

FH Aachen

Fachbereich Elektrotechnik und Informationstechnik

Studiengang Informatik

Bachelorarbeit

Integration eines eingebetteten Systems in eine Cloud-Infrastruktur am Beispiel eines autonomen Schachtischs

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

Die Kurzfassung gibt auf ein bis zwei Seiten die wesentlichen Inhalte und Ergebnisse der Abschlussarbeit wieder.

Sie gliedert sich inhaltlich in

- das behandelte Gebiet,
- das Zieler Arbeit,
- die Untersuchungsmethode,
- die Ergebnisse und
- die Schlussfolgerungen.

Die Kurzfassung enthält keine Schlussfolgerungen oder Bewertungen, die über die Inhalte der Kapitel der Arbeit hinausgehen.

Alle Aussagen der Kurzfassung finden sich in ausführlicher Form in der Arbeit wieder.
Die Kurzfassung erhält keine Kapitelnummer.

Inhalt

1 Einleitung	1
1.1 Motivation	1
1.2 Zielsetzung	3
1.3 Methodik	3
2 Analyse bestehender Systeme	6
2.1 Existierende Systeme im Vergleich	6
2.1.1 Kommerzielle Produkte	7
2.1.2 Open-Source Projekte	10
2.2 User Experience	12
3 Anforderungsanalyse	14
4 Machbarkeitsanalyse	15
5 Grundlegende Verifikation der ausgewählten Technologien	16
5.1 Erprobung Buildroot-Framework	16
5.2 Verifikation NFC Technologie	16
5.3 Schrittmotor / Schrittmotorsteuerung	16
5.4 3D Druck für den mechanischen Aufbau	17
6 Erstellung erster Prototyp	19
6.1 Mechanik	19
6.2 Parametrisierung Schachfiguren	21
6.3 Schaltungsentwurf	23
6.4 Implementierung HAL	26
6.4.1 TMC5160 SPI Treiber	29
6.5 Fazit bezüglich des ersten Prototypens	33
7 Aufbau des zweiten Prototypen	35
7.1 Modifikation der Mechanik	35
7.1.1 Gehäuse und Design	35
7.1.2 3D-Komponenten	38

7.1.3	Positions-Mechanik	39
7.2	Optimierungen der Spielfiguren	41
7.3	Änderungen der Elektronik	42
7.4	Anpassungen HAL	44
7.4.1	Implementierung GCODE-Sender	44
7.4.2	I2C-Seriell Umsetzer	47
7.5	Fazit bezüglich des finalen Prototypens	50
8	Entwicklung der Cloud Infrastruktur	52
8.1	API Design	52
8.2	Service Architektur	54
8.3	Service: Backend	58
8.4	Service: MoveValidator	62
8.5	Service: Webclient	65
8.6	Service: AutoPlayer	66
9	Embedded System Software	69
9.0.1	Anmerkungen Compiler	69
9.1	Ablaufdiagramm	69
9.2	Figur Bewegungspfadberechnung	71
9.3	Schachfeld Scan Algorithmus zur Erkennung von Schachzügen	74
9.4	Inter Prozess Communication	76
9.5	Userinterface	78
10	Fazit	82
10.1	Persönliches Fazit	82
10.2	Ausblick	82
Literaturverzeichnis		83
Abbildungsverzeichnis		89
Tabellenverzeichnis		90
A	Anhang	91

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 simplere und auch Ressourcen-sparendere 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 Funktionen 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 Merhzwecksystems ist die Auswahl einfacher, da man den Prozess auch im Nachhinein noch anpassen kann, weil 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. Fertiggestellte Systeme sind komplexer in der Aufwertung.

Die Fähigkeit zur Erstellung eines solchen System ist daher nicht leichtfertig anzunehmen und es ist mir wichtig, zum Abschluss meines Studiums mein gewonnenes Wissen über Systeme, Komponenten, Zusammenhänge und deren Verbindung 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.

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.

Der Reiz im Schachprojekt liegt in der Bedeutung und der Seltenheit. 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 erhebliche Anpassungen des Spielers an das Spiel. Innerhalb der vergangenen drei Jahrzehnte bewiesen sich immer mehr Konzerne ihre technische Kompetenz und Überlegenheit und die Fähigkeit ihrer Maschinen mittels der Auswertung von Schachalgorithmen und dem möglichst schnellen besiegen derzeitiger Schach-Meister und -Meisterinnen. Die Algorithmen stehen heute in einer Vielzahl als open-source Projekte 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 Spieler zu ermöglichen. Das Ergebnis soll nicht nur viele Zeilen Code sein, sondern auch ein handfestes Produkt, dass meine Qualitäten und Enthusiasmus widerspiegeln.

1.2 Zielsetzung

Das Ziel der nachfolgenden 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
- Spiel Livestream
- Parkposition für ausgeschiedene Figuren
- Stand-Alone Funktionalität

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. Dieses besteht zum einem aus der Positionserkennung und Steuerung der Hardwarekomponenten (Schachfiguren) und zum anderen aus der 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.

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 einfach Bedienoberfläche eine neue Partie gegen den Computer oder einen anderen Menschlichen spieler beginnen zu können.

1.3 Methodik

Im zweiten Kapitel 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.

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 Mehrweiter in Bezug auf den 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 optimaler Weise 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, wurde ein Webclient entwickelt, mit dem es Benutzern möglich ist über einen Webbrowser gegen den Tisch zu spielen. Dieser Client bietet außerdem die Möglichkeit das System während der

Entwicklung testen zu können.

Das neunte Kapitel beschäftigt sich mit der Software, welche auf dem eingebetteten System ausgeführt wird. Diese übersetzt die Spieldaten welche von der Cloud-Infrastruktur abgefragt werden in Zug-Befehle welche von der Mechanik umgesetzt werden. Dabei gilt ein besonderes Augenmerk der Berechnung der Figurbewegung und dem Erkennen von durch den Benutzer getätigten Schachzügen.

Das zehnte und abschliessende Kapitel, befasst sich mit dem 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
 - Square Off - Grand Kingdom
 - DGT Smartboard
 - DGT Bluetooth Wenge
- open-source:
 - Automated Chessboard (Michael Guerero)
 - Automated Chessboard (Akash Ravichandran)
 - DIY Super Smart Chessboard

Für die kommerziell käuflichen Schachspiele 2.1 gibt es kein sehr großes Marktangebot, weswegen für den Vergleich nur zwei Hersteller mit jeweils zwei verschiedenen Modellen gewählt werden konnte. (Derzeit integriert nur das Unternehmen [Square Off](#) eine Funktion, welche die Figuren unterhalb der Tischplatte mechanisch bewegen kann.)

Der zweite Hersteller [DGT](#) wurde dennoch zum Vergleich von zusätzlichen Funktionen herangezogen, da dessen Schachbretter die aktuelle Figurstellungen erkennen können.

Die Tische des Herstellers [DTG](#) 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 Projekten 2.2 hingegen ist signifikanter, 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 zum Teil identischen den Namen ersichtlich, streben alle Tische das gleiche Ziel an und unterscheiden sich daher nur in geringen Funktionen, was im Folgenden nun näher erläutert wird.

2.1.1 Kommerzielle Produkte

Tabelle 2.1: Auflistung kommerzieller autonomer Schachtische

	Square Off - Square Off - Kingdom [8]	Grand Kingdom [7]	DGT Smart Board [2]	DGT Bluetooth Wenge [1]
Erkennung Figur-Stellung	nein (Manuell per Ausgangsposition) Ausgangsposition)	nein (Manuell per Ausgangsposition)	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 (Mobiltelefon + App)	ja (Mobiltelefon + App)	ja (PC + App)	ja (PC + App)
Parkposition für ausgeschiede- ne Figuren	nein	ja	nein	nein
Stand-Alone Funktionalität	nein (Mobiltelefon erforderlich)	nein (Mobiltelefon erforderlich)	nein (PC)	nein (PC)

	Square Off - Kingdom [8]	Grand Kingdom [7]	DGT Smart Board [2]	DGT Wenge [1]
Besonderheiten	Akku für 30 Spiele	Akku für 15 Spiele	-	-

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

Die Bretter des Herstellers DGT erkennen die Position der verwendeten Figuren; eine Auskunft, über die die verwendete Technologie erhält, man jedoch nicht. Die Square-Off-Schachtische verfügen über keine solche Funktion.

Die Abmessungen 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 wiederum ist kleiner als das vorgegebene Turniermaß, was ebenfalls 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 auffallend.

Alle Hersteller bieten eine Bluetooth-Schnittstelle an, einzige 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 diese 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 z.B. Schachuhren.

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

Ein Spiel-Livestream, eine Darstellung die aktuellen oder vergangenen 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 Computer, 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, welche Figuren automatische 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, welche die Berechnungen der Spielerzüge 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](#)[6] für die Verwendung notwendig. Nach einer Analyse der Companion-App, ist zu erkennen, dass hier eine Registrierung inkl Profilerstellung notwendig ist um mit der Verwendung der App forfahren zu können. Erst danach ist ein Spiel gegen den Computer ohne Internet möglich. Alle weiteren Optionen (Spiel gegen andere Spieler, Live-Stream) ist nur über einen Online-Zugang möglich und erfordert je nach gewählter 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, z.B. 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 der Auswertung von gewünschten und gegebenen Funktionen. Dadurch dass nur die Firma [Square Off](#) einen wirklich autonomen Schachtisch anbietet, auch wenn dieser nicht alle angestrebten Funktionalitäten bietet. So hat der Nutzer kaum Auswahlmöglichkeiten auf der kommerziellen Seite.

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 hinzugefügt werden. Alle Eigenschaften der Projekte wurden zum Zeitpunkt der Recherche analysiert und dokumentiert und mit Beginn der Entwicklung als Struktur-Fixpunkt festgelegt. Nachfolgende Entwicklungen werden zu 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 unterscheiden.

Auch existieren weitere Abwandlungen von autonomen Schachbrettern, bei welchem die Figuren von oberhalb des Spielbretts gegriffen bzw. bewegt werden. In einigen Projekten wird dies mittels eines Industrie-Roboters [20] oder eines modifizierten 3D-Druckers[10] 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) [5]	Automated Chess Board (Akash Ravichandran) [14]	DIY Super Smart Chessboard [9]
Erkennung	nein (Manuell per Ausgangsposition)	ja (Kamera / OpenCV)	nein
Figur-Stellung			
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

	Automated Chess Board (Michael Guerero) [5]	Automated Chess Board (Akash Ravichandran) [14]	DIY Super Smart Chessboard [9]
Parkposition für ausgeschiedene Figuren	nein	nein	nein
Stand-Alone Funktionalität	nein (PC erforderlich)	ja	ja
Besonderheiten	-	Sprachsteuerung (Amazon Alexa)	Zuganzeige über 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-Achse verwendet, welche von je einem Schrittmotoren bewegt werden. Die Schachfiguren werden dabei mittels eines Elektromagneten über die Oberseite gezogen. Indes ist ein Magnet oder eine kleine Metallplatte als Gegenpol in den Fuß der Figuren eingelassen worden.

Die Erkennung der Schachfiguren ist augenscheinlich die schwierigste Aufgabe. Hier wurde in der Mehrzahl der Projekte eine Kamera im Zusammenspiel mit einer auf OpenCV basierenden Figur-Erkennung verwendet. Diese Variante ist je nach Implementierung des Vision-Algorithmus fehleranfälliger 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 Figur Typen. Dieses Problem wird durch eine definierte Ausgangsstellung beim Spielstart gelöst. Nach jedem Zug durch den Spieler und der dadurch resultierenden Änderungen in der Figur Positionen in der Matrix können die neuen Figur Stellungen berechnet werden.

Jedoch ist abschließend zu festzuhalten dass es auch bei den open-source Projekten kein Projekt gibt, welches alle gewünschten Features abbildet. Auch fehlen weitestgehend Features, welche die kommerziellen Projekte bieten. Das Ziel soll nun sein, all die positiven Eigenschaften dieser Tische zu vereinbaren 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[19] beschrieben. Darin geht es primär um das Erlebnis, welches der Benutzer bei dem Verwenden eines Produktes erlebt und welche Erwartungen der Benutzer an die Verwendung des Produktes hat.

Bei dem 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. Dieser bewegt die Figuren der Gegenseite.

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 einfach per Knopfdruck gestartet werden können sollte.

Dies soll auf für abgebrochene Spiele gelten, welche von Benutzer oder durch das System abgebrochen werden. Indessen soll das Schachbrett sich ebenfalls selbstständig

zurücksetzen können.

Ein weiter Punkt, welcher bei der User-Experience beachtet werden soll, ist die zeitliche Konstante. 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 z.B. ermöglichen ein Spiel am Morgen zu beginnen und dieses erst am nächsten Tag fortzusetzen.

Auch muss sich hier die Frage gestellt werden, was mit den ausgeschiedenen Figuren geschieht. Bei den autonomeren Schachbrettern von Square Off[7], werden die Figuren an die Seite auf vordefinierte Felder bewegt und können so wieder bei der nächsten Partie vom System aufgestellt werden. Viele andere Projekte schieben die Figuren auf dem Feld heraus, können diese aber im Anschluss nicht mehr gezielt in das Feld zurückholen. So muss diese Aufgabe vom Benutzer geschehen. Auch wir diese Funktionalität von einigen Projekten nicht abgedeckt und der Benutzer muss die Figuren selbständig 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 die 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 das Verhältnis zwischen den Spielfeldabmessungen und den Abmessungen des 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. Ebenfalls wird kein besonderes Augenmerk auf die Geschwindigkeit der Figur-Bewegung gelegt, da hier die Zuverlässigkeit und Wiederholgenauigkeit dieser im Vordergrund stehen.

4 Machbarkeitsanalyse

- welche technologien werden benötigt
- software architektur anforderungen
- hardware anforderungen
- grosse
- wiederholgenauigkeit
- lautstärke
- vorerfahrung in cad ed druck und schaltungsdesign

5 Grundlegende Verifikation der ausgewählten Technologien

5.1 Erprobung Buildroot-Framework

- Erstellen eines einfachen images für das embedded System
- inkl ssh Server und SFTP
- qt 5 libraries
- eigenes package atctp
- test der toolchain

5.2 Verifikation NFC Technologie

- warum gewählter nfc reader => ndef lesen
- reichweiten test mit 22mm
- test mit benachbarten figuren
- warum kein RFID => keine speicherung von id auf der controller seite
- selbherstellung von eigenen figuren ohne modifikation der controllerseite
- test mit figuren nebeneinander

5.3 Schrittmotor / Schrittmotorsteuerung

- warum => einfache ansteuerung

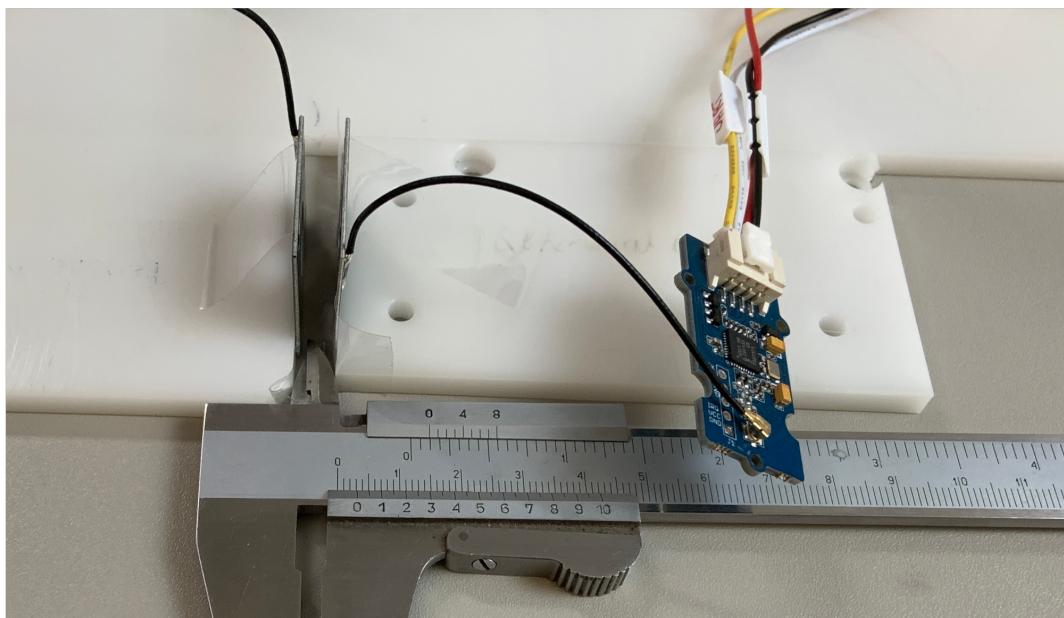


Bild 5-1: Grove PN532 NFC Reader mit Kabelgebundener Antenne

- keine STEP DIR somit muss embedded nicht echtzeitfähig sein und kann ggf auch andere task abarbeiten
- TMC schrittmotortreiber spi configuration
- und goto move => wait for move finished irw testen
- dafür einfacher python testreiber geschribene
- schrittverlust nicht zu erwarten

5.4 3D Druck für den mechanischen Aufbau

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

Zuvor wurden einige Testdrucke durchgeführt, um die Qualität der zuvor gewählten Druckparameter 5.1 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 können. Dies betrifft vor allem die Genauigkeit der Bohrungen in den gefertigten Objekten, da hier später Bolzen und Schrauben ein nahezu spielfrei eingeführt werden müssen. Ein Test, welcher die Machbarkeit von Gewinden zeigt, wurde nicht durchgeführt, da alle Schrauben später

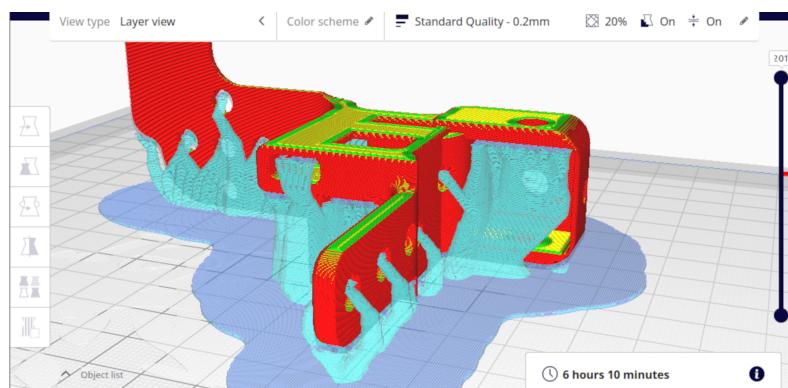


Bild 5-2: 3D Druck: Objekt (rot,gelb,grün),Tree Structure (cyan)

mit der passenden Mutter gesichert werden sollen. So soll eine Abnutzung durch häufige Montage der gedruckten Bauteile verhindert werden.

Bei dem 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 [11] verwendet, da dieser zum einen 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 bereitgestellte Tree Support Structure aktiviert. **5-2** Diese bietet den Vorteil gegenüber anderen Stützstrukturen, dass sich diese leichter entfernen lässt und weniger Rückstände an den Bauteilen hinterlässt. Diese Vorteile wurde mit verschiedenen Testdrucken verifiziert und kommen insbesondere bei komplexen Bauteilen mit innenliegenden Elementen zum Tragen, bei denen eine Stützstruktur erforderlich sind.

Tabelle 5.1: Verwendete 3D Druck Parameter. Temperatur nach Herstellerangaben des verwendeten PLA Filament.

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, sind hierbei individuell für den zu gewählten 3D Drucker zu ermitteln. Allgemein wurden hier die Standardeinstellungen verwendet, welche in diesem Falle einen guten Kompromiss zwischen Qualität und Druckzeit lieferten.¹⁸

6 Erstellung erster Prototyp

6.1 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 möglich ist. Durch den bereits vorhandenen Rahmen ist es simpel 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 hatte. Bei den Schlitten, welche auf den Aluminium-Profilen laufen, wurden fertige Standartkomponenten 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 müssen.

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

6 Erstellung erster Prototyp



Bild 6-1: Prototyp Hardware: Erster Prototyp des autonomen Schachtisch

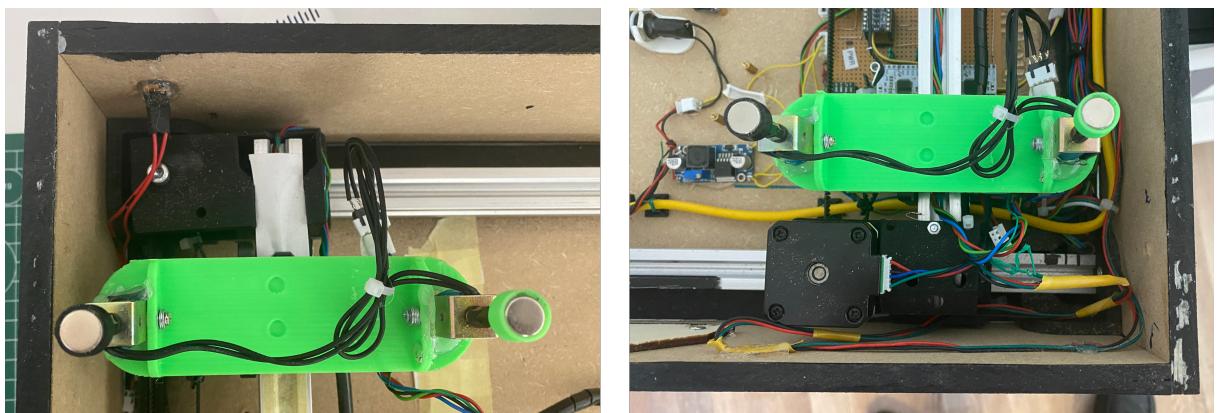


Bild 6-2: Zwei Elektromagnete. Schlitten befindet sich jeweils in den Ecken

durch die Verkleinerung der Halterungen der Aluminium-Profile dieser nicht ausreicht. Mit dieser Konstellation können die Figuren nicht ausreichend weit aus dem Spielfeld platziert werden und verbleiben in den äußeren Spielfeldern. Dieser Effekt war unerwünscht und schränkt 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 Elektromagnet erweitert. Diese befinden sich nun nicht mehr mittig auf dem Schlitten, sondern wurden um 110mm in Richtung der X-Achse versetzt 6-2. So ist 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 zeitlich kürzeren Stillstand der Figur auf dem Schachfeld.

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 stellt jedoch eine solide Grundlage dar, welche für die weitere Software und Hardware-Entwicklung essentiell ist.

6.2 Parametrisierung Schachfiguren

Da das System die auf dem Feld befindlichen Schachfiguren anhand von Near Field Communication (NFC) Tags erkennt, müssen diese zuerst mit Daten beschrieben werden. Die verwendeten NXP [NTAG 21](#)[13] Integrated Circuit (IC), besitzen einen vom Benutzer verwendbaren Speicher von 180 Byte. 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 diesen 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 auf diese formatiert werden. Die meisten käuflichen Tags sind bereits derart formatiert. Alternativ kann dies mittels Mobiltelefon und passender Applikation geschehen. Da NDEF Informationen über die Formatierung und der gespeicherten Einträge speichert, stehen nach der Formatierung nur noch 137 Bytes des NXP NTAG 21 zur Verfügung.

Per Lesegerät können anschließend mehrere NDEF Records auf den Tag geschrieben werden. Diese sind mit Dateien auf einer Festplatte vergleichbar und können verschiedenen Dateiformaten und Dateigrößen annehmen. Ein typischer Anwendungsfall ist der 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. [16]

SETTINGS

FIGURE TYPE

KING QUEEN ROOK BISHOP KNIGHT PAWN

FIGURE COLOR

BLACK 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 6-3: Prototyp Hardware: Tool zur Erstellung des NDEF Payloads: ChessFigureID-Generator.html

Der autonome Schachtisch verwendet den einfachsten NDEF-RTD Typ, den sogenannten Text-Record, welcher zum Speichern von Zeichenketten genutzt werden kann, ohne das eine Aktion auf dem Endgerät ausgeführt wird. Jeder Tag einer Schachfigur, welche für den autonomen Schachtisch verwendet werden kann, besitzt diesen NDEF Record 6-4 an der ersten Speicher-Position. Alle weiteren eventuell vorhandenen Records werden vom Tisch ignoriert. [15]

Um die Payload für den NFC Record zu erstellen wurde ein kleine Web-Applikation 6-3 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.

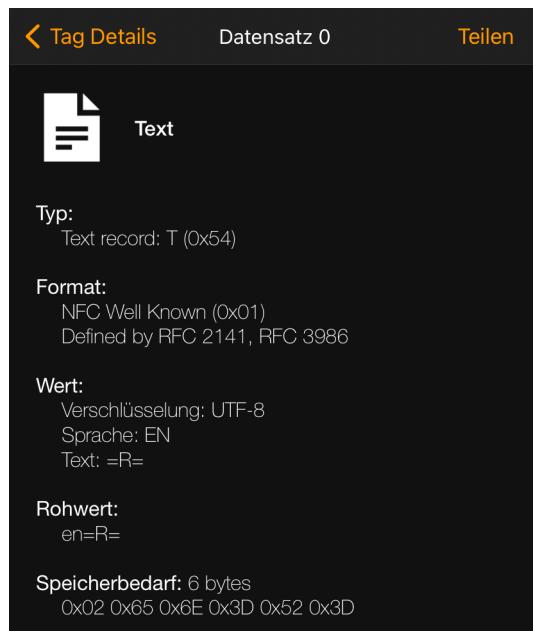


Bild 6-4: Prototyp Hardware: NDEF Text Record Payload für einen weißen Turm

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

6.3 Schaltungsentwurf

Durch die zuvor durchgeführte Validierung der verwendeten Technologien, konnte ein Blockdiagramm 6-5 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-BOB](#). Diese sind direkt über einen Serial Peripheral Interface (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

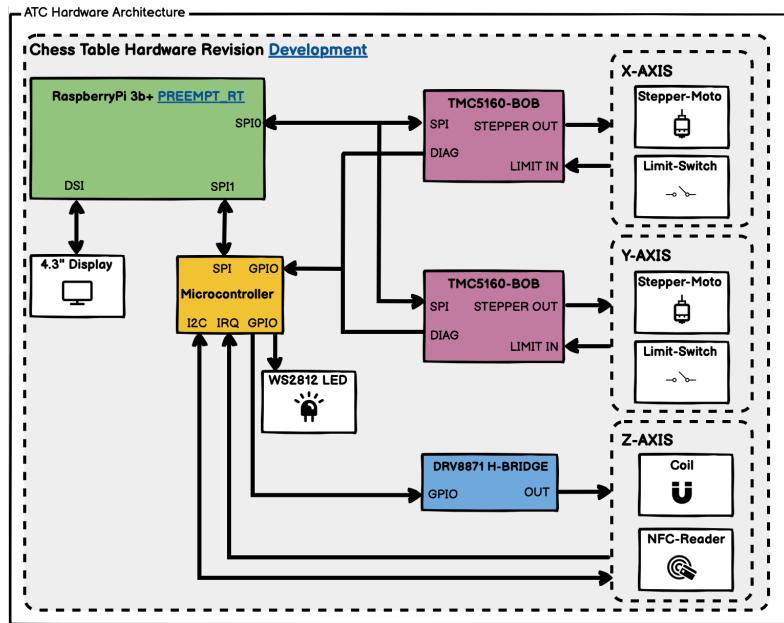


Bild 6-5: Prototyp Hardware: Blockdiagramm

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 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 auf maximale Flexibilität 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, sondern nur die Zustandsinformationen über den SPI Bus übertragen werden sollen. Dies spart zusätzliche General Purpose Input/Output (GPIO)

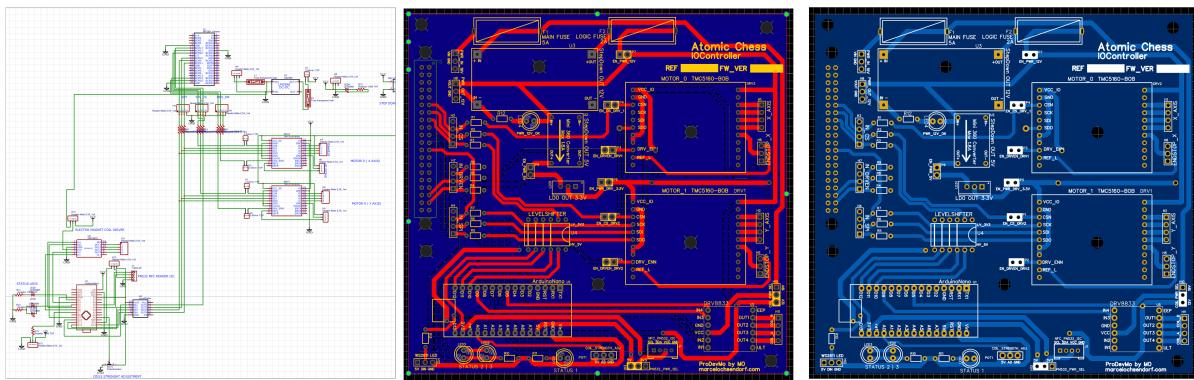


Bild 6-6: Prototyp Hardware: Schaltplan und finaler PCB Entwurf

Anschlüsse und somit sind alle Komponenten über einen zentralen Bus kontrollierbar, welches einen möglichen Tausch des eingebetteten Systems in späteren Revisionen vereinfacht.

Der zusätzliche Mikrokontroller übernimmt auch die Kommunikation mit dem [PN532](#) Modul, da dieses sonst über seine Inter-Integrated Circuit (I2C) Schnittstelle mit einem entsprechenden Host-System kommuniziert. Der Mikrokontroller übernimmt somit ebenfalls die Konversation 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 Mikrokontroller ausgeführt wird, realisiert, und je nach empfangenem Kommando die entsprechende Komponente ausgewählt.

Nach der Festlegung der zu verwendenden Komponenten, wurde ein entsprechender Schaltplan **6-6** nach den zuvor erörterten Vorgaben entworfen. Hierbei wurde die Vorgaben der Datenblätter und der Application-Notes in diesen Orientiert. Da es sich hier um einen ersten Funktionsentwurf handelt, wurde zusätzliche Testpunkte in das Design eingefügt.

Somit ist 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 ist 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 Levelshifter eingesetzt, welcher den SPI Bus des eingebetteten Systems mit dem der Mikrokontroller trennt.

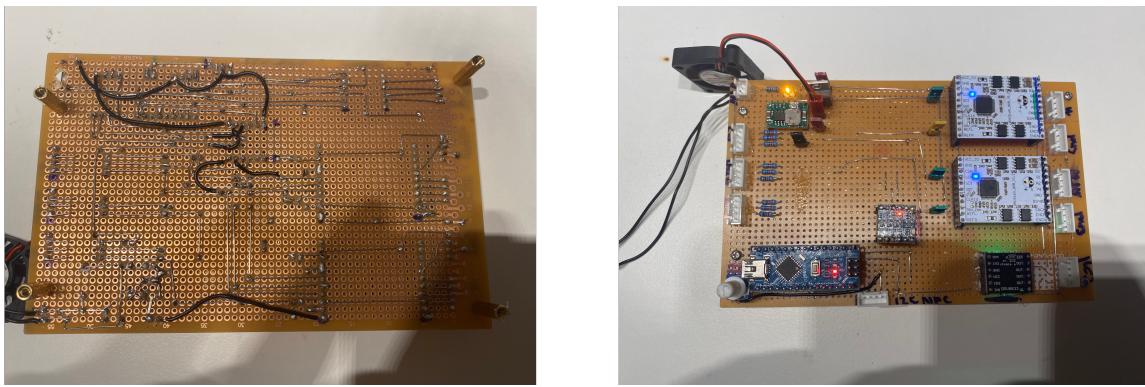


Bild 6-7: Prototyp Hardware: Aufbau der Lochrasterplatine

Durchgeführte Tests mit dem verwendeten [Atmega328p](#) haben ergeben, 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 machten sich durch eine gestörte Kommunikation mit dem [PN532](#) Modul bemerkbar und eine Auslesen von NFC Tags war nur in 60% der Fälle fehlerfrei möglich.

Im Anschluss wurde die Versorgungsspannung auf 5V erhöht, welches zur Folge hat, dass die Ein- und Ausgänge ebenfalls mit diesem Pegel arbeiten; dieser Schritt 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 6-7 im Eurokartenformat manuell aufgebaut und getestet.

Aus diesem Design wurde ein Printed Circuit Board (PCB) Layout für eine einfache 2 lagige Platine erstellt. Dieses orientiert sich an der zuvor umgesetzten Lochrasterplatine und spiegelt das Layout wider. Auch wurde hier nicht auf den Platzverbrauch geachtet. Es wurde 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.

6.4 Implementierung HAL

Die Hardware Abstraction Layer (HAL) stellt das Verbindungsglied zwischen der Hardware und der Benutzer-Software dar. In diesem Fall übernimmt diese 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. Weiterhin können Daten über das File-Handle gelesen- und geschrieben werden. Somit ist eine Kommunikation mit der Hardware-Ebene möglich.

Diese Funktionalität wird von der für diese Projekte implementierten HAL in Form einer C++ Klasse abgebildet und ermöglicht einen einfachen Zugriff auf die elektrisch verbundenen Komponenten. Zusätzlich wird in dieser 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 die 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 diese 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 aktuellen Revision aus, so ist keine manuelle Pflege der Tabelle während einer möglichen Produktion nötig.

```
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();
```

6 Erstellung erster Prototyp

```
24 void move_to_postion_mm_absolute(const int _x, const int _y,
25     const bool _blocking);
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 //SPICommunication.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));
27
28        //READ RESULT BACK
29        if (res == -1)
30        {
31            if (c > MAX_SPI_RETRIES)
32            {
33                //Handle error
34            }
35            else
36            {
37                c++;
38            }
39        }
40    }
41}
```

```

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_BYTE)
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 //...

```

Die HAL und deren benötigten Softwarekomponenten zur Buskommunikation und Hardware-Revisionsbestimmung wurde 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 ausgehen.

6.4.1 TMC5160 SPI Treiber

Der Treiber für die verwendeten TMC5160 Schrittmotor-Treiber ist ebenfalls ein Bestandteil der 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 6.1 setzen zu können in Bezug auf Beschleunigung und Positioniergeschwindigkeit des Motors.

Tabelle 6.1: TMC5160 Beschleunigungskurve / RAMP Parameter

Parameter	Value
V_START	1
A1	25000
V1	250000

Parameter	Value
A_MAX	5000
V_MAX	1000000
D_MAX	5000
D1	50000
V_STOP	10

Der Treiber wird nur im Position-Mode betrieben, welcher eine wesentliche Eigenschaft dessen ist. 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 anderen Aufgaben können währenddessen ausgeführt werden. Die Beschleunigungskurve kann zusätzlich manuell angepasst werden. Hier wurden jedoch die Standardwerte aus dem Datenblatt verwendet, welches sich bei mehreren Tests als optimal in Bezug der Geräuschemission des Motors herausstellten.

```

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 OFF TO 10 SEC

```

6 Erstellung erster Prototyp

```
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);
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
52 int TMC5160::read(const int _address)
53 {
54     //POPULATE WRITEBUFFER = READ REGISTER ADDRESS
55     const size_t DATA_LEN = 5;
56     uint8_t write_buffer[] = { _address & 0x7F, 0, 0, 0, 0 };
57     uint8_t read_buffer[] = { _address & 0x7F, 0, 0, 0, 0 };
58     //FIRST WRITE REGISTER ADRESS TO READ
59     int res = SPICommunication::getInstance()->spi_write(
60         SPI_CS_DEVICE ,write_buffer , DATA_LEN);
61     //READ RESULT
62     res = SPICommunication::getInstance()->spi_write(
63         SPI_CS_DEVICE , read_buffer , DATA_LEN);
64     //PARSE RESULT INTO INT
65     int value = read_buffer[1];
66     value = value << 8;
67     value |= read_buffer[2];
68     value = value << 8;
69     value |= read_buffer[3];
70     value = value << 8;
71     value |= read_buffer[4];
72     return value;
73 }
74
75 //EXAMPLE USAGE, GOTO POSITION
76 void TMC5160::go_to(const int _position) {
```

```

74     write(REGDEF_RAMPmode, 0);
75     //SET XTARGET REGISTER = TARGET POSITION
76     //NON BLOCKING
77     write(REGDEF_XTARGET, _position);
78     //USE move_to_position_mm_relative FOR A BLOCKING VARIANT
79 }
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_1) {
91         std::this_thread::sleep_for(std::chrono::microseconds
92             (1));
93     }
94     //STOP MOTOR
95     hold_mode();
96     //SAVE LATCHED POSITION
97     int offset = get_position() - get_latched_position();
98     write(REGDEF_XACTUAL, offset);
99     //SAVE OFFSET
100    int currpos = get_position();
101    set_postion_offset(currpos);
102    //RESET RAMP
103    write(REGDEF_RAMPSTAT, 4);
104    //GOTO THE NEW ZERO POSTION
105    set_AMAX(RAMP_AMAX);
106    set_VMAX(RAMP_VMAX);
107    go_to(0);
108    //DISABLE HARD ENDSTOP
109    enable_switch(TMC5160::REF_SWITCH::REF_L, true, false,
110        true);
111    disable_motor(); //DISBLE MOTOR
112 }
113 //...

```

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 zuerst an die Home-Position gefahren werden. Dazu besitzt das System zwei Endschalter, welches jeweils mit einem Schrittmotor-Treiber verbunden sind. Diese besitzen zwei solcher 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 Registereintrag 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 0 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.

6.5 Fazit bezüglich des ersten Prototypens

In Hinsicht auf den Umsetzungsprozess des autonomen Schachttischs stellt die Fertigstellung des ersten Prototypens einen ersten großen Erfolg dar Dennoch konnten nicht alle zuvor gestellten Requirements mit diesem Design umgesetzt werden.

Dazu zählt zum einen der Bewegungsspielraum der einzelnen Achsen. Dieser wurde bereits wären der Entwicklung durch die Verwendung von zwei Elektromagneten künstlich verlängert. Nach einem Langzeittest stelle sich jedoch diese Methode als zu Fehleranfällig dar. Die Parkpositionen, welche sich an den zwei Seiten des Spielbrettes befinden, konnten nicht durchgehen zuverlässig angefahren werden und boten nur Platz für 14 ausgeschiedene Figuren pro Spielerfarbe. Somit ist ein komplettes Abräumen des Spielfeldes nicht möglich, was jedoch in der Praxis selten vorkommt.

Zum anderen ist der Aufbau und die anschließende Kalibrierung der Mechanik und der entsprechenden Offset-Werte in der Software nicht trivial und benötigen 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 den verschraubten Holzplatten machen jedoch eine Vervielfältigung mit gleicher Qualität schwierig. Ein Re-Design der inneren Komponenten gestaltet sich schwierig, da hier bereits mehrere Iterationen durchgeführt wurden, um eine maximalen möglichen Verfahrweg zu ermöglichen.

Auf Seiten der Elektronik arbeitet diese ehr zuverlässig und bereitete keinerlei Probleme. Jedoch stellen die verwendeten Motortreiber einen größeren Kostenfaktor dar und

der Zeitaufwand für den Zusammenbau und Überprüfen dieser dar. Die verwendeten Elektromagnete sind für 9V Betriebsspannung ausgelegt, mussten jedoch über ihren Spezifikationen mit 12V betrieben werden, welche bei einem Dauerbetrieb zu stark erhöhten Temperaturen führte.

Allgemein war hier die Entscheidung die Außenmaße des Tisches zu optimieren nicht ideal und führt zu diversen Problemen. Diese konnten jedoch mit verschiedenen Workarounds behoben werden können. Ein Spiel ist mit diesen Prototypen mit Einschränkungen möglich und bildet bis auf das Fehlen der nicht funktionstüchtigen Parkpositionen die zuvor festgelegten Requirements ab. Im weiteren Verlauf der Entwicklung steht jedoch die Verbesserung der Zuverlässigkeit und die fehlerfreie Umsetzung der Parkposition für ausgeschiedene Figuren. Ein einfacherer Zusammenbau auch für dritte sollte ebenfalls ins Auge gefasst werden. Hierzu wird ein komplettes Re-Design der Mechanik sowie der Elektronik nötig sein. Anpassungen der Software ist dadurch ebenfalls nötig, stellt jedoch durch den modularen Aufbau dieser kein Hindernis dar. Die durch diese Prototypen gewonnenen Erkenntnisse können somit direkt in das neue Design einfließen.

7 Aufbau des zweiten Prototypen

7.1 Modifikation der Mechanik

7.1.1 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 muss 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 wurde als Basis des neuen System Aluminium-Profilstangen gewählt. Da der Tisch keine größeren Kräfte aufnehmen muss, wurde ein Stangengrundmaß von 20 x 20 mm gewählt. Diese 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 einander 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 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 eine Schachfeld-



Bild 7-1: Production Hardware: Finaler autonomer Schachtisch

größe 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 mögliche Einschränkungen der Mechanik vorzubeugen.

Die Platte wird dann in Alu-Profilstangen eingelassen; die Stangen sollen die Platte umrahmen. Mit einem Grundmaß von 20 x 20 mm für die Profilstangen ergibt sich somit ein Gesamtmaß von 660 mm x 660 mm. Benötigt werden 8 Profilstangen der Länge 620 mm und 4 der Länge 170 mm, welche zu einem Quadratischen Kasten zusammengesetzt werden. Für die X-Achsen werden 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 Winkel gehalten, welche zum einen als Auflage für die Tischplatte fungieren und zum anderen das Befestigen der beiden X-Achsen Profilstangen dient.

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 Seiterelemente 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 verwendete Lack-Tisch und auch das selbsterstellte Untergestell als solche waren preisgünstig und leicht erhältlich, zudem war 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 dieser 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-Tischs wurden erneut verwendet; diese konnten für die zweite Revision verwendet werden und so zusätzlich die gleiche Montagehöhe zwischen der ersten und der zweiten Revision erreicht werden. Da selbst die Höhe der Quader des Schachtisches identisch ist, sind beide Tische nun gleich hoch. Eine Alternative Lösung wäre der Erwerb von simplen Hohlleisten der gleichen Länge oder aber das Integrieren weiterer Profilstangen, welche man optimaler Weise 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. Da hingegen benötigt der Tisch nun 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.

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

7.1.2 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 für Folgenden als Grundlage für alle neuen Komponenten genommen. 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 ergab bei den verwendeten Druckern 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.

7.1.3 Positions-Mechanik

Die Mechanik des ersten Prototypens wurde für die Erstellung des zweiten Prototypens 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 ermöglicht. Die Führung entlang der X-Achse erfolgt in der Mitte des Tischs, die Y-Achse wurde links und rechtsseitig rollbar gelagert und in der Mitte über einen Riemen gezogen. Dies hat zur Folge, dass bei entstehender Unwucht, welche durch die Bewegung des Schlittens auf der Y-Achse natürlich ist, die Y-Achse in ihren Lagerungen nicht mehr parallel verläuft, sondern beim Betätigen des Motors der X-Achse die Y-Achse in einem unerwünschten Winkel bewegt wird.

Die Konsequenz dessen ist, 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 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, sodass linksseitig und rechtsseitig unmittelbar an der Lagerung gezogen werden kann. Dies erwies sich jedoch als unpraktikabel und würde einen zusätzlichen Kostenfaktor darstellen. Ein stabileres Befestigen der Y-Achse in ihren Lagerungen hatte zur Folge, dass der Widerstand der Lagerungen zunahm und die Bewegung der Achse nur unter erhöhten Kräften möglich war.

Deswegen wurde ein völlig anderes System für die zweite Revision des Schachtisches gewählt, welche 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 des einen 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, es keine beweglichen Teile von nennenswerter Masse gibt. In der ersten Revision wurde beim Bewegen der X-Achse alle Komponenten bewegt, zu denen auch der höher gewichtige Schrittmotor zählt. 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.

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. Es ist lediglich in den Parametern CoreXY zu aktivieren und die Schrittweiten auszurechnen. Wird anschließend eine zu fahrende Strecke vorgegeben, berechnet Marlin selbstständig, wie schnell und in welche Richtung jeder der beiden Motoren bewegt

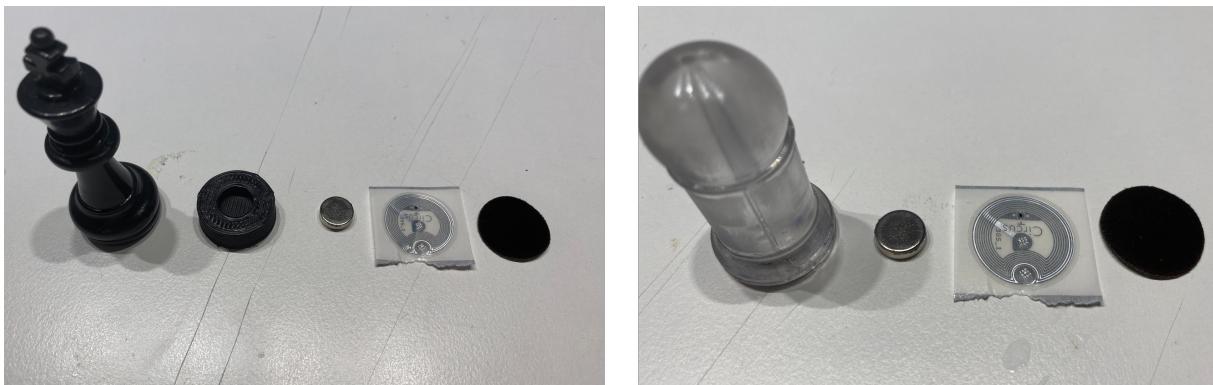


Bild 7-2: Schachfiguren im Vergleich

werden muss.

In der Komplexität des Aufbaus und dessen Zeitaufwand 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.

7.2 Optimierungen der Spielfiguren

Die bisher genutzten vorgefertigten Figuren funktionierten grundsätzlich gut mit dem ersten Prototyp. Allerdings wiesen 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, diese jeweils aus fünf 7-2 Einzelteilen bestehen:

- Figur
- Basisplatte
- Magnet
- NFC Tag

- Filzgleiter

Die Größe der Figuren kann durch die fest definierte Schachfeldgröße von 57mm und der 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 verschiedenen Bewegungsgeschwindigkeiten getestet, ergaben allerdings für diesen Anwendungsfall keine merkliche Verbesserung.

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, wurde einige Figuren zusätzlich mit einer 20mm Unterlegscheibe am Boden beschwert. Diese 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 das diese mit 5 Gramm zusätzlich genug Gewicht hinzufügen, um die magnetische Beeinflussung zu unterbinden.

Testweise wurden einige Figuren mittels 3D Drucker 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 verschiedenen 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 7-2 Einzelteile (Figur, Magnet, NFC-Tag, Filzgleiter) verwendet und zusätzliches verkleben dieser ist nicht mehr notwendig. Diese können direkt in den Fuß der Figur mittels einer Presspassung eingelegt werden.

7.3 Änderungen der Elektronik

Mit ein relevanter Kritikpunkt, welcher bereits während des Aufbaus des ersten Prototyps zu erkennen war, ist 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 dem Aufbau und Erprobung des ersten Prototyps wurde ersichtlich, dass hier nicht alle zuerst angedachten Features der Treiber benötigt werden und so auch Alternativen in Betracht gezogen werden konnten. Zusätzlich konnte die Elektronik nur beschränkt mit anderen Systemen verbunden werden, was insbesondere durch die verwendete SPI Schnittstelle geschuldet war.

All diese Faktoren erschweren einen einfachen Zusammenbau des autonomen Schachttischs. Die Lösung stellt die Verwendung von Standardhardware dar. Nach der Minimierung der elektrischen Komponenten und des mechanischen Aufbaus ist zu erkennen, dass der autonome Schachttisch einer CNC-Fr bzw. eines 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 Konsumerbereich sind auch kleine und preisgünstige Steuerungen 7.1 erhältlich, welche 2-3 Schrittmotoren und diverse zusätzliche Hardware ansteuern können.

Hierbei existiert eine große Auswahl dieser mit den verschiedensten Ausstattungen. Bei der Auswahl dieser wurde vor allem auf die Möglichkeit geachtet sogenannte Silent-Schrittmotortreiber verwenden zu können, um die Geräuschimmissionen durch die Motoren so weit wie möglich zu minimieren. Im ersten Prototyp wurde unter anderem aus diesem Grund die [TMC5160-B0B](#) Treiber ausgewählt. Die meisten Boards bieten austauschbare Treiber, so ist es auch im nachhinein möglich diese auszuwechseln.

Tabelle 7.1: 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 können. 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

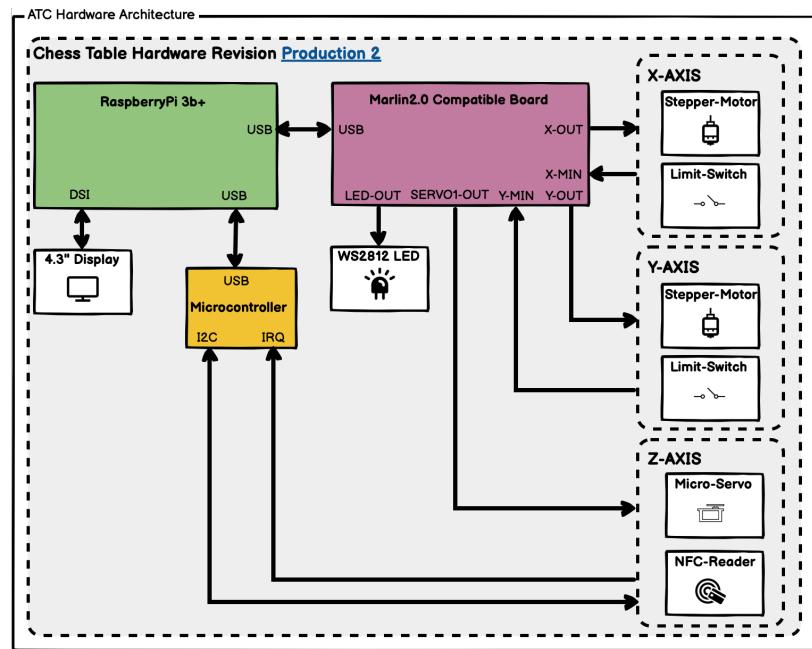


Bild 7-3: Production Hardware: Blockdiagramm

einfache Kommunikation gewährleistet. Das verwendete eingebettete System im autonomen Schachtisch 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 Marktpreises genug Ressourcen auch für spätere Erweiterung bietet und eine Unterstützung für die neuste Version der Marlin-FW[21] bereitstellt. Somit wurde die Elektronik durch die verwendete Plug&Play stark vereinfacht 7-3.

7.4 Anpassungen HAL

7.4.1 Implementierung GCODE-Sender

Durch die durchgeführten Änderungen an der Elektronik insbesondere durch die Verwendung einer Marlin-FW[21] fähigen Motorsteuerung, ist eine Anpassung der HAL notwendig. Diese unterstützt die Ansteuerung der Motoren und anderen 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 Befehle textbasierten Befehle 7.2, Komponenten dieser 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 von dem Hotend angefahren werden sollen.

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

Marlin-FW[21] bietet dabei einen großen Befehlssatz an G-Code Kommandos an. Bei diesem Projekt werden jedoch nur einige G-Code Kommandos verwendet 7.2, welche sich insbesondere auf die Ansteuerung der Motoren beschränken.

Tabelle 7.2: Grundlegende verwendete G-Code Kommandos

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

Die erforderlichen Kommandos wurden auf ein Minimum beschränkt, um eine maximale Kompatibilität bei verschiedenen G-Code-fähigen Steuerungen zu gewährleisten. Die Software unterstützt jedoch weitere Kommandos wie z.B. M150 mit welchem speziellen Ausgänge für LEDs gesteuert werden können. Dieses Feature bietet sowohl die verwendete Marlin-FW[21] als auch die verwendete Steuerung an. Sollte die verwendete Steuerung solch ein optionales Kommando nicht unterstützen, so werden diese ignoriert was zur Folge hat, dass auch preisgünstige Steuerungen verwendet werden können.

Die Kommunikation zwischen Steuerung und eingebetteten System geschieht durch eine USB Verbinden. 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 7.3 mit verschiedenen getesteten Steuerungen sowie deren USB-Vendor und Product-ID angelegt.

Tabelle 7.3: 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 bereitstellen. 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 CHARAKTER 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.

7.4.2 I2C-Seriell Umsetzer

Durch den Wegfall der zuvor eingesetzten Elektronik und der 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 die Möglichkeit, jedoch wurde auch hier auf eine USB Schnittstelle gewechselt. So ist es möglich das System auch an einem anderen Host-System zu betreiben, wie z.B. an einem handelsüblichen Computer.

7 Aufbau des zweiten Prototypen

Dazu wurde ein Schnittstellenwandler entwickelt welcher die I2C Schnittstelle zu einer USB Seriell wandelt. Indes wurde ein Atmega328p Mikrokontroller eingesetzt, da dieser weit verbreitet und preisgünstig zu beschaffen ist. Die Firmware des Mikrokontroller stellt ein einfaches kommandobasiertes Interface bereit. Die Kommunikation ist mit der Kommunikation und der Implementierung des G-Code Senders vergleichbar und teilen sich 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 {
4     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_SEPERATOR);
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        //DETERM 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 7.4, auf welchen die Firmware getestet wurde.

Tabelle 7.4: 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
HL-340	1a86	7523	User-Move-Detector
STM32F411	0483	5740	User-Move-Detector

7.5 Fazit bezüglich des finalen Prototypens

Der in der zweiten Iteration entstandene Prototyp wurde in viele Elemente aus der ersten Iteration grundlegend überarbeitet. Dabei entstand ein völlig neues Design, welches sich auf einfach zu beschaffende Komponenten und Materialien stützt. Diese ermöglichen einen simpleren und zeitlich effektiven Zusammenbau des vollständigen autonomen Schachtischs und bieten die Möglichkeit auf eine einfache Erweiterung des Systems.

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

Ein großer Kritikpunkt des ersten Prototypens war die nicht vollständig funktionsfähigen Park-Positionen für die ausgeschiedenen Figuren. Durch die Vergrößerung des Bewegungsspielraums der Achsen und der 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, welcher der Tisch mittels der LED Beleuchtung geben kann, ist der Nutzer nicht mehr auf die Graphical User Interface (GUI) angewiesen und erfährt visuell, ob der gegnerische Spielzug beendet wurde. Der Nutzer kann ohne Aufwand erkennen, in welchem Zustand sich der autonome Schachtisch befindet.

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

Somit ist festzuhalten, dass mit der zweiten Revision alle zuvor geforderten Eigenschaften 8.1 zufriedenstellend umgesetzt werden konnten. Die erstellte Hard- und Software

bietet zusätzliche zahlreiche Erweiterungsmöglichkeiten.

Tabelle 7.5: Eigenschaften der 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, welche 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 dem 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 andere Schachformen wie das Schnellschach, bei denen die Zugzeit begrenzt ist.

8 Entwicklung der Cloud Infrastruktur

Die erste Phase der Entwicklung des Systems bestand in der Auslegung und Erstellung der Cloud-Infrastruktur und der darauf ausgeführten Services. Die “Cloud” stellt in diesem Zusammenhang einen Server dar, welcher aus dem Internet über eine feste IPv4 und IPv6-Adresse verfügt und frei konfiguriert werden kann. Auf diesem System ist der Schach-Cloud Stack **8-1** installiert, welcher zum einen aus der Schach-Software besteht, welche in einem Docker-Stack ausgeführt wird und zum anderen....

8.1 API Design

Das System soll so ausgelegt werden, dass es zu einem späteren Zeitpunkt mit verschiedenen Client-Devices mit diesem kommunizieren können. Dazu zählen zum einen der autonome Schachtisch, aber z.B. auch einen 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 durch eine URI **8-2** eindeutig identifizierbar sind. Auf diese können mittels verschiedenster HTTP Anfragemethoden (GET, POST, PUT, DELETE) zugegriffen werden. Jeder dieser Methoden stellt einen anderen Zugriff auf die Ressource dar und beeinflusst somit das Verhalten und die Rück-Antwort dieser.

Eine URI besteht dabei aus mehreren Teilen. Das Schema gibt an wie die nachfolgenden Teile interpretiert werden sollen. Dabei wird 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 Anwendun-

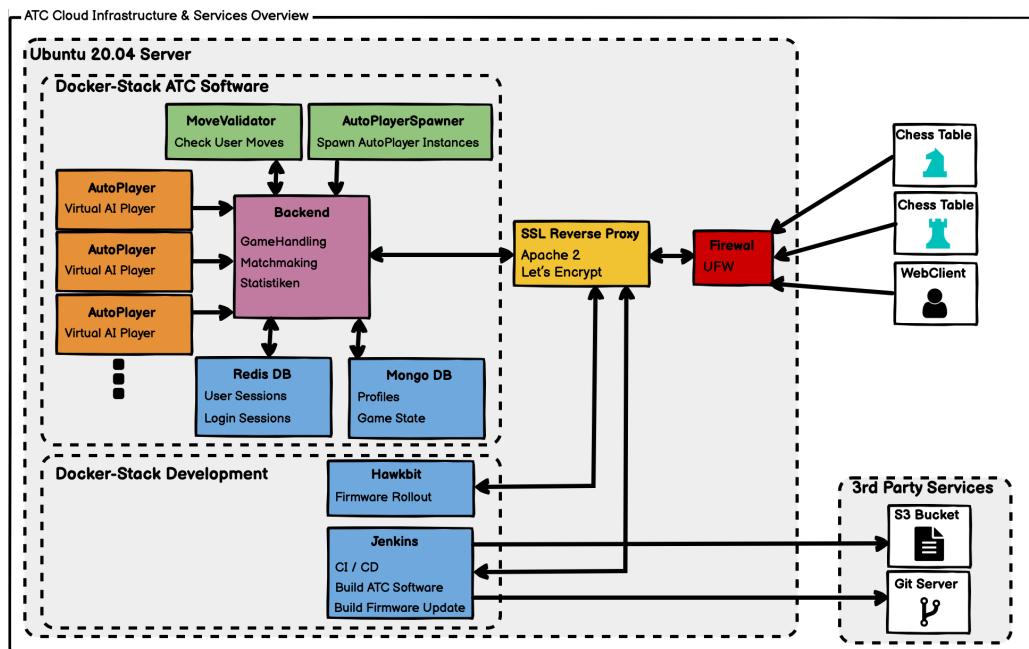


Bild 8-1: Gesamtübersicht der verwendeten Cloud-Infrastruktur



Bild 8-2: Cloud-Infrastruktur: Aufbau einer URI

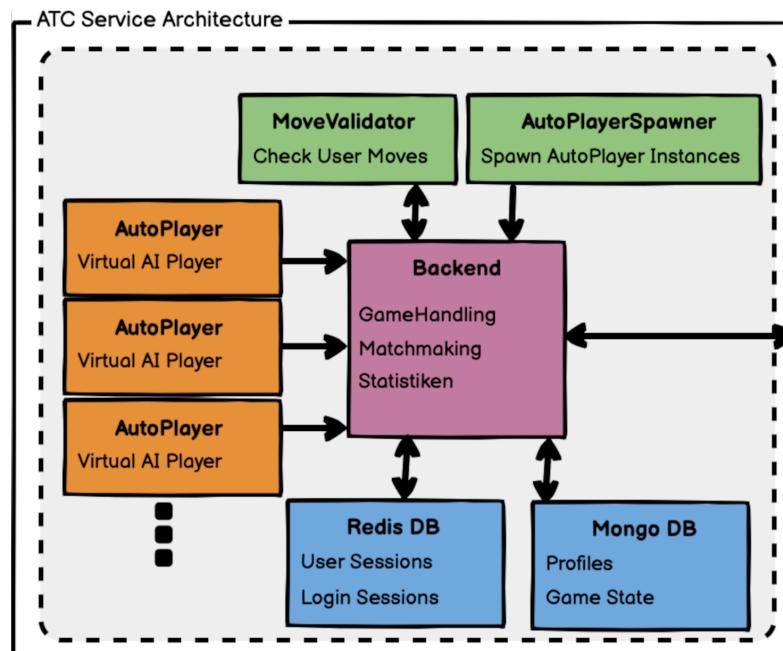


Bild 8-3: Cloud-Infrastruktur: Aufbau der Service Architecture

gen und Systemen bereit, welche durch ein Netzwerk miteinander verbunden sind. Dieser Ansatz ist somit geeignet um die verschiedenen Client Systeme (Schachtisch, Webclient) eine Kommunikation mit dem Server zu erlauben.

8.2 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 8-3. Durch dieses Design ist es zusätzlich möglich, die eigentliche Schach-Logik auszulagern zu können und somit ist es theoretisch möglich 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 jede dieser Services stateless und keine eigenen Daten speichert, werden zwei Datenbank-Services benötigt um die Spieldaten zu speichern:

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

Hierbei wurde auf zwei verschiedenen Datenbanken gesetzt. Die [Redis](#) 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, welche die notwendigen Informationen zu der aktiven Session des Clients gespeichert. Diese werden durch die ID des Clients abgefragt. Hierzu 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 abgeglegt werden. Zusätzlich zu den Spieldaten (Spiele, Spielstände, Statistiken), werden auch die Nutzerprofile speichert. Ein Profil wird beim ersten Anmeldevorgang erstellt und enthält außer den Profilinformationen (Geräte-ID, Namen, Spielertyp) auch die Referenzen auf die gewonnenen und verlorenen Spiele. Die können später für die Visualisierung verwendet werden.

Alle aufgelisteten Services werden in separaten Containern betrieben. Die Containervirtualisierung geschieht mittels der Software [Docker](#). Diese stellt ein einfaches Interface zur Erstellung von Containern und der Verwaltung dieser. Um einen Container auf dem System starten zu können, muss dieser zunächst aus einem Image heraus erstellt werden. Diese Image wird mittels einer [Dockerfile](#) beschrieben und besteht 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 zur Ausführung der ein-

gentlichen Software benötigt wird. Auch existieren bereits vorgefertigte Images, welche bereits Software für einen spezifischen Anwendungsfall enthält.

```
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 eine manuelles Management dieser als nicht praktikabel. Zu diesem Zweck existieren mehrere Tools und Systeme um solche Aufgaben zu automatisieren. Ein weitere Punkt sind Abhängigkeiten, welche unter den Container bestehen. In diesem Fall benötigt der Backend-Service die beiden Datenbanken um starten zu können. Somit ist es essentiell, dass diese bereits zuvor erstellt wurden und ausgeführt werden. Solche Funktionalitäten deckt das sehr leichtgewichtige Tool `docker-compose` 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      - "AtomicChessRedisDatabase:AtomicChessRedisDatabase"
12      - "AtomicChessMongoDatabase:AtomicChessMongoDatabase"
13      - "AtomicChessMoveValidator: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     #network_mode: "host"
29     restart: always
30     ports:
31       - 5000:5000
32     environment:
33       - PRODUCTION
34
35 AtomicChessRedisDatabase:
36   image: redis:latest
37   restart: always
38   container_name: atcredis
39   ports:
40     - 6379:6379
41   #network_mode: "host"
42
43 AtomicChessMongoDatabase:
44   image: mongo:latest
45   container_name: atcmongo
46   restart: always
47   environment:
48     - MONGO_DATA_DIR=/data/db
49     - MONGO_LOG_DIR=/dev/null
50   #volumes:
51   #   - ./data/db:/data/db
52   ports:
53     - 27017:27017
54   command: mongod --logpath=/dev/null # --quiet
55
56
57 AtomicChessAutoPlayer:
58   #depends_on:
59     - AtomicChessBackend
60   #container_name: atcautoplayer
61   build:
62     context: ../ATC_AutoPlayer/
63     image: atcautoplayer:latest
64     network_mode: "host"
65     restart: "always"
66     scale: 3 # SPAWN THREE INSTANCES
67     environment:
68       - PRODUCTION
69       - BACKEND_IP=127.0.0.1:3000 #HOST IP:PORT OF BACKEND
70         EXAMPLE 127.0.0.1:3000 USING ONLY HTTP
71       #- USE_HOSTNAME_HWID=TRUE # USE THE LOCAL MACHINE
72         HOSTNAME AS HWID
73       #- PLAYER_TYPE_HUMAN=1 # SIMULATE A HUMAN PLAYER TYPE
```

The screenshot shows a POST request to `http://atomicchess.de:3000/rest/login?hwid=f1ow389djiw&playertype=1`. The 'Params' tab is selected, showing two parameters: `hwid` with value `f1ow389djiw` and `playertype` with value `1`. The 'Body' tab shows the JSON response:

```

1  {
2      "err": null,
3      "status": "ok",
4      "sid": "daada700-a38a-11eb-9c30-7907ebbd50ac",
5      "profile": {
6          "profile_config": {
7              "SETTINGS": null,
8              "USER_DATA": null
9          },
10         "logs": [],
11         "hwid": "f1ow389djiw"
}

```

The status bar at the bottom indicates `Status: 200 OK Time: 130 ms Size: 865 B`.

Bild 8-4: Cloud-Infrastruktur: Backend Login-Request und Response

8.3 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. Diese 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 verwendet zum einen eine self-signed Certificate, sowohl als auch ein Zertifikat der Lets Encrypt Organisation[12]. Somit ist die vom Backend bereitgestellte API und zum späteren Zeitpunkt erstellen Webclient (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 Backen-Service stellt die grundleggegenden Funktionen bereit, welche die Clients benötigen. Dazu zählen unter anderem:

- Profilverwaltung
- Matchmaking
- Spielstatus
- Client authentifizierung

Jeder Client meldet sich mittels der `/rest/login` Route an. Das Backend prüft ob bereits ein Spielerprofil 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 8-4 erhält der Client einen Session-Token. Nur mit diesem Token, können weitere Funktionen des Backends verwendet werden. Dieser Token ändert sich nach jedem Login-Prozess, somit kann nur ein Client Token-Inhaber sein und andere zuvor angemeldete Clients wird dieser entzogen.

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 somit 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. Durch den Typ werden zuerst mit zwei Spielern ein Match gestartet, wenn diese vom einem der folgenden Typen ist:

- autonomer Schachtisch
- Webclient

```

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
15      if (CFG.getConfig().matchmaking_ai_enable === true &&
16        gpfm_res.player_searching_human.length === 1 &&
17        gpfm_res.player_searching_ai.length >= 1) {
18        //...
19        //START A MATCH FOR THESEES TWO PLAYERS => REMOVE
20        // LOBBY ENTRY FROM DB AND CREATE A NEW GAME IN
21        // THE GAME DATABASE
22        GH.start_match(gpfm_res.player_searching_human[0].
23          hwid, gpfm_res.player_searching_ai[0].hwid,
24          function (sm_err, sm_res) {
25            if (sm_err) {
26              //ON ANY ERROR THE CLIENT WILL RESET THE
27              // FAULTY STATE ITSELF AND RELOGIN TO THE
28              // SYSTEM
29              console.error(sm_err);
30            }
31          });
32        //MORE THAN 1 HUMAN PLAYER WAITING
33      }
34    }
35  }
36);

```

```

21      } else if (gpfm_res.player_searching_human.length > 1)
22      {
23          //THEN MAKE A MATCH BEWEEN THE TWO HUMAN PLAYER
24          //SORT PLAYER WITH THE LONGEST WAIT TIME IN THE
25          //LOBBY
26          gpfm_res.player_searching_human.sort(
27              player_sort_function_swt);
28          //SELECT THE MOST WAITING PLAYER
29          const p1 = gpfm_res.player_searching_human[0];
30          //SELECT A RANDOM OTHER PLAYER
31          const p2 = gpfm_res.player_searching_human[
32              HELPER_FUNCTIONS.randomInteger(1, gpfm_res.
33              player_searching_human.length - 1)];
34          //...
35          //START A MATCH
36          GH.start_match(p1.hwid, p2.hwid, function (sm_err,
37              sm_res) {
38                  if (sm_err) {
39                      //ON ANY ERROR THE CLIENT WILL RESET THE
34                      FAULTY STATE ITSELF AND RELOGIN TO THE
35                      SYSTEM
36                      console.error(sm_err);
37                  }
38              });
39      }, true);
40  );

```

Somit wird sichergestellt, dass zuerst alle menschlichen Spieler zusammen ein Spiel beginnen und erst im letzten Schritt, ein Mensch gegen dem Computer spielen muss. 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 enthält alle Spiele und deren aktuellen Status:

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

Diesen Eintrag 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 die-

sen mittels de 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 zurückversetzt, 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/hearbeat` Route zurückmelden. Somit weiß der Backen-Service, dass der Client noch existiert. Bleibt ein Request innerhalb einer bestimmten Zeit aus, werden alle akutellen 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.

8.4 Service: MoveValidator

Der MoveValidator-Service bildet im System die eigentliche Schachlogik ab. Die Aufgabe ist es, die vom Benutzer eingegebenen Züge auf Richtigkeit zu überprüfen und auf daraufhin 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.⁸⁻⁵ Der Response wird dann vom Backend in der Datenbank gespeichert und weiter an die Client-Devices verteilt.

Tabelle 8.1: MoveValidator-Service API Overview

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

Allgemein geschieht die Kommunikation über vier API Calls, welche vom MoveValidator-

The screenshot shows a Postman interface with the following details:

- Method:** POST
- URL:** http://move_validator:5000/rest/execute_move
- Body (form-data):**

KEY	VALUE	DESCRIPTION
<input checked="" type="checkbox"/> fen	rnbqkbnr/ppppppp/8/8/8/8/PPPPP/PPPP/RNBQK... [REDACTED]	
<input checked="" type="checkbox"/> move	e2e4	
- Body (Pretty):**

```

1 {
2   "err": null,
3   "move_executed": true,
4   "new_fen": "rnbqkbnr/ppppppp/8/8/4P3/8/PPPP1PPP/RNBQKBNR b KQkq - 0 1",
5   "next_player_turn": 1
6 }
```
- Status:** 200 OK
- Time:** 10 ms

Bild 8-5: MoveValidator: Beispiel Request zur Ausführung eines Zuges auf einem gegebenen Schachbrett

Service angeboten werden 8.1. 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 arbeitet wurde das komplette System so umgesetzt, dass dieses mit einem Spielfeld in einer Zeichenketten representation 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.

Die FEN Notation ist universal und kann jede Brettstellung darstellen. Auch enthält diese nicht nur die Figur Stellungen, sondern auch weitere Informationen, wie die aktuelle Nummer des Zuges oder welcher Spieler gerade an der Reihe ist. Diese werden dann in der Extended Forsyth-Edwards-Notation (X-FEN) Notation angegeben, bei der zusätzlich zu der Brettstellung auch noch die weiteren Informationen angehängt werden 8.2.

Tabelle 8.2: Vergleich FEN - X-FEN

FEN-TYPE	FEN-String
FEN	rnbqkbnr/pp1pppp/8/2p5/4P3/5N2/PPPP1PPP/RNBQKB1R
X-FEN	rnbqkbnr/pp1pppp/8/2p5/4P3/5N2/PPPP1PPP/RNBQKB1R b KQkq - 1 2

FEN-TYPE	FEN-String
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 Bibliothek [3]. Diese unterstützt zusätzlich die Generierung der für den Benutzer möglichen Schachzügen, 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 zusätzliche 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 Zügen, 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; darin sind unter anderem auch alle anderen Figur-Züge enthalten, solange der König des aktuellen Spielers sich aktuell auf dem Schachbrett befindet.

Wenn ein Spieler an der Reihe ist und einen Zug getätigter 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 getätigten Zug wieder her. Danach hat der Benutzer die Möglichkeit einen alternativen Zug auszuführen.

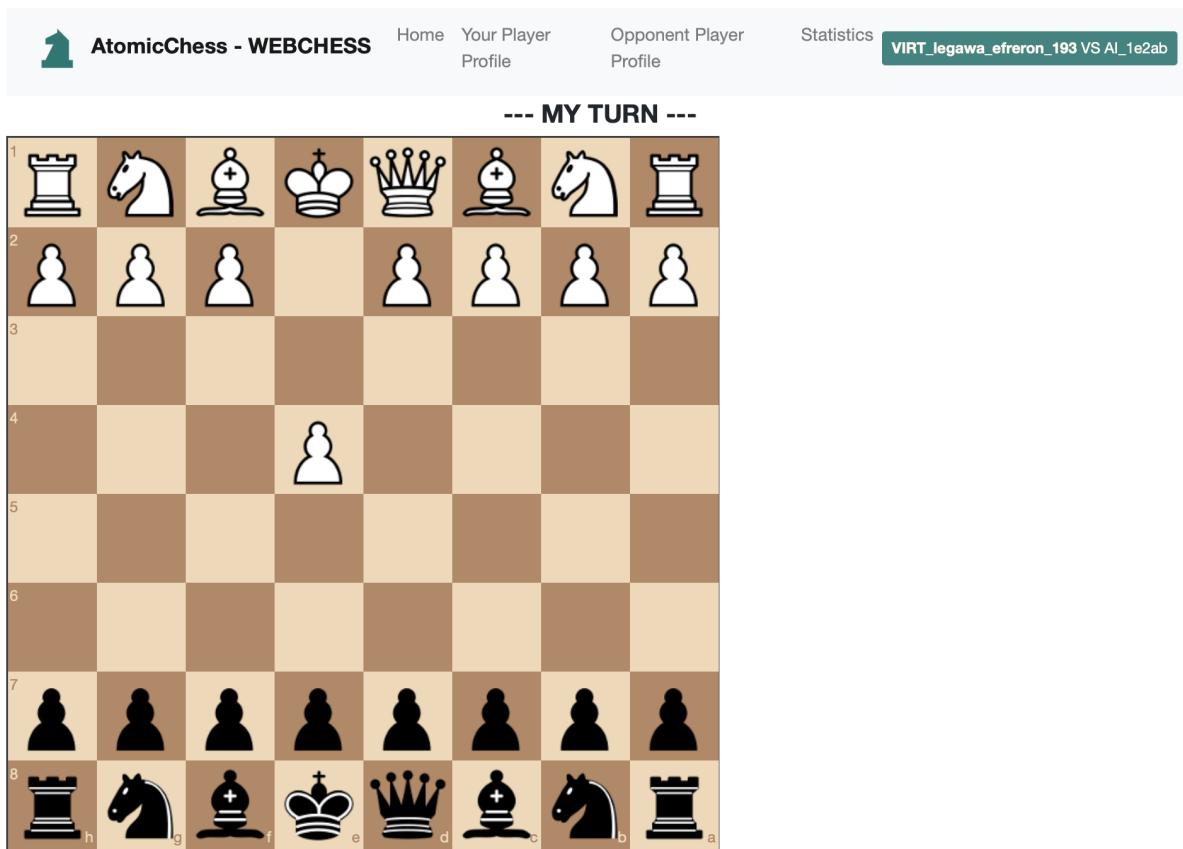


Bild 8-6: WebClient: Spielansicht

8.5 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) umgesetzt im Zusammenspiel mit Hypertext Markup Language (HTML) und Cascading Style Sheets (CSS) 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 Webclienten 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 als zusätzliche JS Bibliothek verwendet, welches eine Manipulation der HTML Elemente stark vereinfacht. Diese bietet insbesondere einfach zu nutzende HTTP-Request Funktionen bzw. Asynchronous JavaScript and XML (AJAX) an, welche für die Kommunikation mit dem Backen-Service verwendet werden. Diese werden im Hintergrund eingesetzt, sodass der WebClient auto-

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

Der Hauptanwendungsfall des Webclienten **8-6** während der Entwicklung ist es, weitere Spieler zu simulieren und so ein Spiel mit nur einem autonomen Schachtisch testen zu können. Durch den Webclient ist zusätzliche 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 getätigter werden.

Während der Implementierung wurde der Webclient weiter ausgebaut und es wurde 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 Information über den Spielstatus angezeigt werden.**8-7** Auch ist es möglich, aktuell laufende Spiele in Echtzeit anzeigen zu lassen; somit wurde eine Livestream-Funktionalität implementiert.

8.6 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 anderen 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.

The screenshot shows a chess game statistics page. At the top, there's a navigation bar with a chess knight icon, the text "AtomicChess - STATISTICS", and a "Home" link. Below it is a section titled "LAST GAMES - HISTORY" with a table:

#	CREATED	PLAYER WHITE	PLAYER BLACK	MOVE COUNT	GAME_STATE	VIEW
1	2021-04-20_19:18:40	AI_1e2ab	VIRT_legawa_efreron_193	39	RUNNING	SHOW DETAILS

Below this is a "GAME INFORMATION" section for game ID 25a34660-a1fc-11eb-9cfc-ff3c5889598c. It includes a chessboard diagram, player information, and board FEN details:

- PLAYER_TURN:** WHITE
- IS_GAME_OVER:** false
- START BOARD FEN:** rnbqkbnr/pppppppp/8/8/8/PPPPPPP/RNBQKBNR
- LATEST BOARD FEN:** 1kr3nr/7p/p7/1p2Q3/2P5/8/PP3PPP/R1B2K1R w - - 0
20

At the bottom, there's a "CURRENT BOARD" section with a legend:

8	King	Queen	King	Queen	Knight	Bishop	Rook
---	------	-------	------	-------	--------	--------	------

Bild 8-7: Webclient: Statistiken

Eine weitere Modifikation ist die Verwendung einer Schach-AI, da dieser Service als Computerspieler agieren soll. Hierzu kam die Open-Source Chess Engine Stockfish[18] in der Version 11 zum Einsatz. Die Stockfish-Engine bietet noch weitere Features, als nur die nächstbesten Züge zu einem gegebenen Schachbrett zu ermitteln.

Die AutoPlayer-Instanz kommuniziert über das Universal Chess Interface (UCI) Protokoll[17] mit der Engine. Dieses Protokoll wird in der Regel von Schach-Engines verwendet, um mit einer 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 sendet daraufhin eine List möglicher Züge zurück. Der erste Index dieser Liste ist dabei der am besten bewerteten Zug der Engine.

Im Kontext des AutoPlayer-Service wird der Engine nur das aktuelle Spielbrett übermittelt und der nächstbeste 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 fallen. 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. Diese Feature wurde insbesondere bei der Entwicklung des Webclient und der Steuerungssoftware für den autonomen Schachtisch verwendet.

9 Embedded System Software

- Hauptsoftware zur Steuerung der Elektrik/Mechanik
- Kommunikation mit dem Cloud-Server

9.0.1 Anmerkungen Compiler

Die Controller-Software wurde in C++ erstellt und verwendet Features des C++ 17 Standart:

- constexpr lambda
- lambda capture

Diese Features werden im Interprocess Communication (IPC) Modul (s.u.) , sowie einigen verwendeten Bibliotheken verwendet. Auf dem Host-Entwicklungssystem, sowie dem eingebetteten System wurde der GCC-Compiler mit der Version 10.2 installiert und wird für das Erstellen der einzelnen Software-Componenten verwendet.

9.1 Ablaufdiagramm

Nach dem Start der Controller-Software folgt diese einem Fest vorgegebenen Ablauf 9-1. 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 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.

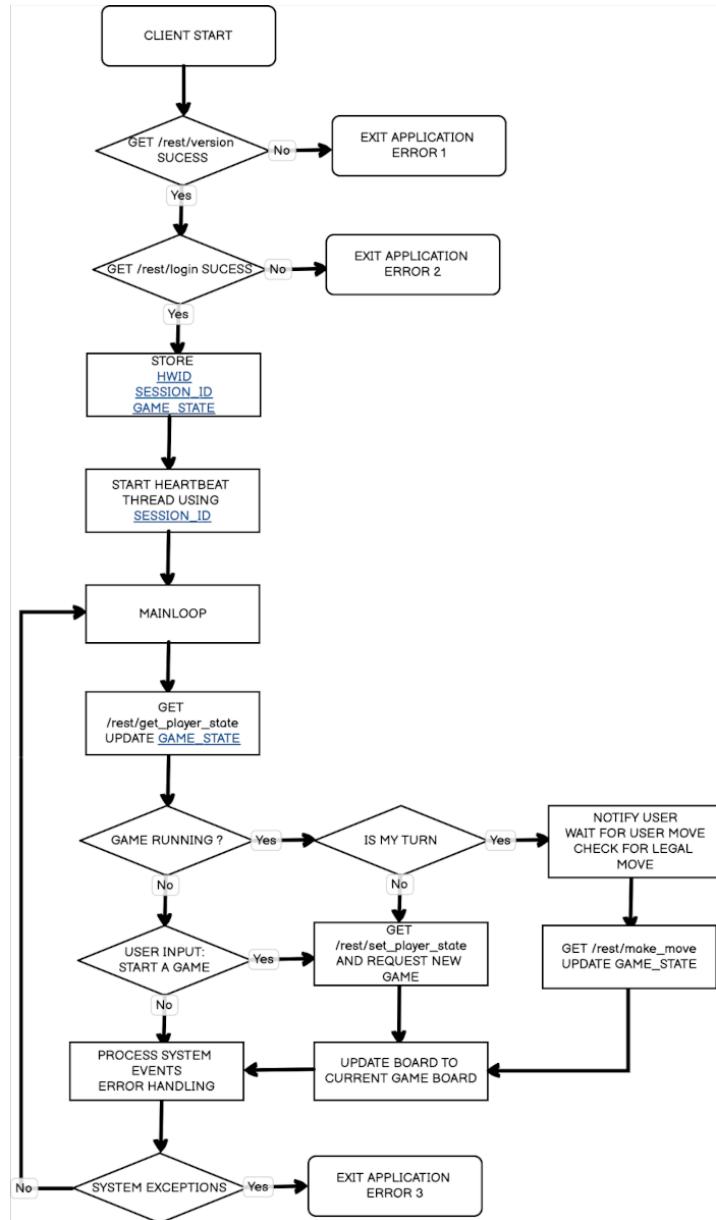


Bild 9-1: Embedded System Software: Ablaufdiagramm

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 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. Welches entweder durch manuelles Eintippen des Schachzugs über die GUI geschieht oder über eine manuelle Bewegung der Figuren. 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 informieren, diesen Rückgängig zu machen. Ist der Zug jedoch gültig, wird dieser vom Server an den anderen Client übermittelt und dieser muss anschließend wie beider Sync-Phase das Spielbrett aufbauen.

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 die 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.

9.2 Figur Bewegungspfadberechnung

Nach dem Start der Software wird 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 befinden, ansonsten in ein Start des Programms nicht möglich.

Während der Sync-Phase muss die Software das vorgegebene Schachfeld herstellen. Dazu hält die Software den aktuellen Brett-Zustand vor und vergleicht diese mit dem Ziel-Schachbrett. Durch einen Vergleich dieser können die sich geänderten Figuren

lokalisiert werden. Dadurch dass immer Ziel und Aktuelles-Spielbrett miteinander verglichen werden, können mehrere Züge auf einmal durchgeführt werden. Hierbei ist es auch möglich auf Spielbrett in einem Zustand X, wieder die Ausgangsposition herstellen zu können. Somit kann ein beliebiges Spielfeld vorgegeben werden, welches der Tisch dementsprechend aufbaut.

Um dies zu ermöglichen, wird aus der Vergleich der beiden Spielbretter eine Differenz in Form einer Liste gebildet. In dieser sind alle Änderungen eines einzelnen Feldes 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
12    for (int i = ChessField::field2Index(ChessField::
13        CHESS_FILEDS::CHESS_FIELD_A1);
14            i < ChessField::field2Index(ChessField::CHESS_FILEDS
15                ::CHESS_FIELD_PARK_POSITION_WHITE_1); i++) {
16        ChessPiece::FIGURE tmp_curr = getFieldOnField(
17            board_current, ChessField::Index2Field(i));
18        ChessPiece::FIGURE tmp_target = getFieldOnField(
19            board_target, ChessField::Index2Field(i));
20        //CHECK IF EQUAL FIGURES
21        if (ChessPiece::compareFigures(tmp_curr, tmp_target))
22        {
23            continue;
24        }
25        ChessBoard::FigureFieldPair tmp;
26        tmp.field_curr = ChessBoard::FigureField(ChessField::
27            Index2Field(i), tmp_curr);
28        tmp.field_target = ChessBoard::FigureField(ChessField
29            ::Index2Field(i), tmp_target);
30        tmp.processed = false;
31        diff_list.push_back(tmp);
32    }
33    return diff_list;
34 }
```

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 kann 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 keine Änderungen auf dem Spielbrett mehr gibt. Der rekursive Ansatz ist notwendig, da nicht alle Figuren ihren Bestimmungsort einnehmen können, wenn diese noch von einer anderen Figur belegt sind. Diese muss zuerst auf deren Ziel-Feld geschoben werden.

Aus den Start und Ziel-Feldern werden im letzten Schritt-Wegpunkte **9-2** generiert. Diese beschreiben den Weg, welchen die Figur von Start zum Zielfeld ablaufen muss. Das Spielbrett wurde so designt, dass zwischen jeder Figur auf dem Feld immer 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 und an den Rand dieses Bewegt wurde, kann die Figur ungehindert zwischen den anderen 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 diesem befindet.

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

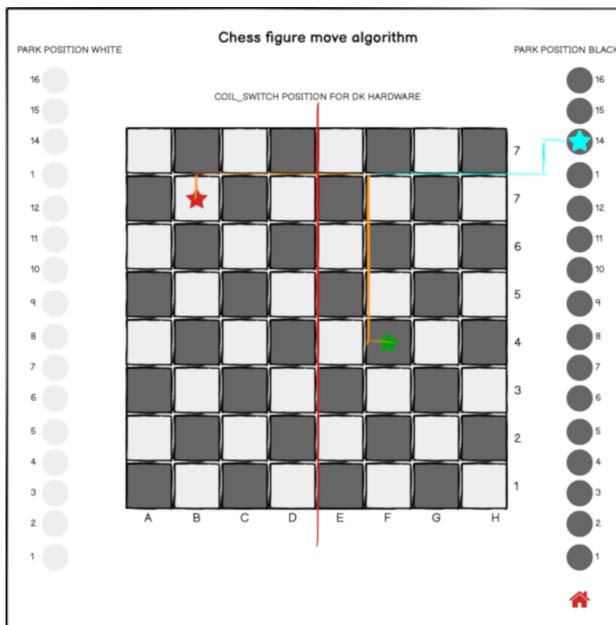


Bild 9-2: Embedded System Software: Figur Wegpunkte

9.3 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, manuell eingegeben werden. 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 Schachfelder verfügt, wurde der existierenden 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 der Bewegung der Mechanik ca 3 Sekunden benötigt. Zusätzlich verlängert sich die Scandauer durch ein leeres Schachfeld, da der Scanner mehrere Versuche unternimmt ein gültiges NFC Tag zu erkennen.

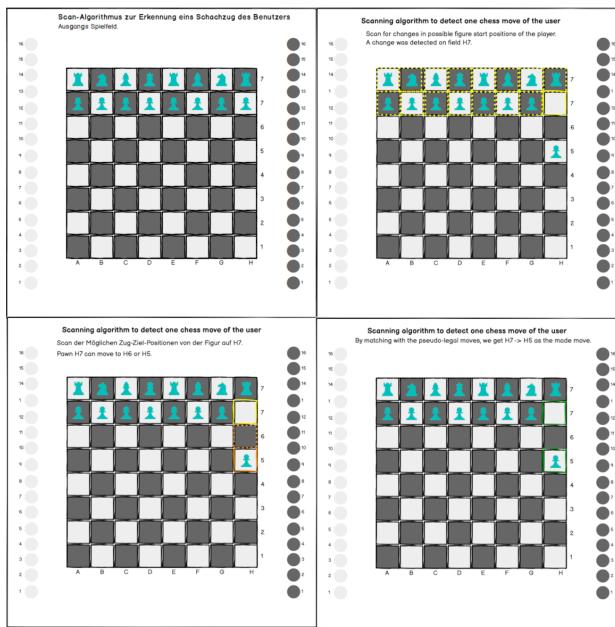


Bild 9-3: Embedded System Software: Schachfeld Scan Algorithmus Ablauf

Somit muss mittels eines Algorithmus 9-3 entschieden werden, welches Felder als mögliche Kandidaten in Fragen kommen. Hinweise auf diese Felder bietet der aktuelle Spiel-Status, welche 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 mögliche 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 Feld ziehen kann. 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.

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 abzuscanen. Darauf hin 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.

9.4 Inter Prozess Communication

Bei der Entwicklung des Systems wurde darauf geachtet, dass das User-Interface auswechselbar bleibt. Somit ist es auch möglich, ein webbasiertes User-Interface zu integrieren. Dazu wurde ein zusätzliches IPC Layer hinzugefügt, welches 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 welcher die Kommunikation mit den anderen IPC Instanzen verwaltet wird.

Der Haupthread 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

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 Basierende Implementierung ist es möglich die andern IPC Instanzen auszulagern und auf verschiedenen Endgeräten ausführen zu können.

```
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 Webbrower möglich, welches 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 guicomunicator.cpp
2 //SIMPLIFIED EXAMPLE USAGE
3
4 //INIT IPC SERVER
5 guicomunicator gui;
6 gui.start_recieve_thread();
7 //CHECK OTHER PROCESS REACHABLE
8 while (!gui.check_guicomunicator_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);
23 //FLIP SCREEN ORIENTATION
24 gui.createEvent(guicomunicator::GUI_ELEMENT::
25                  QT_UI_SET_ORIENTATION_180, guicomunicator::GUI_VALUE_TYPE
26                  ::ENABLED);
27
28 //GET EVENT FROM OTHER PROCESSES STORED IN EVENT QUEUE
29 guicomunicator::GUI_EVENT ev = gui.get_gui_update_event();
30 if (!ev.is_event_valid){
31     gui.debug_event(ev, true);
32     continue;
33 }
34 //CHECK EVENT QUEUE FOR USER INPUT
35 if(ev.event == guicomunicator::GUI_ELEMENT::BEGIN_BTN_SCAN &&
36     ev.type == guicomunicator::GUI_VALUE_TYPE::CLICKED) {}
```

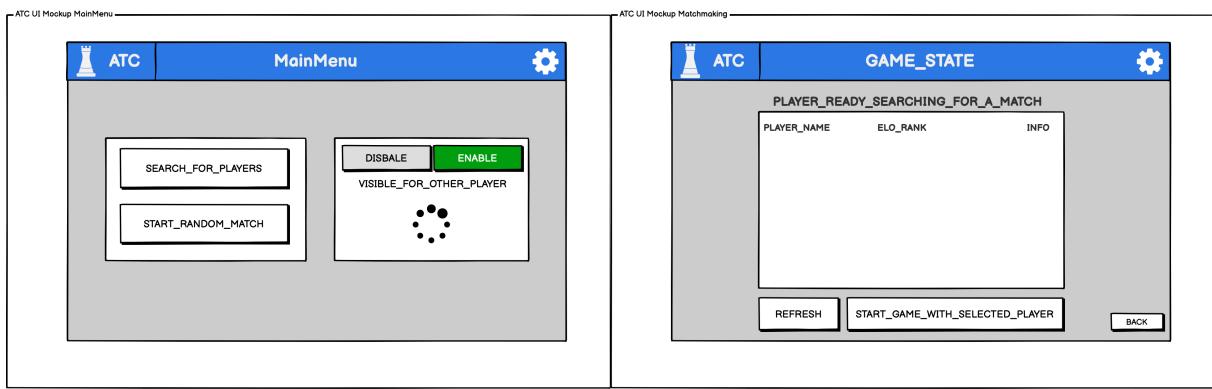


Bild 9-4: Embedded System Software: User-Interface Mockup

9.5 Userinterface

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 Schachspiels nötig sind. Durch die kleinen Abmessungen des Displays mit 4.3 Zoll, wurde alle Bedienelemente in ihrer Größe angepasst, sodass der Benutzer auch von einer weiter entfernten Position den Zustand direkt erkennen kann. Auch wurden 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 der meist nur also Informationsquelle über den Spielstand dienenden User-Interface, bietet diese viele Möglichkeiten der Konfiguration des Systems. Somit kann auf ein weiteres Eingabegerät, wie z.B. einem Mobiltelefon verzichtet werden, da alle relevanten Einstellungen im Optionen-Menu vorgenommen werden können.

Als Framework wurde hier das Qt[4] verwendet, da dieses bereits im Buildroot-Framework in der Version 5.12 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 festgehalten und konnten anhand dieser schnell umgesetzt werden.

Das Qt[4] bietet dazu einen separaten Editor [Qt Design Studio](#) an, in denen die zuvor

erstellen Wireframe-Grafiken importiert wurden und anschliessen mit den Bedienelementen ersetzt werden könnten. 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    // 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-Interface bestand in der Verbindung, der in Qt Modeling Language (QML) erstellten Bedienelemente durch den [Qt Design Studio](#)-Editor und der User-Interface Backend Logik. Diese beschränkt sich auf die Initialisierung des Fensters und dem anschließenden laden und darstellen

des QML Codes. Die Backend-Logik Funktionalitäten in einem QML Typ `MenuManager` angelegt, welcher vor dem Laden des eigentlichen User-Interface QML-Code 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        );
16    //LOAD User-Interface QML
17    QQuickView view;
18    //...
19    view.engine()->addImportPath("qrc:/qml/imports");
20    view.setSource(QUrl("qrc:/qml/WINDOW.qml"));
21    view.engine()->rootContext()->setContextProperty("app", &app
22        );
23    //...
24    //IMPORTANT STEP: AFTER INIT THE MainMenu COMPONENT HAS NO
25    //PARENT
26    //SO WE NEED TO SET IT MANUALLY TO MAKE C++ -> QML FUNCATION
27    //CALLS WORKING
28    QObject *object = view.rootObject();
29    QObject *rect = object->findChild<QObject*>("mainmenu");
30    if (rect){
31        rect->setParent(object);
32    }
33    //FINALLY SHOW MENU ON SCREEN
34    view.show();
35 }
```

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 in der Makefile das `USES_QT` Define-Flag gesetzt. Wenn dieses 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 z.B. durch eine Benutzereingabe oder wenn das User-Interface die 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
15         ::START_CALBOARD_PROC);
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 guicommunicator::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 == guicommunicator::GUI_ELEMENT::SWITCH_MENU){
29         switch_menu(ev.type);
30     }
31     //...
32 }
```

10 Fazit

Zusammenfassend lässt sich feststellen, dass das Ziel der Arbeit erreicht wurde. Es wurde ein Prototyp eines autonomen Schachtischs entwickelt.

- mit am weitesten forgeschrittenen open-source autonomes Schachtisch Projekt
- vom versierten Benutzer selbständig aufbaubar
- leichte bedienung
- lässt spiel für erweiterungen
-

10.1 Persönliches Fazit

10.2 Ausblick

- Einbindung in existierende Schach-Clouds z.B. <https://lichess.org/>
- user-port für Erweiterungen (z.B. DGT Schachur)

Literaturverzeichnis

- [1] DGT: *DGT Bluetooth Wenge*. <https://www.topschach.de/bluetooth-wenge-eboard-figuren-p-3842.html>. Version: 28.03.2021
- [2] DGT: *DTG Smart Board*. <https://www.topschach.de/smart-board-p-3835.html>. Version: 28.03.2021
- [3] FIEKAS, Niklas: *Python-Chess Library*. <https://github.com/niklasf/python-chess>. Version: 28.03.2021
- [4] GROUP, Qt: *Qt*. <https://www.qt.io>. Version: 28.03.2021
- [5] GUERERO, Michael: *Automated Chess Board*. <https://create.arduino.cc/projecthub/maguerero/automated-chess-board-50ca0f>. Version: 28.03.2021
- [6] INC., Infvention: *Square Off - Chess App*. <https://apps.apple.com/app/square-off-chess/id1267805783>. Version: 28.03.2021
- [7] INC., SquareOff: *Grand Kingdom Set*. <https://squareoffnow.com/product/gks>. Version: 28.03.2021
- [8] INC., SquareOff: *Kingdom Set*. <https://squareoffnow.com/product/kds>. Version: 28.03.2021
- [9] MACHINES, DIY: *DIY Super Smart Chessboard*. <https://www.instructables.com/DIY-Super-Smart-Chessboard-Play-Online-or-Against-/>. Version: 28.03.2021
- [10] 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

[11] McEVOY, Brian: *Ultimaker Cura Slicer*. <https://github.com/Ultimaker/Cura>.

Version: 28.03.2021

[12] NON-PROFIT-ORGANISATION, Let's E.: *Let's Encrypt*. <https://letsencrypt.org/de/>. Version: 28.03.2021

[13] NXP: *NXP NTAG 21.* <https://www.nxp.com/products/rfid-nfc/nfc-hf/ntag-for-tags-labels/ntag-213-215-216-nfc-forum-type-2-tag-compliant-ic-with-144-504-888-bytes-user-data> Version: 28.03.2021

[14] RAVICHANDRAN, Akash: *Automated Chess Board*. <https://create.arduino.cc/projecthub/automaters/automated-chess-5dbd7a>. Version: 28.03.2021

[15] 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. Version: 28.03.2021

[16] 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. Version: 28.03.2021

[17] 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

[18] TORD ROMSTAD, Joona K. Marco Costalba C. Marco Costalba: *Stockfish - Strong Open Source Chess Engine*. <https://stockfishchess.org/>. Version: 28.03.2021

[19] 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/normenausschuesse/naerg-veroeffentlichungen/wdc-beuth:din21:289443385>. Version: 04.04.2021

- [20] YOUSIFNIMAT: *Chess Robot.* [https://www.instructables.com/Chess-Robot/.](https://www.instructables.com/Chess-Robot/)
Version: 28.03.2021
- [21] ZALM, Erik van d.: *Marlin Firmware.* <https://marlinfw.org/>, <https://github.com/MarlinFirmware/Marlin>. Version: 28.03.2021

Akronyme

AI Artificial Intelligence. 59, 66, 67

AJAX Asynchronous JavaScript and XML. 65

API Application Programming Interface. 52, 58, 60, 62, 64, 67, 71

ATC Atomic Chess Table. 14, 51

CAD Computer-Aided Design. 17, 19, 42, 45

CNC Computerized Numerical Control. 39, 43, 44

CSS Cascading Style Sheets. 65

FDM Fused Deposition Modeling. 17, 18

FEN Forsyth-Edwards-Notation. 63, 64

FIFO First In – First Out. 76

GPIO General Purpose Input/Output. 24–26, 48

GPL General Public License. 11

GUI Graphical User Interface. 50, 67, 71

HAL Hardware Abstraction Layer. 26, 27, 29, 44–46, 50

HTML Hypertext Markup Language. 65

Akronyme

HTTP Hypertext Transfer Protocol. 52, 65

HTTPS Hypertext Transfer Protocol Secure. 52, 58

I2C Inter-Integrated Circuit. 25, 47, 48

IC Integrated Circuit. 21, 48

ID Identifikator. 27, 45, 46, 55

IPC Interprocess Communication. 69, 76, 77, 80

JS JavaScript. 65

JSON JavaScript Object Notation. 55, 77

LED Light-Emitting Diode. 11, 24, 45, 50

NDEF NFC Data Exchange Format. 21, 22

NDEF-RTD NDEF Record Type Definition. 21, 22

NFC Near Field Communication. 21–24, 26, 28, 41, 42, 47, 48, 51, 74, 75

PCB Printed Circuit Board. 26

PLA Polylactic Acid. 17

PWM Pulse Width Modulation. 24

QML Qt Modeling Language. 79, 80

REST Representational State Transfer. 52, 66

SPI Serial Peripheral Interface. 23–25, 27–29, 43

Akronyme

TCP Transmission Control Protocol. 66, 76, 77

TLS Transport Layer Security. 58, 59

UCI Universal Chess Interface. 67

UI User Interface. 74

USB Universal Serial Bus. 10, 14, 43–48, 51

WLAN Wireless Local Area Network. 10, 14, 51

X-FEN Extended Forsyth-Edwards-Notation. 63, 64, 67

Abbildungsverzeichnis

5-1	Grove PN532 NFC Reader mit Kabelgebundener Antenne	17
5-2	3D Druck: Objekt (rot,gelb,grün),Tree Structure (cyan)	18
6-1	Prototyp Hardware: Erster Prototyp des autonomen Schachtisch	20
6-2	Zwei Elektromagnete. Schlitten befindet sich jeweils in den Ecken	20
6-3	Prototyp Hardware: Tool zur Erstellung des NDEF Payloads: ChessFigureIDGenerator.html	22
6-4	Prototyp Hardware: NDEF Text Record Payload für einen weißen Turm	23
6-5	Prototyp Hardware: Blockdiagramm	24
6-6	Prototyp Hardware: Schaltplan und finaler PCB Entwurf	25
6-7	Prototyp Hardware: Aufbau der Lochrasterplatine	26
7-1	Production Hardware: Finaler autonomer Schachtisch	36
7-2	Schachfiguren im Vergleich	41
7-3	Production Hardware: Blockdiagramm	44
8-1	Gesamtübersicht der verwendeten Cloud-Infrastruktur	53
8-2	Cloud-Infrastruktur: Aufbau einer URI	53
8-3	Cloud-Infrastruktur: Aufbau der Service Architecture	54
8-4	Cloud-Infrastruktur: Backend Login-Request und Response	58
8-5	MoveValidator: Beispiel Request zur Ausführung eines Zuges auf einem gegebenen Schachbrett	63
8-6	Webclient: Spielansicht	65
8-7	Webclient: Statistiken	67
9-1	Embedded System Software: Ablaufdiagramm	70
9-2	Embedded System Software: Figur Wegpunkte	74
9-3	Embedded System Software: Schachfeld Scan Algorithmus Ablauf	75
9-4	Embedded System Software: User-Interface Mockup	78

Tabellenverzeichnis

2.1	Auflistung kommerzieller autonomer Schachtische	7
2.2	Auflistung von Open-Source Schachtisch Projekten	10
3.1	Auflistung der Anforderungen an den autonomen Schachtisch	14
5.1	Verwendete 3D Druck Parameter. Temperatur nach Herstellerangaben des verwendeten PLA Filament.	18
6.1	TMC5160 Beschleunigungskurve / RAMP Parameter	29
7.1	Standardhardware 3D Drucker Steuerungen	43
7.2	Grundlegende verwendete G-Code Kommandos	45
7.3	Hinterlegte G-Code Steuerungen	46
7.4	Hinterlegte Mikrocontroller	49
7.5	Eigenschaften die finalen Prototypen	51
8.1	MoveValidator-Service API Overview	62
8.2	Vergleich FEN - X-FEN	63

A Anhang