/Output. 22, 23, 32–34, 58

**GPL** General Public License. 12

## Akronyme

---

**GPU** Graphics Processing Unit. 23

**GUI** Graphical User Interface. 17, 23, 60, 79, 81, 82, 84

**HAL** Hardware Abstraction Layer. 34, 35, 37, 54, 56, 61

**HDMI** High Definition Multimedia Interface. 22

**HF** high frequency. 21

**HTML** Hypertext Markup Language. 77

**HTTP** Hypertext Transfer Protocol. 65, 77

**HTTPS** Hypertext Transfer Protocol Secure. 65, 69

**I<sub>2</sub>C** Inter-Integrated Circuit. 22, 33, 57, 58

**IC** Integrated Circuit. 29, 58

**ID** Identifikator. 35, 55, 66, 67

**IPC** Interprocess Communication. 88, 89, 93

**JS** JavaScript. 77

**JSON** JavaScript Object Notation. 66, 89

**LAN** Local Area Network. 22, 23

**LED** Light-Emitting Diode. 12, 32, 55, 60

**NDEF** NFC Data Exchange Format. 29, 30

**NDEF-RTD** NDEF Record Type Defintion. 30

**NFC** Near Field Communication. 21, 22, 29–32, 34, 36, 51, 57, 58, 62, 87

## Akronyme

---

**PCB** Printed Cirtcuit Board. 34

**PLA** Polylactic Acid. 25

**PWM** Pulse Width Modulation. 32

**QML** Qt Modeling Language. 92

**REST** Representational State Transfer. 63, 79

**SPI** Serial Peripheral Interface. 22, 23, 25, 31–34, 36, 37, 52

**SSH** Secure Shell. 17

**TCP** Transmission Control Protocol. 78, 89

**TLS** Transport Layer Security. 69, 70

**UCI** Universal Chess Interface. 79

**UDEV** userspace device file system. 17

**UI** User Interface. 87

**USB** Universal Serial Bus. 12, 15, 22, 23, 53, 55, 57, 58, 61, 99

**USCF** United States Chess Federation. 9, 16

**WLAN** Wireless Local Area Network. 12, 15, 22, 23, 61

**X-FEN** Extended Forsyth-Edwards-Notation. 75, 76, 79

# Abbildungsverzeichnis

4-1	Grove PN532 NFC Reader mit kabelgebundener Antenne . . . . .	21
4-2	3D Druck: Objekt (rot,gelb,grün),Tree Structure (cyan) . . . . .	26
5-1	Prototyp Hardware: Erster Prototyp des autonomen Schachtischs . . .	28
5-2	Zwei Elektromagnete. Schlitten befindet sich jeweils in den Ecken . . .	28
5-3	Prototyp Hardware: Tool zur Erstellung des NDEF Payloads: ChessFigureIDGenerator.html . . . . .	30
5-4	Prototyp Hardware: NDEF Text Record Payload für einen weißen Turm	31
5-5	Prototyp Hardware: Blockdiagramm . . . . .	32
5-6	Prototyp Hardware: Schaltplan und finaler PCB Entwurf . . . . .	33
5-7	Prototyp Hardware: Aufbau der Lochrasterplatine . . . . .	34
6-1	Production Hardware: Finaler autonomer Schachtisch . . . . .	45
6-2	Schachfiguren im Vergleich . . . . .	50
6-3	Production Hardware: Blockdiagramm . . . . .	53
7-1	Gesamtübersicht der verwendeten Cloud-Infrastruktur . . . . .	64
7-2	Cloud-Infrastruktur: Aufbau einer URI . . . . .	64
7-3	Cloud-Infrastruktur: Aufbau der Service Architecture . . . . .	66
7-4	Cloud-Infrastruktur: Backend Login-Request und Response . . . . .	69
7-5	MoveValidator: Beispiel Request zur Ausführung eines Zuges auf einem gegebenen Schachbrett . . . . .	75
7-6	Webclient: Spielansicht . . . . .	77
7-7	Webclient: Statistiken . . . . .	78
8-1	Embedded System Software: Buildroot Pakete . . . . .	82
8-2	Embedded System Software: Ablaufdiagramm . . . . .	83
8-3	Embedded System Software: Figur Wegpunkte . . . . .	86
8-4	Embedded System Software: Schachfeld Scan Algorithmus Ablauf . . .	88
8-5	Embedded System Software: User-Interface Mockup . . . . .	91
9-1	Erstellten Prototypen . . . . .	96

# Tabellenverzeichnis

2.1 Auflistung kommerzieller autonomer Schachtische . . . . .	8
2.2 Auflistung von Open-Source Schachtisch Projekten . . . . .	12
3.1 Auflistung der Anforderungen an den autonomen Schachtisch . . . . .	15
4.1 Vergleich Yocto - Buildroot . . . . .	18
4.2 Verwendete 3D Druck Parameter. Temperatur nach Herstellerangaben des verwendeten PLA Filaments. . . . .	26
5.1 TMC5160 Beschleunigungskurve / RAMP Parameter . . . . .	38
6.1 Standardhardware 3D Drucker Steuerungen . . . . .	53
6.2 Grundlegende verwendete G-Code Kommandos . . . . .	55
6.3 Hinterlegte G-Code Steuerungen . . . . .	56
6.4 Hinterlegte Mikrocontroller . . . . .	59
6.5 Eigenschaften die finalen Prototypen . . . . .	61
7.1 MoveValidator-Service API Overview . . . . .	75
7.2 Vergleich FEN - X-FEN . . . . .	76