



PE-Bug

PNEUMATIC ELECTRONIC BUG

DIPLOMARBEIT

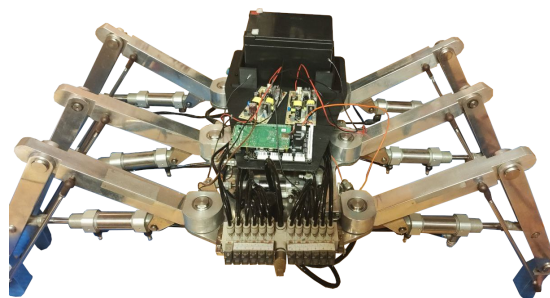
JONATHAN BOGENSPERGER

ROMAN HACKHOFER

CHRISTOPH PUCHER

5BHEL

BETREUER: DI ROLAND NAUTA



17. Mai 2021

Zusammenfassung

English

The aim of this thesis is the renovation of an old project of the Bulme called “Kunstprojekt Spinne“. This process consists of three parts. The first section is concerned with the mechanical and pneumatic parts. The renewal and improvement of the electric hardware is central to the second part. The third and last part is dedicated to the software. The focus of the project is to get the “Spinne“ back to a state in which it is usable for laboratory classes. Keeping the operational complexity down is key to making it accessible for laboratory lessons. Furthermore it could be exhibited at various events like open days. Easy use helps to make it more accessible for viewers. This work is directed towards people that are going to work with the “PE-Bug“ in the future.

Deutsch

Die vorliegende Diplomarbeit setzt sich die Renovierung eines alten Projektes der Bulme, das Kunstprojekt Spinne, als Ziel. Diese setzt sich aus mehreren Teilen zusammen. Der erste Teil befasst sich mit den mechanischen und pneumatischen Bestandteilen. Der Mittelpunkt des zweiten Teiles ist die elektronische Erneuerung und Verbesserung. Der dritte und letzte Teil widmet sich der Software. Das Hauptaugenmerk dieser Arbeit liegt darauf, die Spinne in einen Zustand zu bringen, indem sie wieder für den Laborunterricht benutzbar ist. Es ist hierbei wichtig die Bedienung einfach genug zu gestalten, damit Übungen in der begrenzten Laborzeit sinnvoll sein können. Ein weiterer Einsatzzweck ist das Präsentieren bei Schulveranstaltungen wie dem Tag der offenen Tür. Auch in diesem Fall ist die einfache Bedienung hilfreich, da es die Interaktivität fördert. Diese Arbeit richtet sich an alle, die in der Zukunft mit dem PE-Bug arbeiten.

Inhaltsverzeichnis

1	Einleitung	9
2	Aufgabenstellung	11
2.1	Mechanische Hardware	11
2.2	Pneumatische Hardware	11
2.3	Elektronische Hardware	11
2.4	Software	12
2.4.1	Benutzerinterface	12
2.4.2	API	13
3	Konzept	14
3.1	Warum nicht wie vorher?	14
3.2	Esp32	14
3.3	Arduino	15
3.4	Raspberry Pi	16
3.5	Raspberry Pi und Arduino	16
3.6	Akkutechnologien vergleichen	17
4	Umsetzung	18
4.1	Pneumatiksystem	19
4.1.1	Bestandsystem	19
4.1.2	Änderungen an dem Pneumatiksystem	21
4.2	Stromversorgung	22
4.2.1	DC/DC Netzteile	22
4.2.2	Akku	23
4.3	Ultraschallabstandsensoren	24
4.4	Hauptplatine	24
4.4.1	Leiterplattenabmessungen	24
4.4.2	Steckersystem	25

4.4.3	Ein- und Ausgänge	26
4.4.4	Weitere Anschlüsse	27
4.4.5	Kondensatoren für den Quartz	28
4.4.6	Levelshifter	28
4.4.7	Spannungsanzeige LEDs	29
4.4.8	Magnetventile schalten	31
4.4.9	Produktion der Leiterplatte	33
4.4.10	Liste der Bauteile	34
4.5	PE-Bug Leiterplatten Prototyp	34
4.6	ICSP Programmierer	36
4.7	Mechanische Teile	37
4.7.1	Befestigungsplatte	37
4.7.2	Lüfterbefestigung	37
4.7.3	Akkubefestigung	39
4.8	Bewegung	40
4.9	Software	42
4.9.1	Aufgabenteilung	42
4.10	Protokoll	42
4.10.1	Aufbau	43
4.10.2	Gruppen	43
4.10.3	Quittierung	44
4.11	Firmware	45
4.11.1	Standard Modus	45
4.11.2	Manueller Modus	46
4.11.3	Instruktionen	47
4.11.4	Einlesen über die serielle Schnittstelle	50
4.11.5	Service Routine	51
4.12	Raspberry PI	53
4.12.1	SSH	53
4.12.2	Systemd	53
4.13	Golang Backend	54
4.13.1	Serielle Schnittstelle	54
4.13.2	HTTP	58
4.14	Benutzteroberfläche	60
4.14.1	HTML	60

4.14.2	CSS	60
4.14.3	Javascript	60
5	Ergebnisse	62
5.1	Mechanische Ergebnisse	62
5.1.1	Befestigungsplatte	62
5.1.2	Akkubefestigung	62
5.2	Leiterplatte	62
5.3	Pneumatik	63
5.4	Firmware Ergebnisse	63
5.4.1	Alte Firmware	63
5.4.2	Neue Firmware	64
5.5	Raspberry Pi Software	64
5.5.1	Weboberfläche	64
5.5.2	Backend	64
6	Resümee	66
6.1	Hardware	66
6.2	Software	66
6.3	Probleme	67
A	Schaltplan	68
B	Leiterplattenlayout	71
C	3D gedruckte Befestigungen	73
D	Pneumatische Schaltung	75
E	Golang Server	77
E.1	main.go	77
E.2	routes.go	79
F	Website Code	82
G	Alte Firmware	86
G.1	Pinbelegung	86
G.2	Einstellungen	87
G.3	Hauptprogramm	87

H	Neue Firmware	105
H.1	Pinbelegung	105
H.2	Einstellungen	106
H.3	Instruktionen	106
H.4	Hauptprogramm	110
I	Programm zur Instruktionswandlung	125
I.1	Hauptprogramm	125

Abbildungsverzeichnis

1.1	BULME Skulptur mit Glaskäfig für den PE-Bug[4]	10
2.1	Die alte Leiterplatte, die ersetzt werden muss.	12
4.1	Blockschaltbild	18
4.2	Kompressoren für das Pneumatiksystem	19
4.3	Symbol eines 5/2-Wegeventil	20
4.4	Symbol eines 5/2-Wegeventil	20
4.5	SMC CD85N16-10C-B Pneumatikkolben	21
4.6	Druckluft Wartungseinheit	21
4.7	geplatzter Druckluftschlauch	22
4.8	Der PE-Bug von oben betrachtet mit neuen Schläuchen für das Pneumatiksystem	23
4.9	Zusammenhang zwischen dem gemessenen Abstand und der Ausgegebenen Spannung[5]	24
4.10	Sechs Pin mini ICSP Header mit dem man einen ATmega328P programmieren kann	28
4.11	Benötigte Kondensatorwerte für Quartzfrequenzen[11, S. 39]	28
4.12	Spannungsteiler für die Serielle Kommunikation zwischen dem Microcontroller und dem Raspberry Pi	29
4.13	Schaltung für den Levelshifter, für die Serielle Kommunikation zwischen dem Microcontroller und dem Raspberry Pi	30
4.14	Kennlinie der LED aus dem Datenblatt[8]	31
4.15	Messung des Stroms, den das Magnetventil benötigt	32
4.16	Schaltung mit der die Magnetventile gesteuert werden	32
4.17	Anordnung der MOSFETs auf der Leiterplatte	34
4.18	Leiterplattenprototyp auf einer Lochrasterplatine	36
4.19	ICSP Programmierer mit einem Arduino Nano	36
4.20	Noch nicht gefärbte Holzplatte mit dem 3D gedruckten Akkuhalter	37

4.21 Die gefärbte, und am PE-Bug angeschraubte, Befestigungsplatte mit dem Akku, den DC/DC Netzteilen und der Leiterplatte montiert	38
4.22 3D gedrucktes Kunststoffteil, um den Lüfter auf dem Board zu befestigen .	38
4.23 3D Modell des Akkuhalters für den PE-Bug	39
4.24 Insekten Bewegung	40
4.25 Beispiel Transaktion	42
4.26 Aufbau eines Befehls	43
4.27 Ausgabe mit Testdaten	45
4.28 Automatische Sensordatenausgabe	46
4.29 Alte Instruktion	48
4.30 Aufbau der alten Instruktion	48
4.31 Neue Instruktion	49
4.32 Bit Tabelle	49
4.33 Umwandlung der Instruktion	50
4.34 Ausgabe einer Dezimalzahl in Binär	50
4.35 Neue Instruktion	51
4.36 Fortsetzungs-Bit == 1	52
4.37 Abfrage der Millisekunden	52
4.38 Unit-File für den Backend-Server [pebugbackend.service]	53
4.40 CLI-Argument Implementierung	54
4.39 Golang Logo	54
4.41 Initialisierung der seriellen Schnittstelle	55
4.42 Fehlerbehandlung	55
4.43 Empfang und Verarbeitung	55
4.44 Auswertung	56
4.45 nebenläufige Schleife	56
4.46 Sende Schleife	57
4.47 Senden mit Kanal	57
4.48 http.HandlerFunc Typdefinition	58
4.49 Registrierung einer http.HandlerFunc	58
4.50 Starten des HTTP-Servers	58
4.51 Funktion für Sensorwerte	59
4.52 Funktion die Funktion für Bewegung zurück gibt	59
4.53 Starten des HTTP-Servers	59
4.54 HTML Body	60

4.55	Einbinden des CSS	60
4.56	CSS für den Container	61
4.57	Rasterlayout mit Debuglinien	61
4.58	Einbinden des Javascript	61
4.59	Zusweisung der OnClick-Funktionen	61
5.1	Der Stecker für die Spannungsversorgung, ohne dem Silkscreen.	63
5.2	Microcontroller mit hochgeladener Firmware.	64
5.3	Finale Version der Weboberfläche	65
5.4	Knopf mit zugewiesenem Event	65
5.5	Inhalt der Anfrage (rot) und der Antwort (blau)	65
5.6	TCP Verlauf der Anfrage	65
5.7	Anfrage im Netzwerkanalyse Tabs des Firefox Debuggers	65
B.1	Top Seite der Leiterplatte	72
B.2	Bottom Seite der Leiterplatte	72
C.1	3D gedruckte Befestigung für den Akku	74
C.2	3D gedruckte Befestigung für den Lüfter	74
F.1	index.html	83
F.2	style.css	84
F.3	script.js	85

Tabellenverzeichnis

3.1	Pro-und-Contra Liste des ESP32	15
3.2	Pro-und-Contra Liste des ATmega328p	15
3.3	Pro-und-Contra Liste des Raspberry Pis	16
3.4	Pro-und-Contra Liste ATmega328P mit einem Raspberry Pi	17
3.5	Pro-und-Contra Liste der Akkutechnologien	17
4.1	Bauteile auf der Leiterplatte	35
4.2	Vor- und Nachteile der verschiedenen Bewegungsabläufe	41
4.3	Beschreibung der Befehle	49

Kapitel 1

Einleitung

Der PE-Bug, kurz für Pneumatic-Electronic-Bug, von dem in der vorliegenden Arbeit die Rede sein wird, ist ein circa 40cm langer und 80cm breiter Roboter aus Aluminium. Früher wurde der PE-Bug Spinne genannt, aber sie hat, anders als der Name vermuten lässt, keine Beziehung zur Biologischen Namenskollegin. Deshalb wurde entschieden den Namen auf PE-Bug zu ändern, was für Pneumatic Electronic Bug steht, denn das ehemalige Maskottchen der BULME hat nur sechs Beine, und Netze kann sie auch nicht spannen. Die Mobilität kommt durch ihr Pneumatiksystem zustande, und Energie wird aus einem Akku bezogen.

Die Spinne wurde 2001 im Rahmen eines Kunstprojekts gebaut. Zusammen mit der Spinne wurde auch eine, vom Architekten und Maler Bruno Wildbach entworfene, Skulptur 1.1 mit integriertem Glaskäfig errichtet, in dem sich der Roboter bewegen konnte[4]. Der Idee des Künstlers entsprechend, wurde sie durch Solarenergie betrieben, und die Bewegungen mit Hilfe eines Lichtschrankens ausgelöst[22]. Aufgrund des Erfordernisses eines modernen Werkstattgebäudes, musste jedoch die Skulptur abgerissen werden, wobei die Spinne schon mehrere Jahre davor dort nicht mehr eingesetzt wurde. Die Spinne wurde unter einem Werkstättentisch gelagert, wo sie unsere Neugierde erweckte, schließlich ist sie älter als jedes einzelne Teammitglied unserer Diplomarbeitsgruppe. Durch das Gespräch mit einem Fachlehrer kam der Vorschlag, die Spinne in ein Diplomarbeitsprojekt zu integrieren. Aufgrund der großen Veränderungen in der Elektronik während der letzten beiden Jahrzehnte war allen sofort klar, dass der Schwerpunkt unserer Arbeit darin liegen wird, den Roboter auf den aktuellen Stand der Technik zu bringen.

Weil gerade in der Technik, Tätigkeiten an ihrer Relevanz gemessen werden, stellte sich auch für uns die Frage nach möglichen Anwendungen, gerade im Hinblick darauf, dass der PE-Bug Eigentum der BULME ist, und diese den finanziellen Rahmen des Projekts zur Verfügung stellt. Da für so einen Roboter keine unmittelbare Verwendbarkeit



Abbildung 1.1: BULME Skulptur mit Glaskäfig für den PE-Bug[4]

besteht, ging unser Nachdenken in die Richtung ihn öffentlichkeitswirksam einsetzbar zu machen. Der Tag der offenen Tür, der an der BULME alljährlich stattfindet, lockt zahlreiche Besucher an, 2021 aufgrund der Pandemie zwar nur virtuell, und es ist das Ziel der Schule sich möglichst vielseitig und umfassend zu präsentieren. Gerade für die Abteilung Elektronik und Technische Informatik ist es oft schwer ihre Inhalte plakativ und publikumswirksam zur Schau zu stellen. Mit diesem Projekt wird es gelingen jene Teile, die zwar für den Fachbereich interessant und wesentlich, aber schwer darstellbar sind auf eine ansprechende Weise zu präsentieren. Den Besuchern des Tag der offenen Tür wird mit dem Ausstellungsstück veranschaulicht, wie das in diesem Schulzweig vermittelte Wissen angewendet werden kann und hilft vielleicht bei der Schulwahl der Besucherinnen und Besucher.

In der vorliegenden Arbeit wird der Werdegang dokumentiert, des weiteren werden Gedankengänge und Irrwege aufgezeigt. Der intensive Austausch innerhalb unseres Teams hat dazu geführt, dass wir einander mit guten Ideen motiviert haben, bestmöglich unser Vorhaben umzusetzen.

Kapitel 2

Aufgabenstellung

Das Ziel des Projektes ist die Wiederherstellung der Spinne zu einem funktionsfähigem Apparat und des weiteren auch eine Modernisierung. Diese Aufgabe unterteilt sich in mehrere Aufgabengebiete. Der notwendige Arbeitsaufwand variiert stark zwischen den Gebieten. Die elektronische Hardware sowie die Software sind unbrauchbar bis nicht vorhanden und müssen vollständig neu gebaut werden. Die mechanische und pneumatische Hardware hingegen sind noch sehr gut erhalten und benötigten nur kleine Reparaturen.

2.1 Mechanische Hardware

Die mechanische Hardware ist vollständig. Alle Gelenke und Lager laufen noch reibungslos und es gibt keine gebrochenen Stellen. Obwohl alles funktionsfähig ist, ist die Spinne noch sehr verstaubt und schmutzig und muss gereinigt werden. Wenn die neue elektrische Hardware nicht mit den bestehenden Befestigungsmöglichkeiten kompatibel ist, müssen diese noch geschaffen werden.

2.2 Pneumatische Hardware

Die pneumatische Hardware der Spinne ist größtenteils in einem guten Zustand. Es wird vermutlich notwendig sein, die Schläuche zu tauschen, aber die anderen Bauteile, wie zum Beispiel die Ventile, die Kolben und die Kompressoren kann man weiterverwenden.

2.3 Elektronische Hardware

Wie schon erwähnt, ist die elektronische Bestandshardware so gut wie unbrauchbar 2.1. Die Steuerung der Spinne wird mit einem Microcontrollerboard von Conrad realisiert,

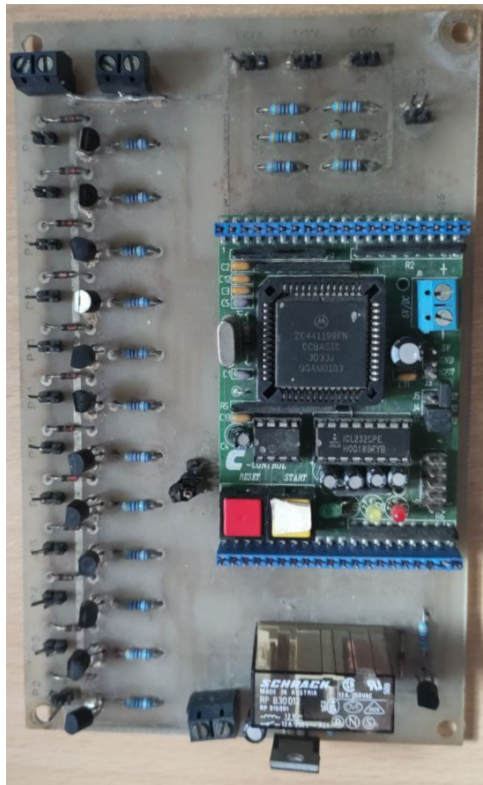


Abbildung 2.1: Die alte Leiterplatte, die ersetzt werden muss.

das einen Controller von Motorola verwendet, der im Internet nicht auffindbar ist. Im Elektronikbereich hat sich in den letzten 20 Jahren zu viel verändert, was bedeutet, dass die Elektronik für die Spinne neu gebaut werden muss.

2.4 Software

Die Software soll die Bewegungsabläufe des PE-Bugs ausführen können, eine Möglichkeit zur Fernsteuerung bieten und Sensordaten zurück liefern. Eine weitere Funktion soll die Erweiterbarkeit von Außen durch eine API sein. Da die alte Software nicht auffindbar ist, müssen alle Funktionen neu erarbeitet werden.

2.4.1 Benutzerinterface

Um bei Präsentationen das Projekt auch interaktiv gestaltbar zu machen, ist es sehr wichtig ein gutes Benutzerinterface zu haben. Besonders wichtig ist auch, dass dieses sehr einfach und auf keinen Fall komplex ist. Die Bedienung soll mit einem ferngesteuerten Auto vergleichbar sein.

2.4.2 API

Damit Schüler nicht jedes Jahr das Rad neu erfinden müssen, ist die Inklusion einer API essentiell. Schüler sollen damit das Projekt erweitern oder verbessern können, ohne eine neue Software für den PE-Bug selbst zu schreiben. Es soll zwei Möglichkeiten geben den PE-Bug zu kontrollieren: einen einfachen Modus in dem man Kommandos wie vor und zurück gibt und einen erweiterten Modus in dem man volle Kontrolle über alle Sensoren und Aktoren hat. Eine Unterteilung gibt es auch bei der Zugänglichkeit der API, diese soll einerseits mit Fernzugriff verfügbar sein sowie in einer hardwarenahen Konfiguration.

Fernzugriff API

Diese API ist für Programme gedacht, die einen sehr hohen Rechenaufwand haben (z.B.: Computervision, Artificial Intelligenz). Solche Programme können dann auf einem externen Computer ausgeführt werden. Eine solche API ermöglicht dem PE-Bug auch mit zukünftigen Technologien ansteuerbar zu sein.

Hardwarenahe API

Die hardwarenahe API ist für Code der direkt auf dem PE-Bug ausgeführt werden soll. Diese API eignet sich vor allem für Projekte, die in sich geschlossen sein sollen oder besondere Timing-Anforderungen haben.

Kapitel 3

Konzept

3.1 Warum nicht wie vorher?

Ein wichtiger Aspekt des Projektes war zu erfassen, welche Funktionen die Spinne früher besaß. Daraus ergab sich die Frage, ob und warum wir die Spinne nicht auf einen Status bringen, der den der vorherigen relativ exakt widerspiegelt. Die alte Elektronik der Spinne war ein in sich geschlossenes System, dass das auf dem Controller gespeicherte Programm ausführte. Diese Funktionalität war für ihren damaligen Einsatz ausreichend, jedoch nicht für unsere Vision. Die Geschlossenheit des Systems macht es schwer, die Spinne interaktiv zu präsentieren und die Möglichkeiten sind auch auf die Leistung des verbauten Mikrocontrollers beschränkt. Wir wollten die Spinne moderner und zugänglicher machen. Dafür ist aber mindestens eine Anbindung nach außen nötig. Da eine solche Anbindung bei der alten Version nicht vorgesehen war, entscheiden wir uns gegen eine Neuauflage und für eine komplett neue Version.

3.2 Esp32

Der ESP-32 ist ein sehr populärer Chip für Internet of Things Anwendungen. Mit integriertem Wlan und Bluetooth ist es ein gut für fernsteuerbare Projekte geeigneter Chip. Auch die IO Fähigkeiten sind ausreichend für das Projekt. Der Zweikernprozessor ermöglicht Wlan oder Bluetoothoperation ohne das Programm regelmäßig unterbrechen zu müssen. Over The Air Updates, auch OTA-Updates, sind ein weiteres Argument für den ESP, da es den Entwicklungsprozess vereinfacht. Jedes Teammitglied hat schon einmal mit einem ESP sehr grundlegende Programme geschrieben, jedoch hat der ESP eine sehr steile Lernkurve und bei etwas größeren Projekten explodiert die Komplexität der Entwicklungsumgebung. Diese Komplexität ist im Endeffekt auch der Grund, warum diese

Lösung sehr früh aus dem Entscheidungsprozess ausgeschieden ist, da wir mit anderen Technologien viel besser vertraut sind.

Tabelle 3.1: Pro-und-Contra Liste des ESP32

Pro	Contra
WLAN	Komplexere Programmierumgebung
Bluetooth	Nur oberflächliche Erfahrung mit ESP
OTA Updates	Hardware
2-Kern Prozessor	Wenig Speicher für Benutzeroberfläche

3.3 Arduino

Im Gespräch war ebenfalls der Neuaufbau mit einem Arduino, speziell dem ATmega328p. Mit seinen 32 Pins verfügt der ATmega328p über genügend Pins um alle Beine und bestehenden Sensoren anzusprechen sowie einen Anschluss zur Außenwelt zu gewährleisten. Zur digitalen Kommunikation stellt der ATmega328p sowohl UART als auch SPI und I2C zur Verfügung. Vorteil dieser Lösung ist, dass der ATmega328p kostengünstig ist und durch den Unterricht jedem Mitglied dieser Diplomarbeitgruppe bekannt ist. Durch die Arduino IDE ist ebenfalls gewährleistet das zukünftige Benutzer den μC ohne größere Probleme programmieren können. Allerdings wäre das Projekt insofern eingeschränkt, da der ATmega328p nur über einen Kern verfügt und Wlan und Bluetooth nicht am ATmega328p realisiert sind. Diese müssten über einen externen Chip realisiert werden. [11]

Tabelle 3.2: Pro-und-Contra Liste des ATmega328p

Pro	Contra
Kostengünstig	Kein Bluetooth
Erfahrung mit der Arduino Hardware	Kein Wlan
Kostenlose Arduino IDE	Nur ein Prozessorkern

3.4 Raspberry Pi

Neben den Microcontrollern ESP32 und ATmega328P wurde auch überlegt, einen Raspberry Pi zu verwenden, der wie der ESP32 WLAN und Bluetooth hat, aber wie ein Desktop PC über Gigabit Ethernet und USB Anschlüsse verfügt. Im Vergleich zu den Microcontrollern ist der Raspberry Pi 4 auch um einiges schneller, er hat vier ARM CPU Kerne und bis zu acht Gigabyte RAM[13]. Und weil der Raspberry Pi unter GNU+Linux läuft, kann man Programme in jeder Programmiersprache schreiben. Alleine unter den drei Teammitglieder, kennen sich alle in unterschiedlichen Sprachen am besten aus. Jedoch hat GNU+Linux den Nachteil, dass es kein Echtzeitbetriebssystem (RTOS) ist, und deshalb für das Steuern von Geräten nicht optimal ist. Ein weiterer Nachteil ist, dass der Raspberry Pi zum einlesen von analogen Werten einen externen ADC[20] bräuchte.

Tabelle 3.3: Pro-und-Contra Liste des Raspberry Pis

Pro	Contra
802.11ac WLAN	Kein RTOS
Bluetooth 5.0	Kein ADC
GNU+Linux	
USB und Gigabit Ethernet	
Schneller Prozessor und viel RAM	

3.5 Raspberry Pi und Arduino

Wenn man die Vor- und Nachteile von ATmega328P und Raspberry Pi vergleicht, sieht man, dass diese sich sehr gut ergänzen. Daraus entstand die Idee beide zu benutzen. Die beiden Prozessoren mit einander kommunizieren zu lassen, ist sehr trivial, da beide über UART verfügen. UART ist eine simple Schnittstelle, auf der man ASCII-Charaktere überträgt. Die Schnittstelle ist auch in der Software sehr einfach zu bedienen. Das Leiterplattenlayout muss natürlich ATmega328P und Raspberry Pi beinhalten. Es muss auch Software für beide Systeme geschrieben werden. Die Kosten erhöhen sich nur marginal, da ein ATmega328P kostengünstig ist.

Tabelle 3.4: Pro-und-Contra Liste ATmega328P mit einem Raspberry Pi

Pro	Contra
802.11ac WLAN	Komplexe Leiterplatte
Bluetooth 5.0	Software für zwei Geräte
GNU+Linux	
USB und Gigabit Ethernet	
Schneller Prozessor und viel RAM	
Erfahrung mit beiden Umgebungen	
Echtzeitfähigkeit	

3.6 Akkutechnologien vergleichen

Beim alten Kunstprojekt wurde die Spinne über Kabel von einem stationären Bleiakku mit Strom versorgt. Die Überlegung war, den 12V Bleiakku durch einen Lithium-Ionen Akku zu ersetzen, der leichter wäre. Jedoch haben die höheren Kosten, und die erhöhte Gefahr, wenn man den Akku außerhalb der Vorgaben betreibt, dagegen gesprochen. In der Tabelle 3.5 sind die Vor- und Nachteile eines Lithium-Ionen Akkus angeführt. Falls der Bleiakku eine nicht zufriedenstellende Akkulaufzeit hat, ist es möglich, den Akku durch einen Lithium Akku zu tauschen, der eine höhere Energiedichte hat.

Tabelle 3.5: Pro-und-Contra Liste der Akkutechnologien

Pro	Contra
Hohe Energiedichte	Hohe Kosten
Niedriges Gewicht	Erhöhte Gefahr bei falschem Betrieb

Kapitel 4

Umsetzung

In diesem Kapitel wird die Umsetzung des PE-Bugs beschrieben. Der Zusammenhang der Teile, die in dem Kapitel beschrieben werden, ist im Blockdiagramm 4.1 beschrieben. In rot ist gezeigt, wie der PE-Bug mit Strom versorgt wird, in grün kann man die Daten- und Steuerleitungen sehen, wobei Pfeile in beide Richtungen bedeutet, dass ein Datenaustausch stattfindet und in Blau sind die Verbindungen eingezeichnet, über die Druckluft übertragen wird.

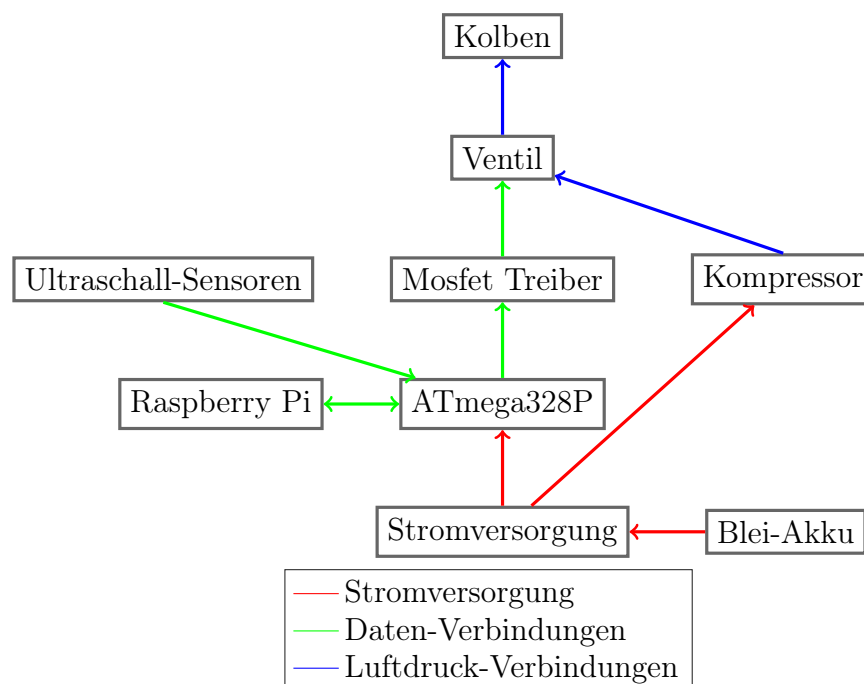


Abbildung 4.1: Blockschaltbild

4.1 Pneumatiksystem

4.1.1 Bestandsystem

Mit dem Pneumatiksystem werden beim PE-Bug die sechs Beine bewegt. Pro Bein werden zwei Pneumatikkolben eingesetzt, die die Beine drehen können, und anheben beziehungsweise absenken können. Mit diesen zwei Bewegungsmöglichkeiten pro Bein, kann sich der PE-Bug fortbewegen. Gesteuert werden die Kolben durch zwölf 5/2-Wegeventile. Das bedeutet, dass die Ventile fünf Anschlüsse, und zwei Zustände haben. Mit Druckluft wird das System durch zwei 12V Kompressoren versorgt.

Kompressoren

Die im PE-Bug verwendeten Kompressoren sind zwei NPK 09 DC in der 12V Version, von KNF Neuberger. Sie haben einen maximalen Strom von 5,2A, haben einen Durchfluss von 15 Liter pro Minute und können bei einem Druck von bis zu 7 Bar arbeiten[7].

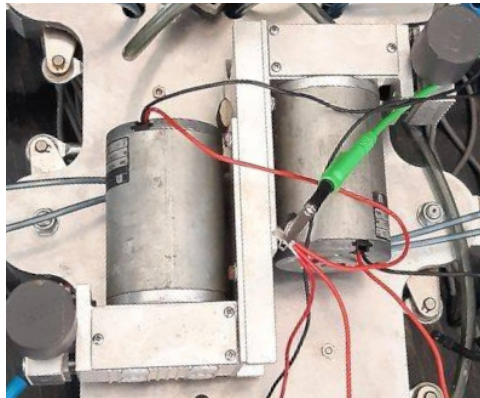


Abbildung 4.2: Kompressoren für das Pneumatiksystem

5/2-Wegeventile

Die Pneumatikkolben des PE-Bugs werden mit 5/2-Wegeventilen gesteuert, bei denen es sich um die SY114-6LOU-Q Magnetventil von SMC handelt. Wie schon erwähnt haben 5/2-Wegeventile fünf Anschlüsse und zwei Stellungen. Zwei der Anschlüsse sind Ausgänge für die Druckluft, die zu den Kolben gehen, ein Anschluss ist die Versorgung von den Kompressoren und aus den letzten beiden Anschlüssen, kann die Luft wieder entweichen. Das Schaltsymbol des 5/2-Wegeventil erklärt die Funktion sehr gut. Das Ventil ist in vier Teile geteilt, in der Abbildung 4.3 in Farben gekennzeichnet. Die beiden äußeren Teile verändern den Zustand des Ventils, und die beiden inneren Teile sind die möglichen Zustände. Der violette Teil eins ist der Elektromagnet, mit dem der Zustand verändert

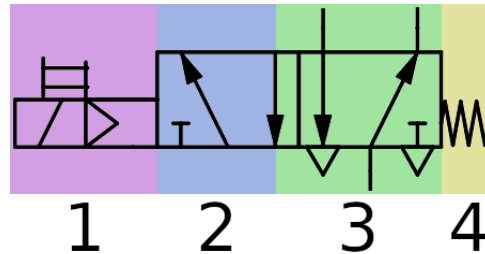


Abbildung 4.3: Symbol eines 5/2-Wegeventil

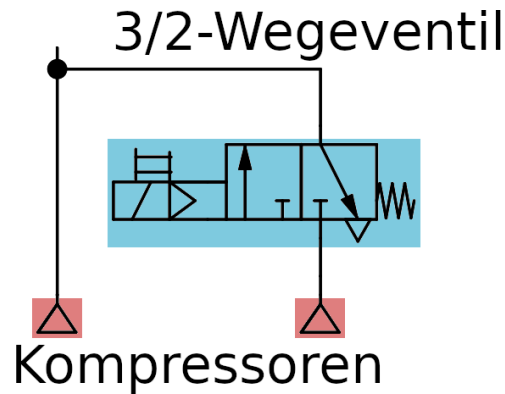


Abbildung 4.4: Symbol eines 5/2-Wegeventil

werden kann, und der gelbe Teil vier ist eine Feder, die den Zustand des Ventils, wenn der Elektromagnet nicht aktiv ist, in den Ausgangszustand zurück setzt. Die Teile zwei und drei sind die beiden möglichen Zustände. Wenn der Elektromagnet inaktiv ist, sind die fünf Anschlüsse, wie im grünen Teil gezeigt, verbunden, und wenn der Elektromagnet aktiviert wird, werden die Anschlüsse, wie im blauen Teil, gezeigt verbunden.

In der Abbildung 4.3 sind unten drei Anschlüsse eingezeichnet (die beiden Dreiecke sind Anschlüsse die an nichts angeschlossen sind) und oben zwei. Der mittlere Anschluss unten ist der Drucklufteingang, und kann mit den oberen Anschlüssen, die zu einem Kolben gehen, verbunden werden. Wenn in einer Seite des Kolbens Luft hineingepumpt wird, kommt aus der anderen Seite des Kolbens Luft heraus, die über die beiden äußeren Anschlüsse unten entweichen kann.

3/2-Wegeventile

Die Ausgänge der beiden Kompressoren sind nicht direkt zusammengeschlossen, sondern gehen durch ein 3/2-Wegeventil 4.4. Wenn 12V an dem Ventil angelegt werden, schaltet das Ventil, und die Ausgänge der beiden Kompressoren sind verbunden. Das ist vermutlich dazu da, dass nach dem Betrieb der Druck aus den Leitungen gelassen wird.

Pneumatikkolben

Die verwendeten Pneumatikkolben 4.5 sind SMC CD85N16-10C-B der SMC CD85 Serie.

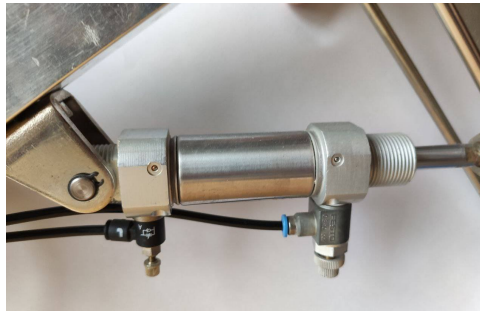


Abbildung 4.5: SMC CD85N16-10C-B Pneumatikkolben

Druckluft Wartungseinheit

Das Pneumatiksystem hat auch eine Druckluft Wartungseinheit 4.6 integriert. Die Druckluft Wartungseinheit besteht aus einem Luftfilter mit Ablauf und einem Druckregler. Das linke Symbol, im Druckluft Wartungseinheit Schaltsymbol, ist der Luftfilter mit Ablauf, und auf der rechten Seite ist der Druckregler.

4.1.2 Änderungen an dem Pneumatiksystem

5/2-Wegeventil

Beim Testen der Ventile stellte sich heraus, dass es zwar möglich war, alle mechanisch zu schalten, aber eines der Ventile ließ sich nicht elektrisch schalten. Nachdem klar war, dass ein neues Ventil gebraucht wird, wurde das fehlerhafte auseinandergenommen, und es stellte sich heraus, dass der Elektromagnet keine Verbindung zum Stecker hatte. Es war aber nicht möglich, den Fehler zu reparieren. Um in Zukunft weitere Ventile austauschen zu können, wurden drei Ersatzventile bestellt.

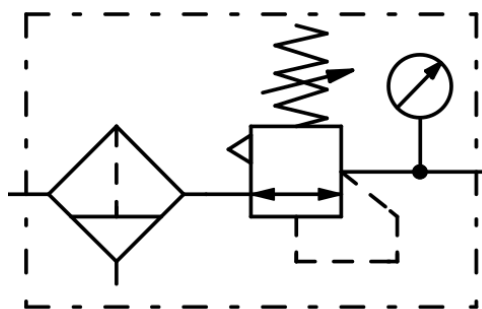


Abbildung 4.6: Druckluft Wartungseinheit

Neue Schläuche

Schon von Beginn des Projekt war es klar, dass die 20 Jahre alten Schläuche des Pneumatiksystems getauscht werden müssen. Viele Schläuche sind vergilbt, und die Leitungen waren sehr fest. Nachdem beim testen des Pneumatiksystems auch ein Druckluftschlauch gelplatzt ist, 4.7 wurden alle gegen neue schwarze Schläuche 4.8 getauscht.



Abbildung 4.7: geplatzter Druckluftschlauch

4.2 Stromversorgung

4.2.1 DC/DC Netzteile

Um den PE-Bug zu betreiben braucht man vier Spannungen, 3,3V für den Raspberry Pi, 5V für den Raspberry Pi und den ATmega328P Microcontroller, 12V für die Magnetventile und die Kompressoren und 24V für die Ultraschallsensoren. Die 3,3V werden vom Raspberry Pi mit einem eigenem DC/DC Netzteil aus 5V generiert, 12V gibt es aus dem Blei Akku, der den PE-Bug mit Strom versorgt. 5V und 24V kann man mit DC/DC Netzteilen aus 12V erzeugen. Die dafür gewählten DC/DC Netzteile sind das Mean Well PSD-15A-05 und das Mean Well PSD-15A-24. Alle Netzteile der Mean Well PSD-15 Serie haben eine Ausgangsleistung von ungefähr 15W. Das PSD-15A-05 hat bei 5V Ausgangsspannung einen maximalen Ausgangsstrom von 3A und das PSD-15A-24 kann bei 24V 0.6A liefern. In der PSD-15 Serie gibt es die PSD-15A*, PSD-15B* und die PSD-15C* Netzteile, die für die Eingangsspannung 12V, 24V und 48V gebaut sind[10]. Da beim PE-Bug die Eingangsspannung 12V ist, werden PSD-15A* Netzteile eingebaut.

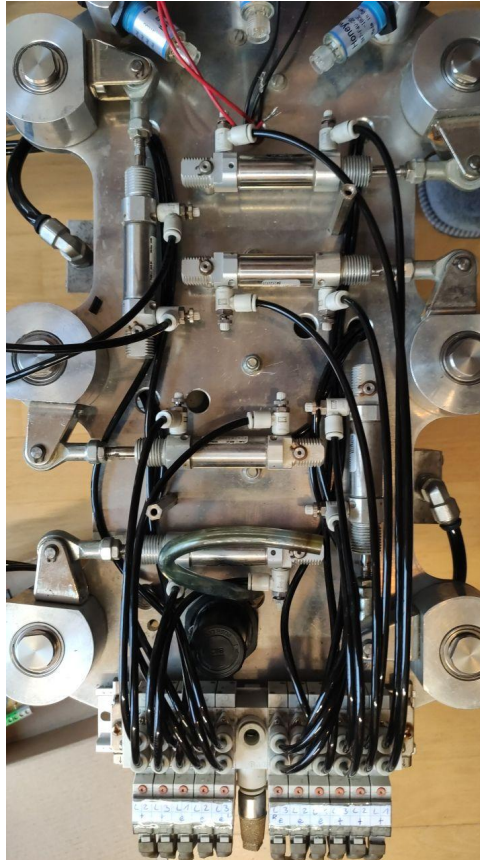


Abbildung 4.8: Der PE-Bug von oben betrachtet mit neuen Schläuchen für das Pneumatiksystem

4.2.2 Akku

Für die Entscheidung, welcher Akku für den PE-Bug geeignet ist, wurde überschlagsmäßig der maximale Strom ausgerechnet, den der Akku liefern muss. Die Verbraucher sind:

- Kompressoren: $2 \cdot 5,2A[7]$
- 5V Komponenten: 3A bei 5V mit 74% Effizienz $\frac{3A \cdot 5V}{0,74} \cdot \frac{1}{12V} = 1,69A$
- Magnetventile: $50mA \cdot 12 = 0,6A$
- Der Stromverbrauch der Ultraschallsensoren ist vernachlässigbar

Zusammengerechnet sind das 12,69A. Der 150-1560 Blei Akku von RS Pro, kann 1C, was in diesem Fall 12A entspricht, für circa 30 Minuten liefern[18], bis die Zellenspannung unter 1,8V geht. Weil man nicht damit rechnen muss, dass durchgehend der maximale Strom gebraucht wird, ist die Laufzeit vom PE-Bug mit dem Akku akzeptabel.

Analogue output

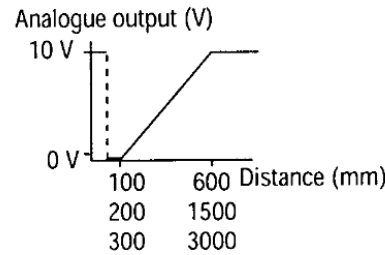


Abbildung 4.9: Zusammenhang zwischen dem gemessenen Abstand und der Ausgegebenen Spannung[5]

4.3 Ultraschallabstandsensoren

Die Ultraschallabstandsensoren, die auf dem PE-Bug schon montiert waren, sind die 947-F4V-2D-1C0-180E von Honeywell. Sie haben einen minimalen Abstand, den sie messen können, von 200mm und einen maximalen Abstand von 1500mm. Die Auflösung ist 0,5mm, und die Sensoren haben eine Betriebsspannung von 24V[5]. Der Sensor hat eine Ausgangsspannung von 0V bis 10V und ist im Bereich von 200mm bis 1500mm linear 4.9.

4.4 Hauptplatine

Auf der Leiterplatte, mit der der PE-Bug gesteuert wird, ist der ATmega328P Microcontroller, der Raspberry Pi und die Ein- und Ausgänge, für die Aktoren und Sensoren, angebracht.

4.4.1 Leiterplattenabmessungen

Die PE-Bug Leiterplatte wird mit dem Raspberry Pi wie ein Pi HAT verbunden. In der Raspberry Pi HAT Mechanical Specification[6] werden die Abmessungen eines Raspberry Pi HATs beschrieben. Weil die PE-Bug Leiterplatte größer als ein Pi HAT ist, sind nur die Abstände zwischen den Befestigungslöchern, die Position der Buchsenleiste und der Durchmesser der Löcher relevant.

Auf dem PE-Bug wird die 100mm*80mm Leiterplatte mit vier M3 Schrauben befestigt. Des weiteren sind auf der PE-Bug Leiterplatte zwei Löcher für M3 Schrauben, um einen Lüfter, für den Raspberry Pi, anzuschrauben.

4.4.2 Steckersystem

Damit man den PE-Bug warten oder umbauen kann, ist es wichtig, dass die Leitungen zu den Magnetventilen, Ultraschallsensoren und den Netzteilen nicht an der Leiterplatte angelötet sind. Es gibt verschiedene Möglichkeiten, die Kabel anzuschließen, und für den PE-Bug wurden die folgenden vier untersucht:

- Stiftleisten und Buchsenleisten
- Schraubklemmen
- Molex Micro-Fit 3.0 System
- Molex KK 254 System

Stiftleisten und Buchsenleisten

Eine mögliche Lösung wäre, wie schon bei der Leiterplatte, die für die Spinne vor 20 Jahren gebaut wurde, Stiftleisten mit Buchsenleisten für die Verbindung zu verwenden. Die Vorteile dieser Lösung sind, dass das System günstig ist, bei lokalen Geschäften verfügbar ist, dass man kein Crimp Werkzeug braucht, um Leitungen an ihnen zu befestigen und, dass man für die Magnetventile, und Ultraschallsensoren keine neuen Stecker löten müsste, weil man die alten Stecker weiterverwenden könnte. Jedoch haben Stiftleisten als Steckersystem zwei Nachteile, die sie für die Anwendung am PE-Bug unbrauchbar machen, zum einen kann man Buchsenleisten relativ leicht von Stiftleisten trennen und zum anderen haben sie keinen Verpolungsschutz.

Schraubklemmen

Die Anschlüsse mit Schraubklemmen teilen einige ihrer Vorteile mit einer Lösung aus Stiftleisten und Buchsenleisten, man kann sie, abhängig vom Geschäft wo man sie kauft, günstig erwerben, und sie sind auch bei lokalen Geschäften verfügbar, was bedeutet, dass man nicht mehrere Wochen auf die Bestellung warten muss. Ein Vorteil, den Schraubklemmen gegenüber einem System aus Stift- und Buchsenleisten haben ist, dass die festgeschraubten Kabel sehr gut fixiert sind, und ein versehentliches Anziehen die Verbindung nicht trennen wird. Aber auch dieses System hat Nachteile, die es für den PE-Bug nicht optimal machen. Für die gleiche Anzahl an angeschlossenen Kabel, brauchen Schraubklemmen doppelt so viel Platz auf der Leiterplattenkante wie Stift- und Buchsenleisten, weil, anders als bei Stift- und Buchsenleisten, haben die einzelnen Anschlüsse nicht einen Abstand von 2,54mm zueinander, sondern 5,08mm, und ein Verpolungsschutz fehlt dem

System auch, was bei unachtsamen Verhalten dazu führen kann, dass die Bauteile zu hohe Spannungen abbekommen oder, dass die Spannungsversorgung verkehrt angeschlossen wird.

Molex Micro-Fit 3.0 System

Das Molex Micro-Fit 3.0 System löst die Probleme, die das System mit den Stift- und Buchsenleisten und die Schraubklemmen haben. Man kann die Stecker nicht ohne weiteres aus den Buchsen ziehen, weil es beim Einstecken einrastet. Des weiteren ist es nicht möglich die Stecker zu verpolen, weil die verschiedenen Pins unterschiedliche Formen haben, und deshalb nur in einer Orientation in die Buchse passen. Und, weil man die Micro-Fit 3.0 Stecker als zweireihige Stecker und Buchsen kaufen kann, braucht eine Buchse, mit sechs Pins, nur 12,65mm[12]. Jedoch haben die Buchsen und Stecker des Micro-Fit Systems relativ hohe Kosten, was der Grund war sich gegen dieses System zu entscheiden.

Molex KK 254 System

Das Molex KK 254 System löst, wie die Micro-Fit Stecker, die Probleme der Stift- und Buchsenleisten und der Schraubklemmen. Um die Verbindung zu trennen, muss man mit recht viel Kraft am Stecker ziehen, es gibt einen Verpolungsschutz und, obwohl es nicht so platzsparend wie das Micro-Fit System ist, braucht es nur so viel Platz wie die Stift- und Buchsenleisten. Weil es fast alle Vorteile der Micro-Fit Stecker und Buchsen hat, aber um einiges weniger kostet, wird für die Leiterplatte das Molex KK 254 System verwendet.

Crimp Terminals

Weil sich die Kosten, für eine Crimpzange bei den wenigen Verbindungen der Leiterplatte, nicht rechtfertigen lassen, sind die Leitungen am PE-Bug an die Crimp Terminals angelötet, was zwar zeitaufwändig war, aber eine zufriedenstellende Verbindung ermöglichte.

4.4.3 Ein- und Ausgänge

Um die Verschiedenen Geräte auf dem PE-Bug zu steuern, braucht die Microcontroller-leiterplatte Ein- und Ausgänge, von denen die Leiterplatte folgende hat:

- 12V Digitale Ausgänge
- 10V Analoge Eingänge
- 5V Digitaler Ein- oder Ausgang

12V Digitale Ausgänge

Um die Magnetventile zu schalten, braucht man 12V Ausgänge. Das wird in 4.4.8 genauer erklärt. Die sechs Beine vom PE-Bug brauchen jeweils zwei Magnetventile, also braucht die Leiterplatte zwölf 12V Digitale Ausgänge. Die Anschlüsse dafür sind vier sechs-Pin Molex KK 254 Stecker. Jeder zweite Pin der Stecker wird für die 12V Spannungsversorgung für die Magnetventile verwendet.

10V Analoge Eingänge

Die Ultraschallsensoren liefern einen analogen Ausgang von 0 bis 10V, jedoch können die ATmega328P ADCs nur Spannungen von 0 bis 5V einlesen. Deshalb werden für die analogen Eingänge 1:2 Spannungsteiler eingesetzt, was bedeutet, dass beide Widerstände des Spannungsteilers, den gleichen Wert haben.

5V Digitaler Ein- oder Ausgang

Der PE-Bug hat einen unbenutzten 5V digitalen Ein- oder Ausgang, der für einen weiteren Sensor oder Aktor verwendet werden kann.

4.4.4 Weitere Anschlüsse

Neben den Ein- und Ausgängen gibt es auf der Leiterplatte noch weitere Anschlüsse.

- AVR mini-ICSP Header
- Spannungsversorgung
- Lüfterstecker

ICSP Header

Um einen ATmega328P Microcontroller zu programmieren, kann man einen ICSP (In Circuit Serial Programmer) Header verwenden. Der verwendete mini ICSP Header hat sechs Pins und eine Pinbelegung, wie in Abbildung 4.10 zu sehen ist.

Spannungsversorgung

Die Leiterplatte wird über einen acht Pin Molex KK 254 Stecker mit Spannung versorgt. Der Stecker hat vier Ground Pins, jeweils einen 12V und 24V Pin und zwei 5V Pins. Für 5V gibt es zwei Pins, weil bei dieser Spannung relativ viel Strom gebraucht wird, der

ICSP Header to program the μC

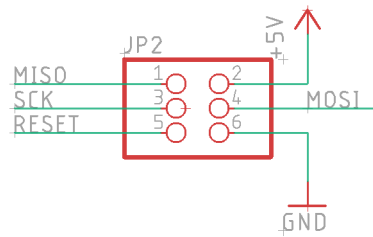


Abbildung 4.10: Sechs Pin mini ICSP Header mit dem man einen ATmega328P programmieren kann

Table 9-3. Low Power Crystal Oscillator Operating Modes⁽³⁾

Frequency Range (MHz)	Recommended Range for Capacitors C1 and C2 (pF)	CKSEL3...1 ⁽¹⁾
0.4 - 0.9	–	100 ⁽²⁾
0.9 - 3.0	12 - 22	101
3.0 - 8.0	12 - 22	110
8.0 - 16.0	12 - 22	111

Abbildung 4.11: Benötigte Kondensatorwerte für Quartzfrequenzen[11, S. 39]

Raspberry Pi kann alleine bis zu 3A verwenden, und weil man einen sechs Pin Stecker mit den Magnetventilstecker verwechseln könnte.

Lüfterstecker

Ein voll ausgelasteter Raspberry Pi 4 kann ohne Kühlung überhitzen[9], und deshalb ist auf dem SOC ein kleiner Kühlkörper geklebt der mit einem 5V 30mm Lüfter aktiv gekühlt wird. Der Lüfter wird mit einem zwei Pin Stecker mit Strom versorgt.

4.4.5 Kondensatoren für den Quartz

Mit dem ATmega328P hat man den Vorteil, dass man einen Arduino Nano als günstiges Developmentboard verwenden kann, und, dass man Teile der Arduino Nano Schaltung übernehmen kann. Der Arduino Nano verwendet einen 16MHz Quartz, der die notwendigen Kondensatoren im Gehäuse eingebaut hat[2], aber der Quartz, der auf dem PE-Bug verwendet werden soll, hat die Kondensatoren nicht eingebaut, und sie müssen deshalb extern hinzugefügt werden. Dem ATmega328P Datenblatt zufolge 4.11 braucht man für eine Clockfrequenz von 8MHz bis 16MHz Kondensatorwerte von 12pF bis 22pF[11, S. 39]. Gewählt wurden Kondensatoren mit einer Kapazität von 20pF.

4.4.6 Levelshifter

Die Kommunikation zwischen dem Raspberry Pi und dem ATmega328P Microcontroller wird über UART abgewickelt. Der Raspberry Pi verwendet UART bei einer Spannung von

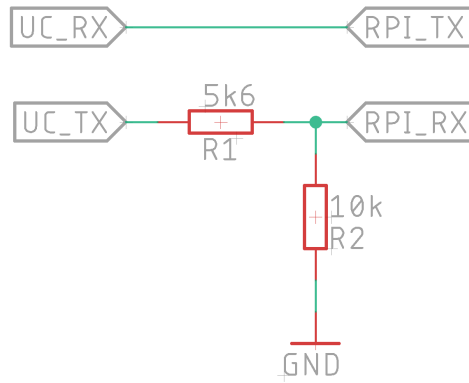


Abbildung 4.12: Spannungsteiler für die Serielle Kommunikation zwischen dem Microcontroller und dem Raspberry Pi

3,3V[16] und der ATmega328P wird beim PE-Bug bei 5V betrieben. Für den ATmega328P ist der Eingang ab einer Spannung von 0,7V HIGH[11, S. 317], also könnte der 3,3V Ausgang vom Raspberry Pi direkt in den Eingang vom Microcontroller geführt werden. In die andere Richtung ist das nicht möglich, weil die Raspberry Pi Eingänge nur 3,3V akzeptieren[16].

Die erste Idee war, die 5V vom Microcontroller mit einem Spannungsteiler 4.12 auf 3,3V zu reduzieren. Wenn man dem Widerstand R_2 den Wert $10k\Omega$ gibt, kann der Widerstand R_1 wie folgt berechnet werden:

$$\frac{5V}{3,3V} = \frac{R_1 + 10k\Omega}{10k\Omega}$$

$$R_1 = \left(\frac{5V}{3,3V} + 1\right) * 10k\Omega = 5,15k\Omega$$

Weil $5,15k\Omega$ nicht in der E12 Reihe vorhanden ist, ist es sinnvoll den nächsthöheren Wert zu verwenden, der $5,6k\Omega$ ist.

Nach einem Gespräch mit dem Diplomarbeitbetreuer wurde entschieden, die Lösung mit dem Spannungsteiler, durch einen Levelshifter IC zu ersetzen. Der gewählte IC ist ein Texas Instruments TXB0104 4.13 4-Bit Levelshifter, im SOIC-14 Gehäuse. Der $10k\Omega$ Widerstand am Pin OE, zieht den Output Enable Pin auf HIGH.

4.4.7 Spannungsanzeige LEDs

Für den Betrieb des PE-Bugs werden vier Spannungen benötigt, 3,3V, 5V, 12V und 24V. Die Spannungsversorgung ist ein 12V Bleiakku, und die anderen Spannungen werden über DC/DC Netzteile erzeugt. Die 5V, die der Microcontroller und der Raspberry Pi brauchen, und 24V für die Ultraschallsensoren werden von zwei externen Schaltnetzteilen erzeugt. Auf dem Raspberry Pi gibt es einen 5V zu 3,3V DC/DC Wandler, weil der Raspberry Pi selbst, und ein IC auf der Leiterplatte 3,3V benötigen.

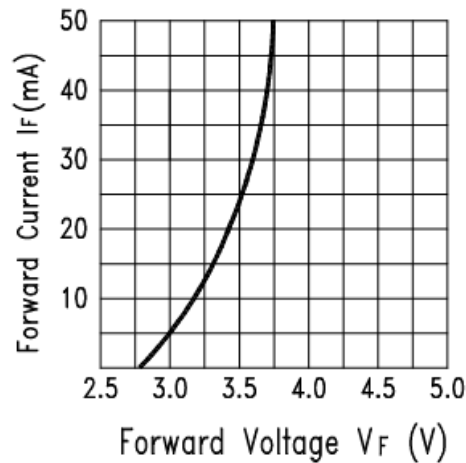


Fig.2 Forward Current vs.
Forward Voltage

Abbildung 4.14: Kennlinie der LED aus dem Datenblatt[8]

Wie 880Ω ist $2,08k\Omega$ auch kein Wert aus der E12 Reihe, weshalb ein Widerstand mit dem Wert $2,2k\Omega$ verwendet wird. Wie bei $U_S=12V$, muss der LED Vorwärtsstrom mit dem neuen Widerstandswert berechnet werden:

$$I_{F_{24V}} = \frac{24V - 3,2V}{2,2k\Omega} = 9,45mA$$

Weil es möglich ist, die LEDs mit einem Vorwärtsstrom I_F von bis zu 20mA zu betreiben, ist es bei den Widerstandswerten nicht so wichtig, dass die genauen Werte verwendet werden, solange vor dem Einlöten, noch einmal der Vorwärtsstrom berechnet wird.

4.4.8 Magnetventile schalten

Die Magnetventile des PE-Bugs schalten mit 12V, und müssen deshalb über Transistoren geschaltet werden. Die Lösung der alten Platine war es, THT Transistoren im TO-92 Gehäuse zu verwenden, aber das hat den Nachteil, dass dafür viel Platz auf der Platine benötigt wird. Auf der PE-Bug Platine werden deshalb SMD MOSFETs verwendet.

Um die richtigen MOSFETs zu wählen, hat der Strom, den die Magnetventile benötigen, gemessen 4.15 werden müssen. Es hat sich herausgestellt, dass beim Schalten, ein maximaler Strom von 50mA nicht überschritten wird. Das SI2300DS N-Kanal MOSFET von Vishay ist dazu geeignet, weil dieser einen Continuous Drain Current von 3,6A bei 25°C hat [21], und das SOT-23 Gehäuse ist relativ klein.

Strombegrenzungswiderstand

Um den maximalen Ausgangsstrom, der I/O Pins vom ATmega328P Microcontroller, der 40mA ist[11, S. 308], nicht zu überschreiten ist es sinnvoll, zwischen dem Microcontroller

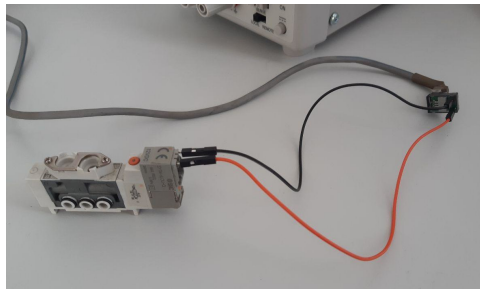


Abbildung 4.15: Messung des Stroms, den das Magnetventil benötigt

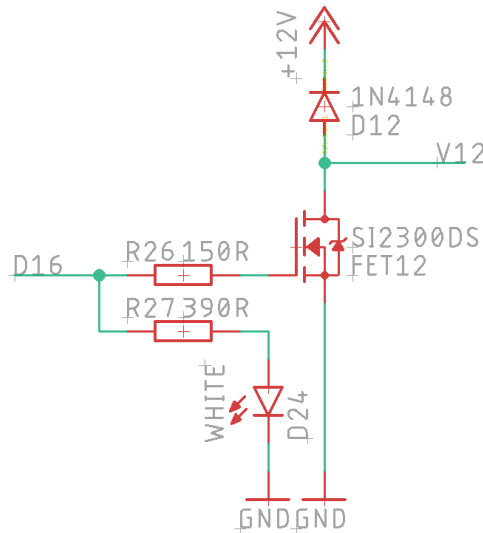


Abbildung 4.16: Schaltung mit der die Magnetventile gesteuert werden

Pin und dem Gate des MOSFETs einen Widerstand 4.16 einzubauen, der den Strom limitiert. Da an dem Widerstand maximal 5V abfallen kann, wird der Widerstandswert wie folgt berechnet:

$$R = \frac{5V}{I_{\max}} = \frac{5V}{40mA} = 125\Omega$$

Der nächsthöhere Widerstandswert in der E12 Reihe ist 150Ω also ist der maximale Strom der aus dem Mikrocontrollerpin fließt $33,3mA$.

$$I_{\max} = \frac{5V}{150\Omega} = 33,3mA$$

Debug LEDs

An jedem Ausgang, mit dem die Magnetventile gesteuert werden, ist eine LED angeschlossen, die beim Debuggen helfen kann. Um den maximalen Ausgangsstrom des Microcontrollers nicht zu überschreiten, darf der Vorwärtsstrom nicht größer als $40mA - 33,3mA = 6,6mA$ sein. In der Kennlinie 4.14 der LED sieht man, dass bei einem Vorwärtsstrom von 5mA, eine Vorwärtsspannung von 3V an der LED abfällt. Der Vorwiderstand der LED

wird wie folgt berechnet:

$$R_V = \frac{5V - 3V}{5mA} = 400\Omega$$

Weil es 400Ω nicht in der E12 Reihe gibt, wird ein 390Ω Widerstand verwendet bei dem ein Vorwärtsstrom von $5,13mA$ fließt, was unter dem Maximum von $6,6mA$ ist.

$$I_F = \frac{2V}{390\Omega} = 5,13mA$$

In dem Fall ist es auch möglich einen 330Ω Widerstand einzulöten:

$$I_F = \frac{2V}{330\Omega} = 6,06mA$$

Es war notwendig auszurechnen, ob ein 330Ω Widerstand möglich ist, weil zum Zeitpunkt des Lötens keine 390Ω Widerstände vorrätig waren.

Freilaufdiode

Weil die Magnetventile eine Induktive Last darstellen, braucht man Freilaufdioden, damit auf dem Drain Pin der MOSFETs keine hohe Spannung anliegt, die die Transistoren zerstören kann. Die Freilaufdiode muss in der Lage sein, den maximalen Strom der Magnetventile, der wie schon erwähnt $50mA$ ist, in Vorwärtsrichtung durchzulassen. Die 1N4148W Diode von Diodes Incorporated hat einen Forward Continuous Current von $300mA$ [3], und ist somit für die Anwendung geeignet.

Layout auf der Leiterplatte

Für jedes Magnetventil gibt es eine Schaltung wie in Abbildung 4.16 gezeigt. Die Bauteile sind auf der Leiterplatte in einem Rechteck angeordnet 4.17, und die zwölf Gruppen an Bauteilen sind platzsparend nebeneinander angeordnet. Die Freilaufdioden sind auf der Rückseite der Leiterplatte platziert.

4.4.9 Produktion der Leiterplatte

In der HTL Bulme gibt es die Möglichkeit, Leiterplatten herzustellen. Das wird durch Ätzen, oder Fräsen, der Kupferschicht gemacht. Jedoch war die Herstellung in der Schule für die PE-Bug Leiterplatte nicht möglich, weil beim Routing, zum Beispiel Vias unter Bauteilen angebracht wurden, oder THT Komponenten, wurden von der top und von der bottom Seite, mit Leiterbahnen verbunden. Es hat auch keinen Grund gegeben, das Leiterplattenlayout so zu verändern, dass es möglich wäre, es in der Schule herzustellen, weil

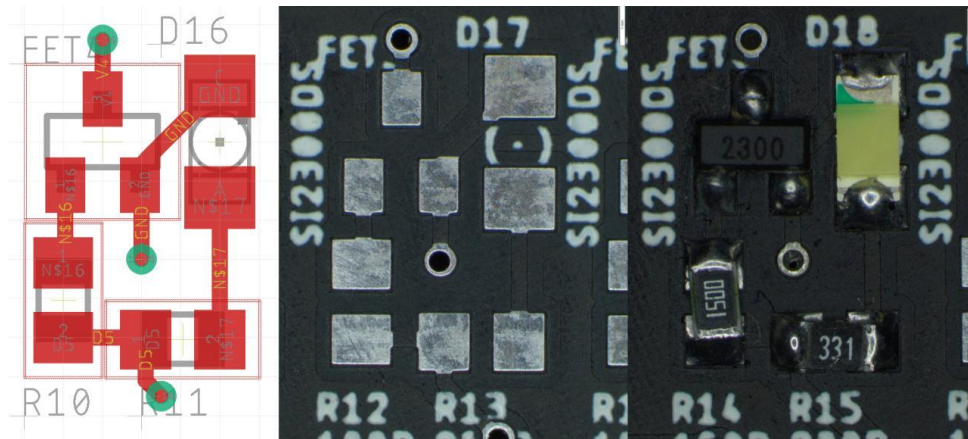


Abbildung 4.17: Anordnung der MOSFETs auf der Leiterplatte

im Wintersemester 2020/21 die Covid-Lockdowns in Kraft waren, und die Produktion in der Schule nur schwerer möglich gewesen wäre. Neben den Gründen, und dem Grund, dass die Qualität, der in der Schule hergestellten Leiterplatten schlechter ist, wurde entschieden, bei der Leiterplattenfirma JLCPCB zu bestellen. Wenn man bei JLCPCB bestellt, ist es sinnvoll die Leiterplatte unter 100mm*100mm zu designen, weil Leiterplatte die größer als das sind, um einiges teurer werden.

4.4.10 Liste der Bauteile

Die PE-Bug Leiterplatte verwendet die Bauteile, die in der Tabelle 4.1 auf der Seite 35 angeführt sind.

4.5 PE-Bug Leiterplatten Prototyp

Um auch schon vor der fertigen Leiterplatte, die Microcontroller Firmware testen zu können, wurde ein Prototyp 4.18 der Leiterplatte, auf einer Lochrasterplatine gelötet. Damit die Bauteile mit der Lochrasterplatine verwendet werden können, sind beim Prototyp nicht die gleichen Bauteile, wie in der fertigen Platine verlötet. Die SI2300DS MOSFETs sind durch IRLZ34N Transistoren ersetzt, und anstatt der 1N4148W Dioden kommen 1N4007 Dioden zum Einsatz. Es muss aber der gleiche Microcontroller, im gleichen TQFP SMD Gehäuse verwendet werden, weil die 28 SPDIP THT Version weniger ADC Eingänge hat[11, S. 14]. Um den Microcontroller trotzdem auf die Lochrasterplatine löten zu können, wird ein Arduino Nano eingesetzt.

Tabelle 4.1: Bauteile auf der Leiterplatte

Wert	Bauteil	Name	Package
100nF	Kondensator	C1-C5, C9	C0805
10 μ F	Elektrolytkondensator	C6	E5-4
20pF	Kondensator	C7, C8	C0805
1N4148W	Diode	D1-D12	SOD123FL
Weiß	LED	D13-D28	1206
SI2300DS	N-Kanal MOSFET	SI2300DS	SOT23
2 Pin	Molex KK 254	J1	6410-02
4 Pin	Molex KK 254	"J2-J5, J11"	6410-04
6 Pin	Molex KK 254	J7-J10	6410-06
8 Pin	Molex KK 254	J6	6410-08
2x20	Stiftleiste	JP1	2x20
2x3	Stiftleiste	JP2	2x3
150 Ω	Widerstand	R1, R6, R8, R10, R12, R14, R16, R18, R20, R22, R24, R26	R0805
180 Ω	Widerstand	R29	R0805
390 Ω	Widerstand	R5, R7, R9, R11, R13, R15, R17, R19, R21, R23, R25, R27	R0805
820 Ω	Widerstand	R28	R0805
1k Ω	Widerstand	R4	R0805
1,8k Ω	Widerstand	R38	R0805
10k Ω	Widerstand	R2, R3, R30-R37	R0805
S1	Taster	SMD_BUTTON	
ATmega328P-AU	Mikrocontroller	U1	TQFP-32
TXB0104	Levelshifter	U2	SOIC14
16MHz	KX-KS Quartz	X1	11.4X4.5
30mm Lüfter	30mm Lüfter	FAN	30mm
2,8mm	2,8mm Loch	H1-H4	
3,2mm	3,2mm Loch	H5-H10	

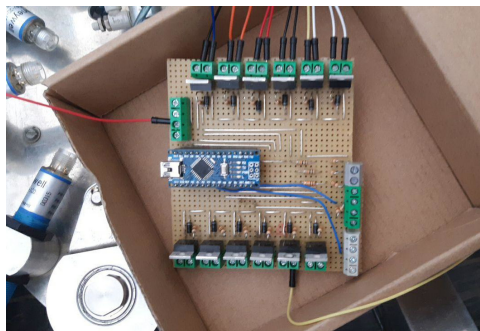


Abbildung 4.18: Leiterplattenprototyp auf einer Lochrasterplatine

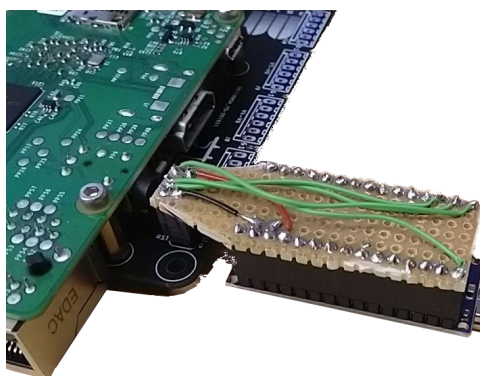


Abbildung 4.19: ICSP Programmierer mit einem Arduino Nano

4.6 ICSP Programmierer

Um auf einen Arduino eine neue Firmware hochzuladen, gibt es zwei Optionen. Die Erste ist, den Arduino Bootloader auf den Microcontroller hochladen, was bedeutet, dass der Arduino über die Serielle Schnittstelle programmiert werden kann. Diese Möglichkeit wird auf den Arduino Entwicklungsboards verwendet, und dafür wird ein USB zu UART IC eingesetzt. Die Zweite Möglichkeit ist es, das Programm mit der ICSP (In Circuit Serial Programmer) Schnittstelle hochzuladen, wofür ein ISP (In System Programmer) verwendet wird. Weil ein Programmer dafür circa 20€ kostet, ist es sinnvoll, eine günstigere Alternative zu suchen. So eine Alternative ist ein Arduino mit dem Arduino as ISP[1] Programm, mit dem ein Arduino, wie zum Beispiel ein Arduino Nano, über die ICSP Schnittstelle, Firmware auf einen anderen Microcontroller hochladen kann. Weil der Aufbau einer Arduino as ISP Installation auf einem Steckbrett zeitintensiv ist, wurde entschieden, einen Arduino Nano auf eine Lochrasterplatine 4.19 zu löten, und die benötigten Leitungen auf der Lochrasterplatine zu führen.

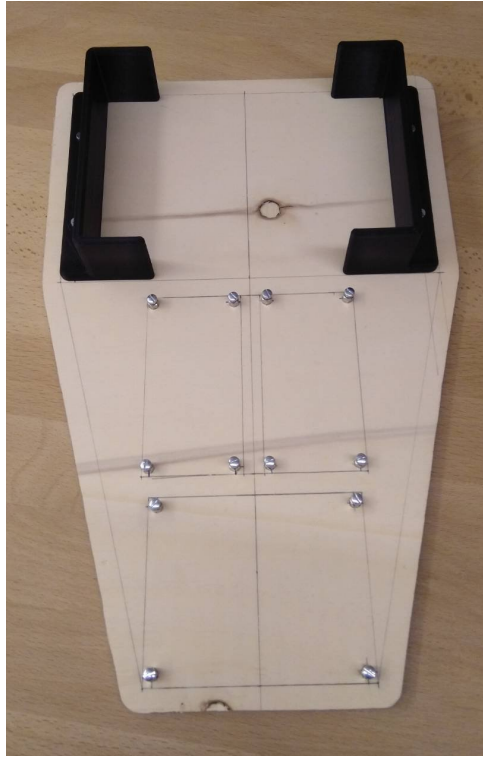


Abbildung 4.20: Noch nicht gefärbte Holzplatte mit dem 3D gedruckten Akkuhalter

4.7 Mechanische Teile

4.7.1 Befestigungsplatte

Der Bleiakku, die DC/DC Netzteile und die Hauptplatine sind auf einer 3mm dicken Holzplatte 4.20 befestigt. Die Holzplatte wurde mit einer Laubsäge ausgeschnitten, die Kanten wurden gefeilt, und die Löcher sind mit einem 3mm Bohrer gebohrt worden.

Um dem Projekt mehr Stil zu verleihen, und es moderner wirken zu lassen, wurde die Platte mattschwarz mit Sprühfarbe 4.21 eingefärbt. Die Platte ist mit zwei M4 Schrauben dort befestigt, wo die alte Leiterplatte angeschraubt war.

4.7.2 Lüfterbefestigung

Damit man sicherstellen kann, dass der Raspberry Pi ausreichend gekühlt ist, ist auf der Leiterplatte des PE-Bug, ein 30mm Lüfter angebracht, der Luft unter den Raspberry Pi, und über einen Kühlkörper, bläst. Um den Lüfter am Board zu befestigen, braucht es ein kleines 3D gedrucktes Kunststoffteil 4.22, das mit M3 Schrauben, auf der Leiterplatte festgeschraubt wird.

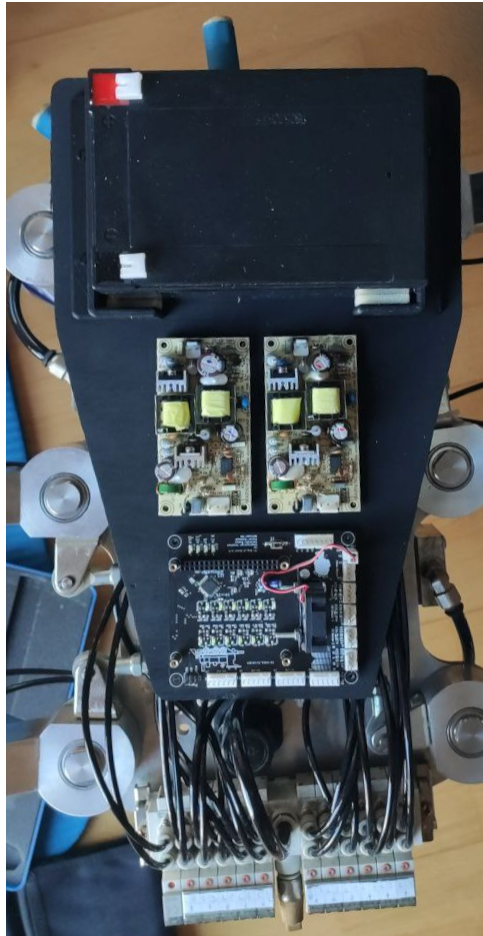


Abbildung 4.21: Die gefärbte, und am PE-Bug angeschraubte, Befestigungsplatte mit dem Akku, den DC/DC Netzteilen und der Leiterplatte montiert

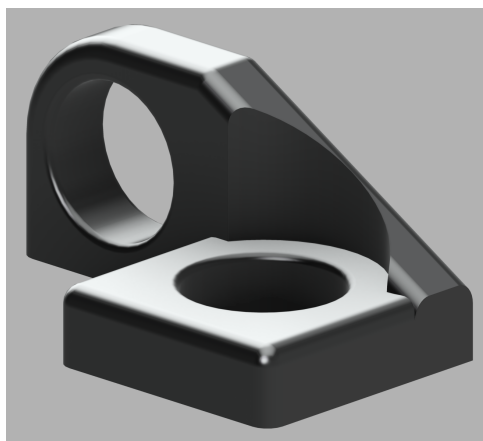


Abbildung 4.22: 3D gedrucktes Kunststoffteil, um den Lüfter auf dem Board zu befestigen

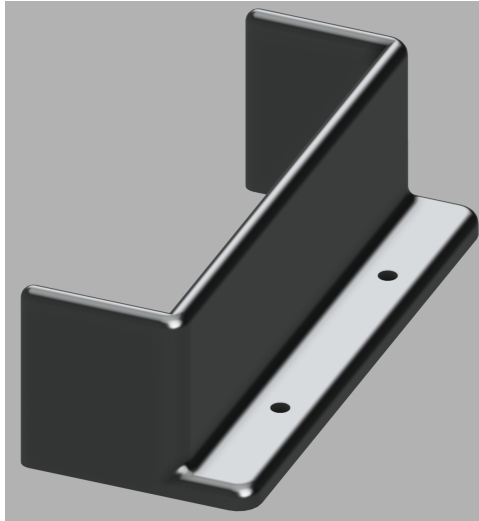


Abbildung 4.23: 3D Modell des Akkuhalters für den PE-Bug

4.7.3 Akkubefestigung

Damit sich der Bleiakku nicht zu sehr bewegt, wurden für den PE-Bug Akkuhalterungen 4.23 3D gedruckt. Die werden mit M3 Schrauben auf der Platte festgeschraubt, auf der auch die Leiterplatte und die DC/DC Netzteile befestigt sind.

4.8 Bewegung

Ein wichtiger Teil der Firmware ist der Bewegungsablauf. Er ist das zentrale Element der μ C-Software und hatte bei der Entwicklung oberste Priorität. Um eine möglichst gute Stabilität zu gewährleisten wurde sich an der Natur orientiert. Da der PE-Bug über sechs Beine verfügt wurde eine Abhandlung über Insekten, von der Webside www.sciencedirect.com[19], herangezogen.

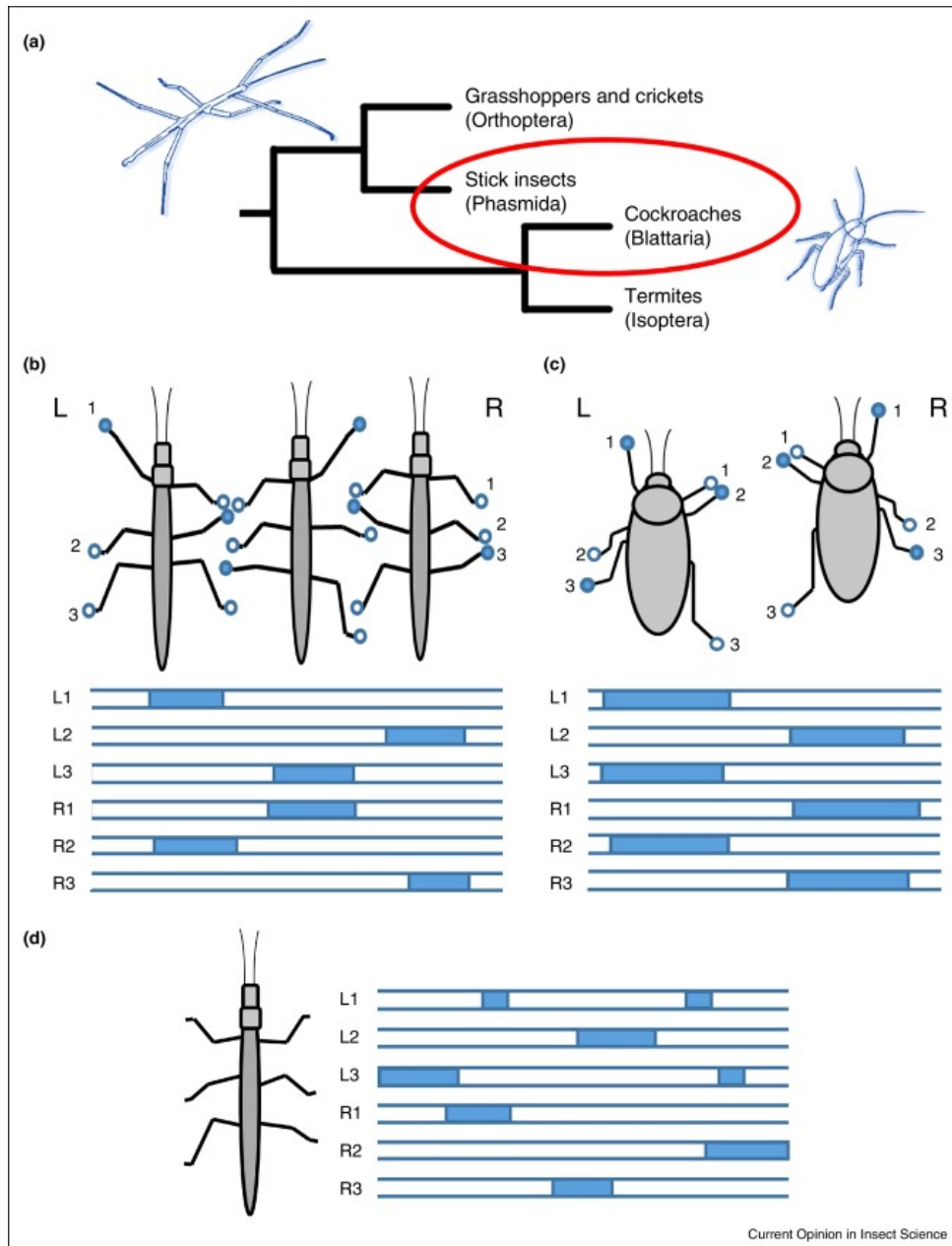


Abbildung 4.24: Insekten Bewegung

Wie in Abbildung 4.24 zu sehen ist, werden darin drei verschiedene Bewegungsabläufe dargestellt.

- Durch die mechanische Beschränkung des PE-Bugs ist der Bewegungsablauf (d) eher ungeeignet. Grund dafür ist die Varianz der Bewegungsdauer, welche die Wartezeiten bei der Bedienung hervorruft und somit den PE-Bug verlangsamen würde. Diese langsamere Vorwärtsbewegung hätte jedoch den Vorteil wesentlich stabiler zu sein.
- Der Bewegungsablauf (b) bietet einen Mittelweg zwischen Stabilität und Mobilität. Dieser lässt sich in drei Abschnitte aufteilen, die jeweils eine gleiche Ablaufdauer aufweisen. Das ermöglicht eine schnellere Fortbewegung, da man nicht auf mögliche Zwischenschritte warten muss.
- Der Bewegungsablauf (c) bietet den einfachsten sowie schnellsten Ablauf und eignet sich somit am Besten für den PE-Bug und seine mechanischen Anforderungen. Die Einteilung in lediglich zwei Blöcke minimiert die Zeit, die benötigt wird um einen Bewegungsblock (z.B. heben, bewegen, senken) auszuführen, da man statt auf zwei (Bewegungsablauf b) nur auf maximal einen Block warten muss.

Tabelle 4.2: Vor- und Nachteile der verschiedenen Bewegungsabläufe

Bewegungsablauf	Pros	Cons
(b)	Stabilität, Mobilität	Wartezeit
(c)	Mobilität, Wartezeit	Stabilität
(d)	Stabilität	Wartezeit, Mobilität

Aufgrund der oben angeführten Vor- und Nachteile, wurde sich für den Bewegungsablauf (d) entschieden.

4.9 Software

Die Software kann in zwei größere Gebiete unterteilt werden: Die Firmware, welche auf dem ATmega328P läuft, und der Anwendungssoftware, die auf dem Raspberry Pi läuft. Die Kommunikation der beiden erfolgt über UART, auch bekannt als serielle Schnittstelle. Da UART keinerlei Fluss-/Transaktionskontrolle besitzt, muss etwas Vergleichbares von uns implementiert werden. 4.25 ist ein Beispiel wie ein solches System funktionieren könnte.

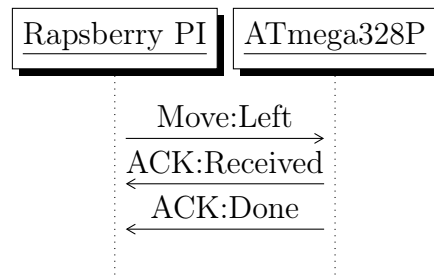


Abbildung 4.25: Beispiel Transaktion

4.9.1 Aufgabenteilung

Da es zwei Prozessoren gibt, ist die Einteilung der Aufgaben wichtig, um redundante Systeme zu vermeiden.

ATmega328P

Der ATmega328P ist für den hardwarenahen Bereich zuständig. Er nimmt Befehle vom Raspberry Pi an und führt diese aus. Auch das Rückmelden von Sensorwerten gehört zu diesem Aufgabenbereich.

Raspberry Pi

Der Raspberry Pi ist für die Steuerung des PE-Bugs zuständig. Weil der Pi um ein Vielfaches mehr Leistung als der ATmega328P besitzt, ist er besser geeignet, um Daten zu verarbeiten. Auch die Konnektivitätsmöglichkeiten sind auf dem Pi besser, deshalb wird dieser auch die externen Verbindungen behandeln.

4.10 Protokoll

Das Protokoll zwischen den beiden Prozessoren ist ein zentraler Baustein der Software und muss klar definiert sein.

4.10.1 Aufbau

Jeder Befehl ist gleich aufgebaut. Er besteht aus einer Gruppe, dem tatsächlichen Befehl und einem optionalen Parameter, sowie einem `\n` um das Ende zu kennzeichnen.

Wichtig: Befehle sind casesensitive!

Gruppe:Befehl:Parameter\n

Abbildung 4.26: Aufbau eines Befehls

4.10.2 Gruppen

Conf

Die *Conf* Gruppe ist für die Konfiguration notwendig.

- `Conf:Delay:{int}` legt das Intervall (in ms) für die Bewegung der Beine fest.
- `Conf:SensorTimeout:{int}` konfiguriert den Zeitabstand zwischen Senosraussendungen. (in ms)
- `Conf:SensorTimeout:{On|Off|Auto}` Stellt die Kompressorsteuerung ein.

Move

Die *Move* Gruppe ist für die Fortbewegung des PE-Bugs zuständig. Die verfügbaren Befehle sind:

- Right
- Forward
- Back
- Left

Turn

Die *Turn* Gruppe ist für Drehbewegungen zuständig. Die verfügbaren Befehle sind:

- Right
- Left

Sensor

Die Sensorwerte werden, sofern konfiguriert, periodisch ausgegeben. Man kann die Werte aber auch manuell mit dem Befehl `Sensor:Distance:Get`.

Die Sensorwerte werden im Format:

`Return:Distance:{Sensor Left};{Sensor Middle};{Sensor Right}`

4.10.3 Quittierung

Alle Befehle müssen vom Mikrokontroller zweifach quittiert werden. Nach dem Empfang des Befehls muss ein *ACK:Received* gesendet werden und nach der vollständigen Ausführung muss ein *ACK:Done* gesendet werden.

4.11 Firmware

Während der Entwicklung der Software für den Arduino Nano wurden zahlreiche Veränderungen und Verbesserungen vorgenommen. Die Entwicklung lässt sich jedoch in zwei Teile unterteilen.

In eine erste Version, die nur den Standard Modus [4.11.1] beinhaltet, und eine zweite Version, die neben dem Standard Modus auch einen manuellen Modus [4.11.2] besitzt, in welchem man jeden Kolben manuell ansteuern kann.

4.11.1 Standard Modus

Bewegungsablauf

Im Standard Modus ist es möglich über die serielle Schnittstelle Bewegungsbefehle, hier instructions [4.11.3], auszuführen. Jeder dieser Bewegungsbefehle beinhaltet alle Informationen die benötigt werden um eine Bewegung, beispielsweise "Vorwärts", auszuführen.

```
ACK:Receivedvi
r_11_e -> 1vi
r_12_e -> 1vi
l_12_e -> 1vi
l_11_e -> 1vi
l_12_e -> 1vi
l_13_e -> 1vi
r_11_e -> 1vi
r_12_e -> 1vi
r_13_e -> 1vi
r_11_e -> 0vi
r_12_e -> 0vi
l_12_e -> 0vi
ACK:Donevi
```

Abbildung 4.27: Ausgabe mit Testdaten

Dabei können die Kolben jedoch nicht gleichzeitig angesprochen werden, sondern nur einer nach dem Anderen. Problematisch an der Fortbewegung ist, dass es keine Abfrage zur aktuellen Position der Kolben gibt. Es ist zwar möglich den logischen Zustand des Pins abzufragen, doch der logische Zustand ändert sich sofort. Wo hingegen das mechanische aus- und einfahren der Kolben Zeit braucht. Um also eine reibungslose Fortbewegung zu ermöglichen ist es notwendig auf die Ausführung von Befehlsblöcken zu warten. Die Zeit welche zwischen jedem Befehlsblock gewartet wird, ist über die serielle Schnittstelle mit einen Multiplikator einstellbar.

Als Befehlsblock wird hier eine Aneinanderreihung von gleichen Befehlen bezeichnet. Zum Beispiel drei aufeinander folgende Befehle von denen jeder ein anderes Bein anhebt.

Sensordatenausgabe

Ein weiteres Feature des Standard Modus ist die automatische Ausgabe der Sensordaten. Diese werden über die serielle Schnittstelle an den Master gesendet. Das Intervall, in der die Daten versendet werden, kann am Master eingestellt werden. Diese Funktion kann auch deaktiviert werden indem man das Intervall '0' setzt.

```
Return:Distance:333;330;326_vn  
Return:Distance:341;337;333_vn  
Return:Distance:323;319;317_vn
```

Abbildung 4.28: Automatische Sensordatenausgabe

Neben der automatischen Ausgabe ist es aber auch möglich Sensordaten manuell anzufordern.

Standby Funktion

Der PE-Bug verfügt zudem über eine Standby Funktion. Wird über zulange Zeit kein Bewegungsbefehl ausgeführt, schalten sich die Kompressoren ab. Diese werden wieder automatisch eingeschalten sobald ein neuer Bewegungsbefehl kommt. Ist diese Funktion unerwünscht kann diese abgeschaltet werden, indem man die Zeit bis zum Abschalten auf '0' setzt.

4.11.2 Manueller Modus

Anders als im Standard Modus [4.11.1], gibt es im manuellen Modus nur einen Befehl, in dem alle Informationen enthalten sind um den PE-Bug zu bedienen. Weiters wird im Manuellen Modus nichts automatisch ausgeführt. Eingaben vom Benutzer werden vom Controller nicht bestätigt und auch Sensordaten werden nur auf Abfrage gesendet. Sollten in kurzer Zeit zu viele Befehle versendet werden, so dass der Controller diese nicht schnell genug abarbeiten kann, werden diese verworfen.

Bewegungsablauf

Um eine Bewegung auszuführen ist es notwendig ein spezielles Bein sowie den richtigen Kolben und die gewünschte Richtung über die serielle Schnittstelle an den μC zu senden. Es ist nicht möglich mehrere Bewegungsbefehle gleichzeitig in einem Feld (array) zu senden.

Sensordatenausgabe

Um eine Sensorausgabe zu erzeugen ist es notwendig einen Bewegungsbefehl an den μC zu senden. Innerhalb dieser Instruktion muss dann das Bit "getsensor" gesetzt sein. Um zu verhindern das sich ein Bein bei einer Sensorausgabe bewegt, muss ein Kolben mit dem gleichen Zustand angesprochen werden, den er gerade besitzt. Diese Lösung wurde deshalb gewählt, weil es dadurch möglich ist Sensordaten während einer Bewegung anzufordern.

Standby

Wie auch bei der Sensorausgabe funktioniert die Standbyfunktion im manuellen Modus nicht automatisch. Es ist hier ebenfalls erforderlich einen Bewegungsbefehl an den μC zu senden. Darin ist es möglich beide Kompressoren über das compressor-Bit ein- und auszuschalten. Da der PE-Bug über keinen Luft-Tank verfügt wird es empfohlen beide Kompressoren während der Bewegung eingeschalten zu haben.

Der genaue Aufbau des Bewegungsbefehls wird im Kapitel 4.11.3 Instruktionen unter Aufbau der neuen Instruktion beschrieben.

4.11.3 Instruktionen

Eine Instruktion ist ein Feld (array), das aus einem oder mehreren Befehlen besteht und beinhaltet alle Informationen die Benötigt werden um eine Bewegung erfolgreich auszuführen. Somit besitzt jede Bewegung eine eigene Instruktion die vom Master angefordert werden kann. Die Anzahl der Befehle ist von Bewegung zu Bewegung unterschiedlich und hat in der Theorie keine Grenzen. Es ist allerdings nicht vorgesehen Bewegungsbefehle in der Firmware zu verändern. Ist eine spezielle Abfolge von Befehlen gewünscht, muss auf den Manuellen Modus [4.11.2] gewechselt werden.

Aufbau der alten Instruktionen

Anfangs wurde jede Instruktion in ein "struct" geschrieben wie es in Abbildung 4.29 und Abbildung 4.30 zu sehen ist. Diese Darstellung ermöglicht eine hohe Lesbarkeit, hat jedoch den Nachteil eines sehr hohen Speicherverbrauchs.

$$3 \cdot \text{int} + 1 \cdot \text{bool} + 1 \cdot \text{unsignedlong} = 162 \text{Bit}$$

```
1  {right, front, elevation, true, 0},  
2  {right, back, elevation, true, 0},  
3  {left, middle, elevation, true, 0},
```

Abbildung 4.29: Alte Instruktion

```
1  struct instruction  
2  {  
3      int side;  
4      int posistion;  
5      int piston;  
6      bool direction;  
7      unsigned long timeout;  
8  };
```

Abbildung 4.30: Aufbau der alten Instruktion

Änderungen durch die neuen Instruktionen

Durch die Neuauflage der Instruktionen der 2. Version der PE-Bug Firmware wurde der Nachteil des Speicherverbrauchs behoben. Neben dem geringeren Speicherverbrauch bietet diese Lösung eine einfachere Einbindung des manuellen Modus, da es nun möglich ist über eine immer gleiche Instruktion jede Funktion des PE-Bugs anzusprechen. Um den Speicher zu entlasten wurde das "struct" als Speicher verworfen und wich einer unsigned short Variable wie sie in Abbildung 4.31 zu sehen ist.

Durch diese Darstellung geht jedoch die Übersichtlichkeit der alten Instruktion verloren. Um den Code trotzdem einfach lesbar zu machen und somit das Arbeiten zu erleichtern wurde der Aufbau des zuvor benutzten structes teilweise als temporäres struct (Kommando struct) beibehalten. Die Verwendung des Kommando structes wird im Kapitel Service Routine [4.11.5] genauer beschrieben.

1	0b1100010100000000 ,
2	0b1110010100000000 ,
3	0b1001010100000001 ,

Abbildung 4.31: Neue Instruktion

Aufbau der neuen Instruktion

Das unsigned short der neuen Instruktion teilt sich wie in Abbildung 4.32 und Abbildung 4.3 zu sehen ist in: 1 Fortsetzungs-Bit, 4 Bits die den jeweiligen Kolben beschreiben, je 1 Bit für Kompressoren und Sensoren sowie 8 Bits für die Verzögerungszeit.

0	1	2	3	4	5	6	7	8	15
CO	SI	PO	PI	DI	GS	CO	TI		

Abbildung 4.32: Bit Tabelle

Tabelle 4.3: Beschreibung der Befehle

Kürzel	Befehl	Deutsch
CO	continue	Fortfahren
SI	side	Seite
PO	position	Position
PI	piston	Kolben
DI	direction	Richtung
GS	getsensor	Sensor
CO	compressor	Kompressor
TI	delaytime	Verzögerungszeit

Umstieg zur neuen Instruktion

Um zu verhindern, dass beim Umschreiben der Instruktionen Fehler entstehen und den Prozess zu beschleunigen, wurde dazu ein eigenes Programm entwickelt. Das Programm liest Instruktionen des alten Formates ein und formatiert diese in das neue Format so um, dass die Ausgabe nur noch in die Firmware des PE-Bugs kopiert werden muss.

Dazu wird jeder Befehl der Instruktion nacheinander eingelesen. Wie in Abbildung 4.33 zu sehen ist, wird dann jede Variable eines Befehl an den vorgesehenen Platz im unsigned short geschoben.

```
1 unsigned short out = 1;
2 out = out << 15;
3 out += i.side << 14;
4 out += i.posistion << 12;
5 out += i.piston << 11;
6 out += i.direction << 10;
7 out += 0 << 9; // sensor request
8 out += 1 << 8; // motor
9 out += i.timeout;
10 return out;
```

Abbildung 4.33: Umwandlung der Instruktion

Da der Inhalt des insigned shorts jedoch als Dezimalzahl ausgegeben wird, muss dieser bevor er ausgegeben wird noch in Binär umgewandelt werden. Dies ist zu sehen in Abbildung 4.34. Dazu wird der Output dieser Funktion Bit für Bit mittels einer Bitmaske auf 1 oder 0 überprüft. Das Ergebnis wird anschließend in der Konsole ausgegeben.

```
1 for (int i = 0; i < 16; i++) {
2     printf("%d", (num & mask ? 1:0));
3     mask = mask >> 1;
4 }
```

Abbildung 4.34: Ausgabe einer Dezimalzahl in Binär

4.11.4 Einlesen über die serielle Schnittstelle

In diesem Kapitel wird der Weg der Information von der Seriellen Schnittstelle bis zur Service Routine [4.11.5] beschrieben.

Dieser ist im Falle des Standard Modus [4.11.1] wesentlich komplexer als beim manuellen Modus [4.11.2]. Im Standard Modus werden bis zu 64 Bits beziehungsweise bis zu einem Zeilenumbruch, Daten aus der seriellen Schnittstelle ausgelesen und in ein Feld geschrieben. Das Feld wird anschließend dem Übertragungsprotokoll [4.10] entsprechen an den ':' in bis zu drei kleinere Felder geteilt. Nach der Teilung des Feldes folgt eine Aneinanderreihung von Abfragen:

- Ist der Befehl eine Anforderung zur Änderung der Einstellungen, wird die Eingabe auf Zahlen geprüft. Sollten sich in der Eingabe Buchstaben oder andere Sonderzeichen befinden, so wird der Befehl verworfen.
- Ist der Befehl ein Bewegungsbefehl [4.11.3], wird die entsprechende Instruktion in den Buffer geschrieben, der von der Service Routine [4.11.5] weiterverwendet wird.
- Ist der Befehl eine Sensoranfrage, werden alle Sensordaten gemäß dem Übertragungsprotokoll über die serielle Schnittstelle übertragen

Sollte einer dieser Fälle eintreffen wird ein Acknowledge an den Master gesendet.

Achtung! Es folgt keine Fehlermeldung, wenn der Befehl nicht existiert oder verworfen wird.

Befindet sich der Controller im manuellen Modus, wird die empfangene Instruktion direkt in den Buffer geschrieben und ausgeführt. Es folgt kein Acknowledge.

4.11.5 Service Routine

In der Service Routine wird der momentan auszuführende Befehl der Instruktion [4.11.3] in ein Kommando struct geschrieben. Das Kommando struct teilt alle Informationen des Befehls, der aus einem unsigned short besteht, in einzelne Variablen auf um den Befehl einfacher verarbeiten zu können.

```

1  struct command
2  {
3      bool resume;
4      bool side;
5      short posistion;
6      bool piston;
7      bool direction;
8      bool getsensor;
9      bool compressor;
10     unsigned long timeout;
11 };

```

Abbildung 4.35: Neue Instruktion

Als erstes erfolgt die Abfrage des Fortsetzungs-Bit:

- Ist das Fortsetzungs-Bit '0' wird der Befehl nicht ausgeführt und der Controller wechselt in den Standard Modus.
- Ist das Fortsetzungs-Bit '1' wird der Befehl ausgeführt. Zudem bleibt der Controller im gleichen Modus.

```
1 | check = Movement(current_inst);  
2 | if (check == true) {instIndex++;}  
3 | if (current_inst.getsensor == 1) {ReadSensorDistance();}  
4 | digitalWrite(c, current_inst.compressor);  
5 | check = 1;
```

Abbildung 4.36: Fortsetzungs-Bit == 1

Wie in der oberen Abbildung 4.36 zu sehen ist, wird erst zum nächsten Befehl weiter geschaltet, wenn der vorhergehende bestätigt wurde. Bestätigt wird ein Befehl erst dann, wenn die in der Instruktion vorgesehene Zeit abgelaufen ist. Die Abfrage erfolgt über den Vergleich der vergangenen Millisekunden wie in Abbildung 4.37 zu sehen ist. Durch die Verwendung der millis() Funktion anstatt der delay() Funktion, ist es gewährleistet das nicht Bewegungsbefehle sowie die automatische Sensorausgabe weiterhin funktionieren.

```
1 | if (timeout * Delay + time <= millis())
```

Abbildung 4.37: Abfrage der Millisekunden

Die Verwendung von millis bringt jedoch auch einen Nachteil. Grund dafür ist der Rückgabewert, der nach ungefähr 49 Tagen overflowed. Um dies zu verhindern ist es erforderlich den μ C regelmäßig neu zu starten.

Wird eine Instruktion erfolgreich ausgeführt, wird ein Acknowledge an den Master gesendet, sofern sich der Controller im Standard Modus befindet.

4.12 Raspberry PI

Die Optionen für das Betriebssystem sind sehr limitiert. Es gibt nur eine offiziell unterstützte Version, RaspberryPi OS auch bekannt unter dem früheren Namen Rasbian. Dieses ist ein Derivat der GNU+Linux Distribution Debian[15]. Es gibt innerhalb dieses Betriebssystems drei Varianten, von diesen ist nur die Lite Variante von Relevanz. Die erweiterten grafischen Oberflächen der zwei anderen Möglichkeiten sind für unsere Anwendung nicht notwendig[14].

4.12.1 SSH

SSH ist ein Netzwerkprotokoll mit dem man sicheren Fernzugriff gewährleisten kann[17]. Es wird mit dem Betriebssystem ein SSH-Server bereits mitgeliefert, deshalb setzen wir SSH ein. Der Verbindungsaufbau kann mit einem SSH-Anwendungsprogramm wie PuTTY oder OpenSSH erfolgen. Mit diesen Programmen bekommt man mit den richtigen Login-Informationen eine Text-Konsole für die vollständige Bedienung des Systems.

4.12.2 Systemd

Systemd ist der auf dem RaspberryPi verwendete System Manager. Systemd ist dafür verantwortlich, dass alle Services gestartet sind und auch ausgeführt werden. Um den Backend-Server automatisch zu starten, wird dieser als Service hinterlegt, dazu muss man ein so genanntes Unit-File schreiben.

```
1 [Unit]
2 Description=Autostart the backend server
3 After=network.target
4
5 [Service]
6 ExecStart=/home/pi/backend/main /dev/ttyS0
7
8 [Install]
9 WantedBy=multi-user.target
```

Abbildung 4.38: Unit-File für den Backend-Server [pebugbackend.service]

Der Service kann manuell mit *systemctl start pebugbackend* auf der Komandozeile starten, *start* kann um den Service wieder zu beenden durch ein *stop* ersetzt werden. Um den Service in den Autostart hinzuzufügen, muss man *systemctl enable pebugbackend*

```
1 || serialPortPath := os.Args[1]
```

Abbildung 4.40: CLI-Argument Implementierung

ausführen. Hier kann man *enable* durch *disable* ersetzen, um den Service wieder aus dem Autostart zu löschen.

4.13 Golang Backend

Um die Bedienung auch für Zugriffe von außen freizugeben, wird ein Backend-Service benötigt, der die Befehle auf der seriellen Konsole in eine andere Form von Befehlen umwandelt. Für diese andere Form wurde HTTP ausgewählt, da es die Entwicklung einer Web-Oberfläche ermöglicht und in den meisten



Abbildung 4.39: Golang Logo

populären Programmiersprachen gut integriert ist. Als Sprache für das Backend wurde Golang gewählt. Die Sprache bietet eine mächtige Standard-Bibliothek und eine einfache Möglichkeit Nebenläufigkeit zu implementieren. Letztere vereinfacht das Empfangen von Nachrichten auf der seriellen Schnittstelle oder das Behandeln von HTTP-Anfragen.

4.13.1 Serielle Schnittstelle

Die Implementierung der seriellen Schnittstelle teilt sich in zwei Routinen sowie eine Initialisierung auf. Eine Routine empfängt und verarbeitet eingehende Daten. Um das Senden für Nebenläufigkeit abzusichern, muss auch dies mit einer eigenen Routine stattfinden.

Initialisierung

Um die serielle Schnittstelle auswählbar zu machen, wird ein Kommandozeilenargument benutzt. Das Einlesen dieses Argumentes wird in 4.40 implementiert.

Die Baudrate für dieses Projekt wurde auf Seite des ATmega328P sowie der des RaspberryPi auf 9600 festgelegt. Aus diesem Grund ist diese auch im Code fixiert.

In 4.42 ist die Code-Passage, die einen potentiellen Fehler beim Öffnen der seriellen Schnittstelle fängt. Wenn hier ein Fehler auftritt, wird zuerst eine Fehlermeldung ausgegeben und anschließend das Programm kontrolliert beendet.

```

1 m := serial.Mode{
2     BaudRate: 9600,
3 }
4 serialPort, err := serial.Open(serialPortPath, &m)

```

Abbildung 4.41: Initialisierung der seriellen Schnittstelle

```

1 if err != nil {
2     fmt.Printf("Something went wrong:\n%v", err)
3     os.Exit(1)
4 }

```

Abbildung 4.42: Fehlerbehandlung

Empfangen

Das Empfangen läuft in einer nebenläufigen Schleife 4.45. Das Programm innerhalb der Schleife lässt sich aufteilen in einen Teil, der die eingehende Nachricht empfängt, sowie diese in das richtige Datenformat 4.43 bringt, und in einen Teil, der diese Daten ausgewertet 4.44.

Die Schleife beginnt damit auf der seriellen Schnittstelle einzulesen, bis ein Newline-Charakter auftritt. Dieser String wird anschließend an jedem ':' geteilt. Die Fehlerbehandlung gibt den Fehlertext aus und beendet anschließend das Programm.

Senden

Eine weitere nebenläufige Schleife wird für das Senden verwendet. Der serialOut Kanal wird benutzt um Nachrichten an die Schleife zu senden. Die Nachricht muss vor dem Absenden noch in eine Bytesequenz konvertiert werden, weil Golang typsicher ist und die

```

1 str, err := reader.ReadString('\n')
2 str = strings.ReplaceAll(str, "\n", "")
3 if err != nil {
4     fmt.Printf("Error: %v\n", err)
5     os.Exit(1)
6 }
7 fmt.Println(str)
8 sstr := strings.Split(str, ":")

```

Abbildung 4.43: Empfang und Verarbeitung

```

1  switch sstr[0] {
2  case "ACK":
3      switch sstr[1] {
4          case "Received":
5              ackc <- 0
6          case "Done":
7              ready = true
8          }
9      break
10     case "Return":
11         switch sstr[1] {
12             case "Sensor":
13                 values := sstr[2:]
14                 sensor.Left, _ = strconv.Atoi(values[0])
15                 sensor.Middle, _ = strconv.Atoi(values[1])
16                 sensor.Right, _ = strconv.Atoi(values[2])
17             }
18         break
19 }

```

Abbildung 4.44: Auswertung

```

1  go func() {
2      for {
3          ...
4      }
5  }()

```

Abbildung 4.45: nebenläufige Schleife

```

1  go func() {
2      for {
3          select {
4              case msg := <-serialOut:
5                  serialPort.Write([]byte(msg))
6          }
7      }
8  }()

```

Abbildung 4.46: Sende Schleife

```

1  serialOut <- "Move:Left"

```

Abbildung 4.47: Senden mit Kanal

Write Funktion eine Bytesquenz annimmt.

4.47 ist ein Beispiel wie man Nachrichten an diese Schleife sendet.

```
1 || type HandlerFunc func(ResponseWriter, *Request)
```

Abbildung 4.48: http.HandlerFunc Typdefinition

```
1 || r.HandleFunc("/sensors", sensors)
```

Abbildung 4.49: Registrierung einer http.HandlerFunc

4.13.2 HTTP

Um eine HTTP API in Go zu bauen, benötigt man eine Funktion, die vom Typ `http.HandlerFunc` 4.48 ist. Dieser Funktionstyp nimmt zwei Argumente entgegen. Das erste Argument ist ein `http.ResponseWriter`, dieser ist ein Objekt und wird benutzt, um die Anfrage zu beantworten. Das zweite Argument ist eine `http.Request`. `http.Request` ist eine Datenstruktur mit Informationen über die Anfrage.

Damit eine Funktion aufgerufen werden kann, muss sie zuerst registriert werden 4.49. In diesem Fall wird für die Zuordnung der Pfade ein Router benützt.

Letzten Endes muss der HTTP Server noch gestartet werden 4.50. Dafür muss man einen TCP/IP Port und einen HTTP-Router übergeben. Um mögliche Fehler an den Benutzer zurück zu geben, wird der Funktionsaufruf in einem `log.Fatal` gemacht.

Sensor Anfragen

Die Funktion `sensors` 4.51 benutzt die zwischengespeicherten Sensorwerte und gibt diese als Antwort auf Anfragen zurück. Das hierfür gewählte Datenformat ist JSON. JSON wurde gewählt, weil es einfach für eine Website zu verarbeiten ist. Wenn bei dem Konvertieren zu JSON ein Fehler auftritt, wird nur ein Fehlercode zurück gegeben.

Bewegungsfunktionen

Die Funktionen, die die Bewegung steuern, werden von einer anderen Funktion 4.52 erstellt. Die erstellende Funktion nimmt die Richtung als Argument entgegen und gibt eine Funktion zurück, die bei einem Aufruf die entsprechende Richtung an die serielle Schnittstelle weitergibt. Zudem wird in der Bewegungsfunktion überprüft, ob der ATmega328P

```
1 || log.Fatal(http.ListenAndServe(":8420", r))
```

Abbildung 4.50: Starten des HTTP-Servers

```

1 func sensors(w http.ResponseWriter, r *http.Request) {
2     b, err := json.Marshal(sensor)
3     if err != nil {
4         w.WriteHeader(500)
5         return
6     }
7     w.Write(b)
8 }

```

Abbildung 4.51: Funktion für Sensorwerte

```

1 func move(dir string) func(w http.ResponseWriter, r *http.Request) {
2     return func(w http.ResponseWriter, r *http.Request) {
3         if !ready {
4             w.WriteHeader(http.StatusBadRequest)
5         }
6         ackmutex.Lock()
7         serialOut <- "Move:" + dir
8         _ = <-ackc
9         ready = false
10        ackmutex.Unlock()
11        w.WriteHeader(200)
12    }
13 }

```

Abbildung 4.52: Funktion die Funktion für Bewegung zurück gibt

bereit ist, einen neuen Befehl zu bekommen. Wenn dieser nicht bereit ist, wird die Anfrage fallen gelassen.

Webfrontend

Um das Webfrontend 4.14 bereit zu stellen, muss der Ordner in dem die Dateien dafür liegen freigegeben werden. Dafür wird ein eigener Pfad `/p/` benutzt. Damit der Go Dateiserver aber die richtigen Dateien holt, muss man mit `http.StripPrefix` das `/p/` entfernen.

```

1 r.PathPrefix("/p/").Handler(http.StripPrefix("/p/", http.FileServer(http
    .Dir("./page/"))))

```

Abbildung 4.53: Starten des HTTP-Servers

4.14 Benutzteroberfläche

Für die Benutzteroberfläche wurde eine Website gewählt, weil diese für Computer, Smartphones und Tablets einfach erreichbar ist.

4.14.1 HTML

HTML stellt das Grundgerüst für die Website dar. Um später ein Raster mit den Knöpfen zu bilden, sind diese in einem Container 4.54. Der Text innerhalb der Knöpfe ist auch in einem eigenen Element, um das Stylesheet zu vereinfachen.

```
1 <div class="buttonmatrix">
2   <div class="button" id="tleft">
3     <div class="buttoninner">Turn Left</div>
4   </div>
5   ...
6 </div>
```

Abbildung 4.54: HTML Body

4.14.2 CSS

4.55 zeigt wie man CSS aus einer Datei einbindet.

```
1 <link rel="stylesheet" href="style.css" />
```

Abbildung 4.55: Einbinden des CSS

Das Rasterlayout wird im Style des Containers festgelegt 4.56. Mit `margin:auto` wird dieser automatisch zentriert. In 4.57 ist das Ergebnis des Rasterlayouts mit eingezeichneten Debuglinien zu sehen.

4.14.3 Javascript

Das Laden des Skriptes 4.58 muss nach dem Laden der Knöpfe erfolgen. Deshalb steht 4.58 im HTML-Dokument am Ende des bodys.

Die Aufgabe des Skriptes ist es, den Knöpfen ein `OnClick`-Event zuzuweisen. Dafür werden alle Elemente, die ein Knopf sind, in einem Array gesammelt. Anschließend wird für jedes Element des Arrays eine Funktion für `onclick` zugewiesen. Diese Funktion ruft die Endpunkte der API, die in 4.13.2 angelegt wurden.

```

1 .buttonmatrix {
2   display: grid;
3   width: 60vh;
4   height: 40vh;
5   grid-template-rows: 20vh 20vh;
6   grid-template-columns: 20vh 20vh 20vh;
7   margin: auto;
8 }

```

Abbildung 4.56: CSS für den Container

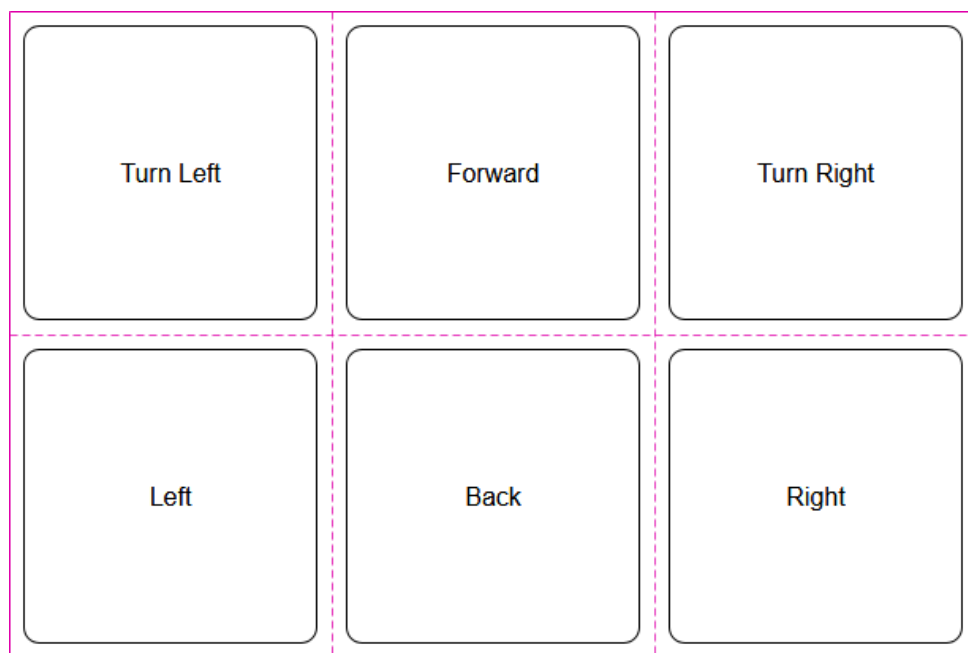


Abbildung 4.57: Rasterlayout mit Debuglinien

```

1 <script src="script.js"></script>

```

Abbildung 4.58: Einbinden des Javascript

```

1 Array.from(document.getElementsByClassName("button")).forEach((element)
   => {
2   element.onclick = () => {
3     fetch("/") + element.id);
4   };
5 });

```

Abbildung 4.59: Zusweisung der OnClick-Funktionen

Kapitel 5

Ergebnisse

5.1 Mechanische Ergebnisse

5.1.1 Befestigungsplatte

Weil die Leiterplatte, die DC/DC Netzteile und der Akku, nicht mit den bestehenden Befestigungsmöglichkeiten kompatibel sind, musste die Befestigungsplatte hergestellt werden. Der Akku wurde auf der Platte am vorderen Teil platziert, was zur Folge hat, dass sich die Platte ein bisschen biegt. Im Nachhinein wäre es wahrscheinlich besser, die Position des Akkus und der Leiterplatte zu tauschen, das wurde jedoch nicht gemacht, damit die Leitungen zwischen den Magnetventilen und dem PCB nicht zu lang werden.

5.1.2 Akkubefestigung

Die 3D gedruckte Befestigung für den Akku, wurde bewusst zu groß gefertigt, um Spielraum für einen eventuell größeren Akku zu gewinnen.

5.2 Leiterplatte

Die Leiterplatte vom PE-Bug ist größtenteils gelungen, jedoch gibt es natürlich ein paar Punkte, die man verbessern könnte.

- Versorgungsspannungsstecker auf der falschen Seite: Der Stecker für die Spannungsversorgung wurde auf die Bottom Seite platziert. Der Stecker hat zwar auf der Top Seite eingelötet werden können, aber der Silkscreen 5.1 passt nicht mit dem Ergebnis zusammen, weil der Stecker keine weiße Box als Umrandung hat.
- Debug LEDs unter dem Raspberry Pi: Die Debug LEDs, für die Magnetventile,

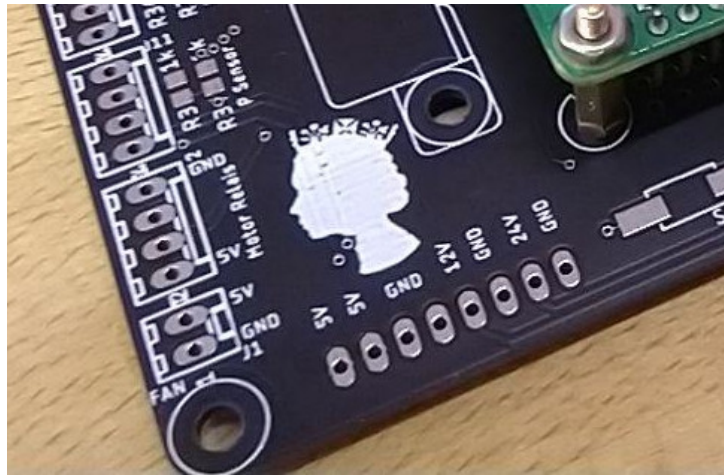


Abbildung 5.1: Der Stecker für die Spannungsversorgung, ohne dem Silkscreen.

wurden unter dem Raspberry Pi angebracht, was das debuggen erschwert, wenn der Raspberry Pi angesteckt ist. Es wäre sinnvoller gewesen, die Leiterbahnen, für die LEDs, unter dem Raspberry Pi herauszuführen, und auf dem Silkscreen kennzeichnen, welche LED für welches Magnetventil verantwortlich ist.

- Leiterbahnen besser geführt: Einige Leiterbahnen nehmen einen merkwürdigen Pfad, zum Beispiel gibt es einige Leiterbahnen, die um Bauteile herumgeführt sind, die aber auf der anderen Seite der Leiterplatte sind.

5.3 Pneumatik

Es wäre sinnvoll gewesen, schon früher die Pneumatik zu testen, weil das Problem mit dem 5/2 Wegeventil, vielleicht schon vor dem Covid-19 bedingten Lockdown gelöst worden wäre, und die Bewegungsabläufe hatte man besser testen können.

5.4 Firmware Ergebnisse

Auch wenn die Firmware ihre Aufgabe erfüllt, verlief die Entwicklung nicht ohne Probleme.

5.4.1 Alte Firmware

Während der Entwicklung der Firmware wurde noch einmal die Aufgabenstellung überarbeitet was zur Folge hatte, dass die komplette Firmware überarbeitet werden musste, obwohl diese schon voll funktionsfähig war. Zudem wurde die alte Firmware durch Speicherprobleme geplagt, die durch unsaubere Programmierung hervorgerufen wurden.

5.4.2 Neue Firmware

Mit der Neuentwicklung wurden neben den neuen Anforderungen auch die Speicherprobleme gelöst. Dadurch wurde jedoch das gesamte Projekt verzögert. Die eigentliche Neuentwicklung verlief jedoch ohne Probleme und konnte durch die Entwicklung des externen Programmes immens beschleunigt werden.

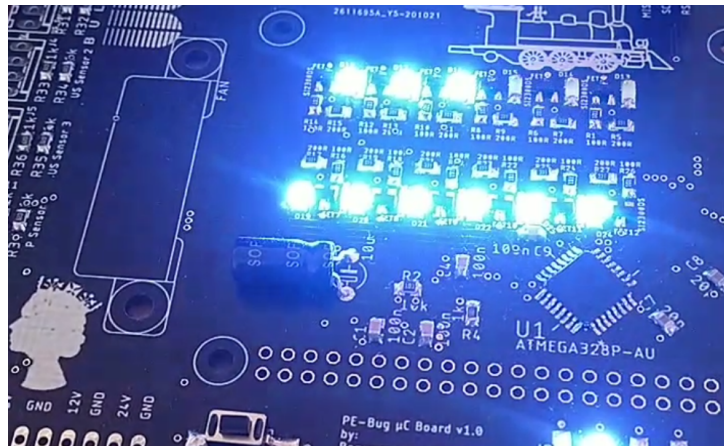


Abbildung 5.2: Microcontroller mit hochgeladener Firmware.

5.5 Raspberry Pi Software

5.5.1 Weboberfläche

Die Entwicklung der Weboberfläche verlief ohne größere Probleme. Das finale Ergebnis ist eine simple Oberfläche 5.3, jedoch ist diese einfach zu erweitern. Nicht nur das Aussehen sondern auch die funktionelle Seite funktioniert wie geplant. Die Überprüfung der richtigen Vergabe der Events wurde mit dem Firefox Debugger bewerkstelligt 5.4.

5.5.2 Backend

Um den Netzwerkteil des Backends zu verifizieren, wurden HTTP-Anfragen mit Wireshark aufgenommen. Das Weitersenden an den PE-Bug wurde durch Tests in Verbindung mit dem ATmega328P verifiziert.

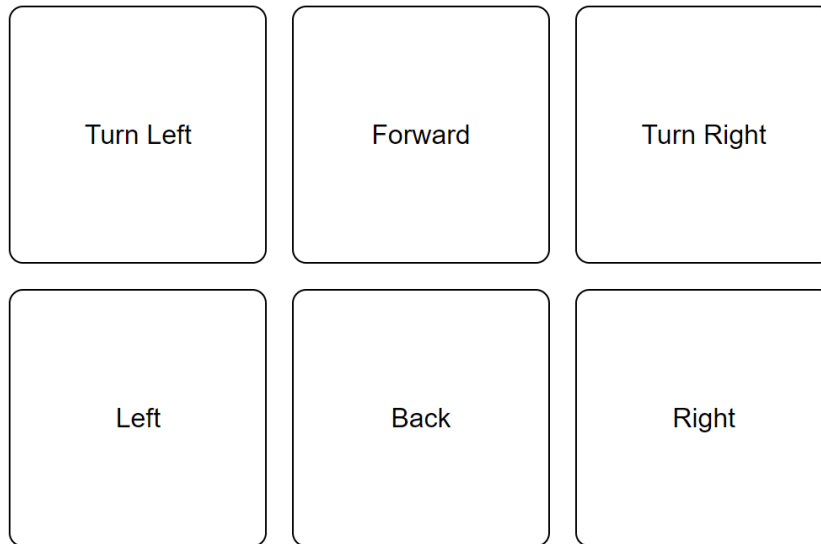


Abbildung 5.3: Finale Version der Weboberfläche

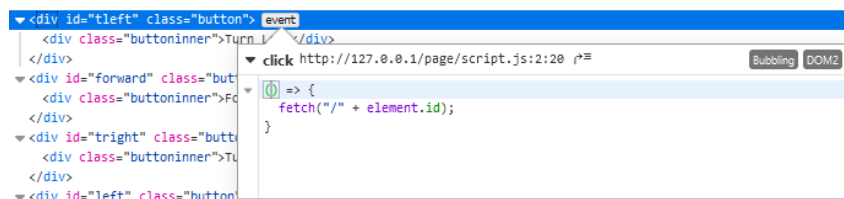


Abbildung 5.4: Knopf mit zugewiesenem Event

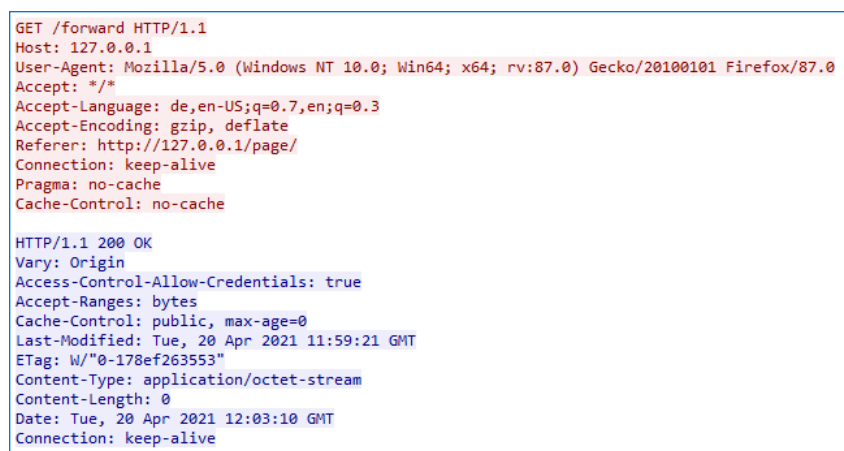


Abbildung 5.5: Inhalt der Anfrage (rot) und der Antwort (blau)

138	6.273549	127.0.0.1	127.0.0.1	TCP	52	10735 → 80 [SYN] Seq=0 Win=65495 Len=0 MSS=65495 SACK_PERM=1
139	6.273580	127.0.0.1	127.0.0.1	TCP	52	80 → 10735 [SYN, ACK] Seq=0 Ack=1 Win=65495 Len=0 MSS=65495 SACK_PERM=1
140	6.273599	127.0.0.1	127.0.0.1	TCP	44	10735 → 80 [ACK] Seq=1 Ack=1 Win=65495 Len=0
141	6.273686	127.0.0.1	127.0.0.1	HTTP	365	GET /forward HTTP/1.1
142	6.273695	127.0.0.1	127.0.0.1	TCP	44	80 → 10735 [ACK] Seq=1 Ack=322 Win=65174 Len=0
147	6.274837	127.0.0.1	127.0.0.1	HTTP	364	HTTP/1.1 200 OK
148	6.274851	127.0.0.1	127.0.0.1	TCP	44	10735 → 80 [ACK] Seq=322 Ack=321 Win=65175 Len=0
194	11.282274	127.0.0.1	127.0.0.1	TCP	44	80 → 10735 [FIN, ACK] Seq=321 Ack=322 Win=65174 Len=0
195	11.282286	127.0.0.1	127.0.0.1	TCP	44	10735 → 80 [ACK] Seq=322 Ack=322 Win=65175 Len=0
198	11.282373	127.0.0.1	127.0.0.1	TCP	44	10735 → 80 [FIN, ACK] Seq=322 Ack=322 Win=65175 Len=0
199	11.282392	127.0.0.1	127.0.0.1	TCP	44	80 → 10735 [ACK] Seq=322 Ack=323 Win=65174 Len=0

Abbildung 5.6: TCP Verlauf der Anfrage

Status	Methode	Host	Datei	Initiator	Typ	Übertragen	Größe
200	GET	127.0.0.1	forward	script.js:3 (fetch)	octet-stream	320 B	0 B

Abbildung 5.7: Anfrage im Netzwerkanalyse Tabs des Firefox Debuggers

Kapitel 6

Resümee

Wie in der Aufgabenstellung erwähnt wurde, war das Ziel dieses Projektes die Wiederherstellung und Modernisierung der Spinne, die im Rahmen dieses Projektes den Namen PE-Bug verliehen bekam.

6.1 Hardware

Die Vermutung der Funktionsfähigkeit der Gelenke und Lager hat sich bestätigt. Keine dieser musste ausgetauscht werden. Anders jedoch die pneumatische Hardware. Diese zeigte sehr wohl Spuren ihres Alters, so wurden alle pneumatischen Schläuche zur Sicherheit ausgetauscht. Weiters musste ein Ventil ausgetauscht werden, welches während dem Transport zerstört wurde. Komplett ausgetauscht wurde die elektronische Hardware. Dabei wurde auf eine Lösung mit zwei Controllern gesetzt. Einmal dem ATmega328p, der für die Ansteuerung der Hardware zuständig ist, sowie einem Raspberry PI, der die Benutzeroberfläche übernimmt.

6.2 Software

Anders als bei der Hardware, war es bei der Software nicht möglich das bereits existierende System zu übernehmen. Um dieses Problem für künftige Projekte zu verhindern wurde ein ATmega328p verwendet, der die Sensor Kommunikation sowie die Ansteuerung der Pneumatik übernimmt. Die Idee ist, dass diese Software nicht mehr verändert wird. Soll der PE-Bug also erweitert werden muss auf den Raspberry PI zurückgegriffen werden. Dieser übernimmt momentan nur die Ansteuerung des ATmega328p's sowie die Benutzeroberfläche. Es ist jedoch angedacht das dieser in Folge von zukünftigen Projekten mit weiteren Features, wie zum Beispiel Artificial Intelligenz, ausgestattet wird.

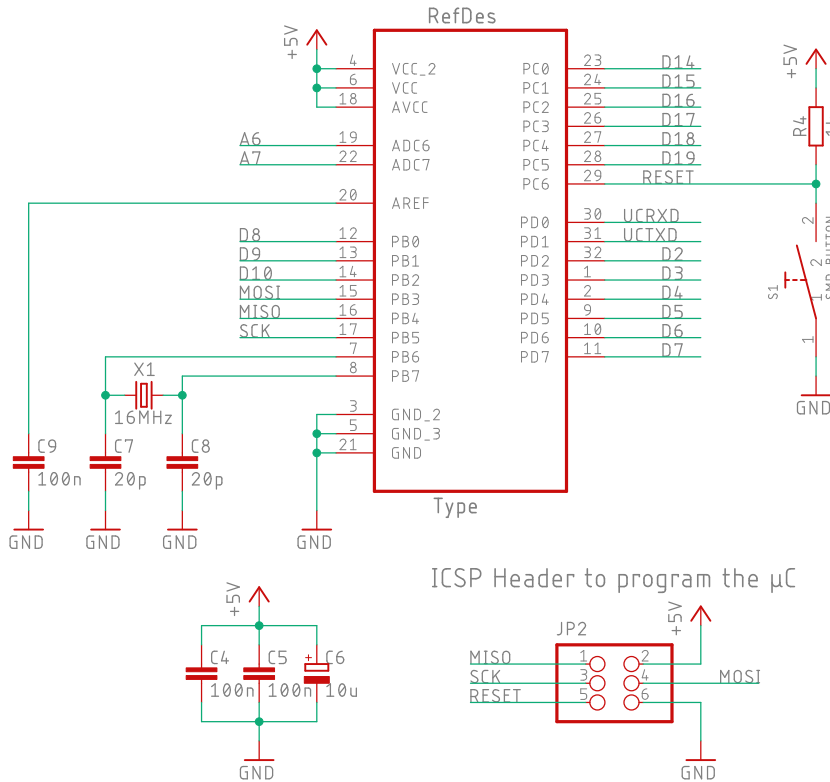
6.3 Probleme

Die Entwicklung des PE-Bugs verlief größtenteils reibungslos. Es mussten keine Projektverzögernden Lieferprobleme verzeichnet werden und auch die Software war vor großen Bugs verschont. Nicht unberührt blieb das Projekt jedoch von der Covid-19 Pandemie. Diese verzögerte die Entwicklung um mehrere Wochen, da das Testen der Hardware sowie die Kommunikation zwischen Hardware und Software nur eingeschränkt möglich war.

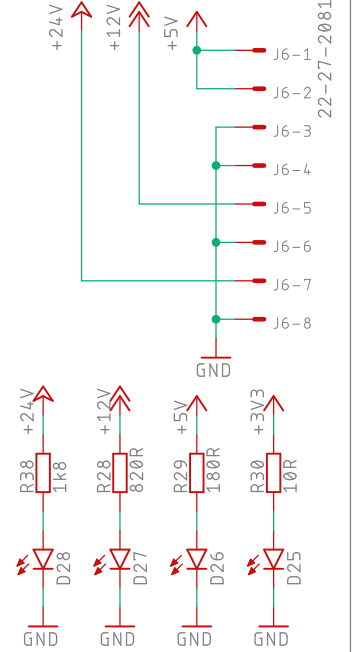
Anhang A

Schaltplan

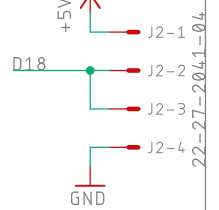
Microcontroller Schematic taken from the Arduino Nano



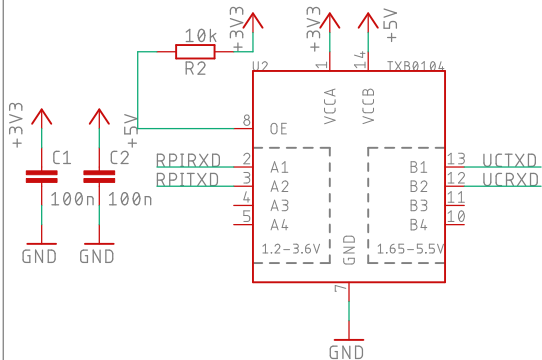
Power Plug and Power LEDs



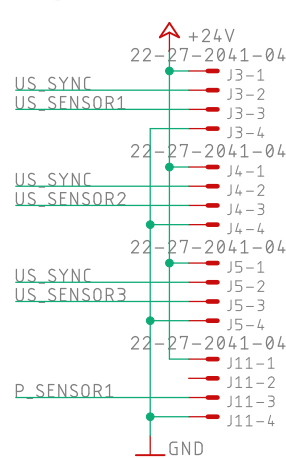
Header for the Motor Relays to connect to



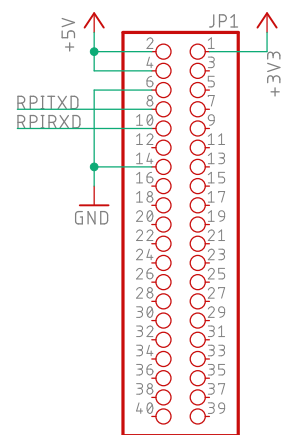
Level Shifter for UART



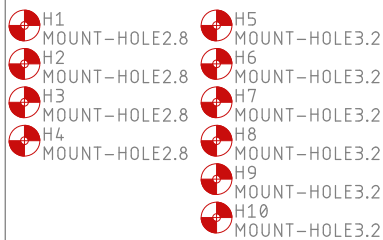
Plugs for Sensors



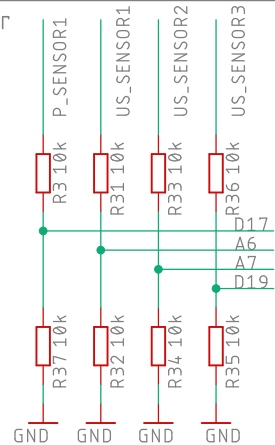
Raspberry PI 40 Pin GPIO Header



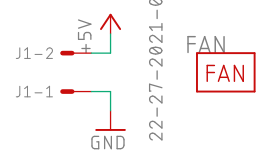
Mounting Holes for M2.5 and M3 screws



Voltage Divider for Sensors



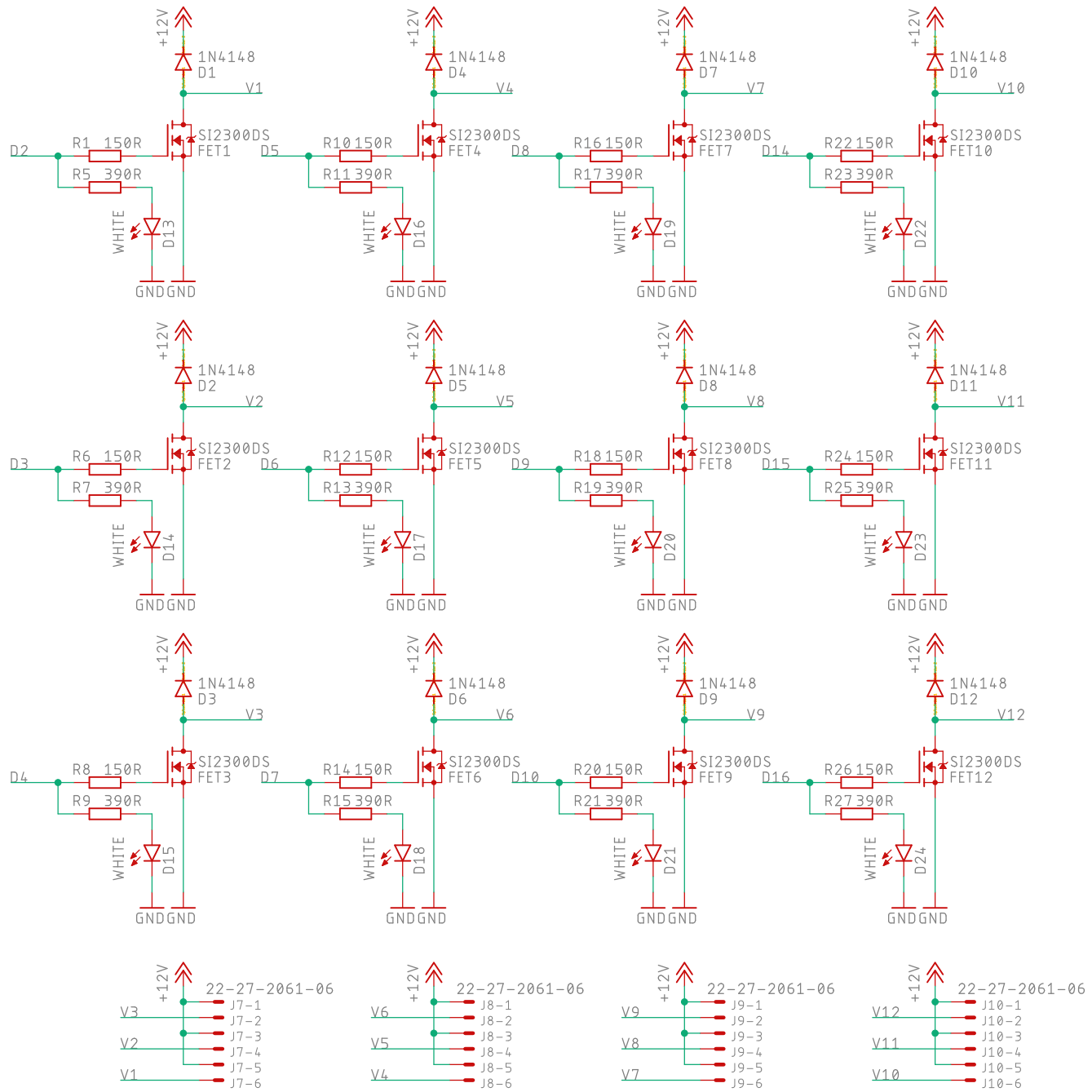
Fan Header



Date: 09.04.21 17:33

TITLE: HATv1.1

Sheet: 1/2



Date: 09.04.21 17:33

TITLE: HATv1.1

Sheet: 1/2

Anhang B

Leiterplattenlayout

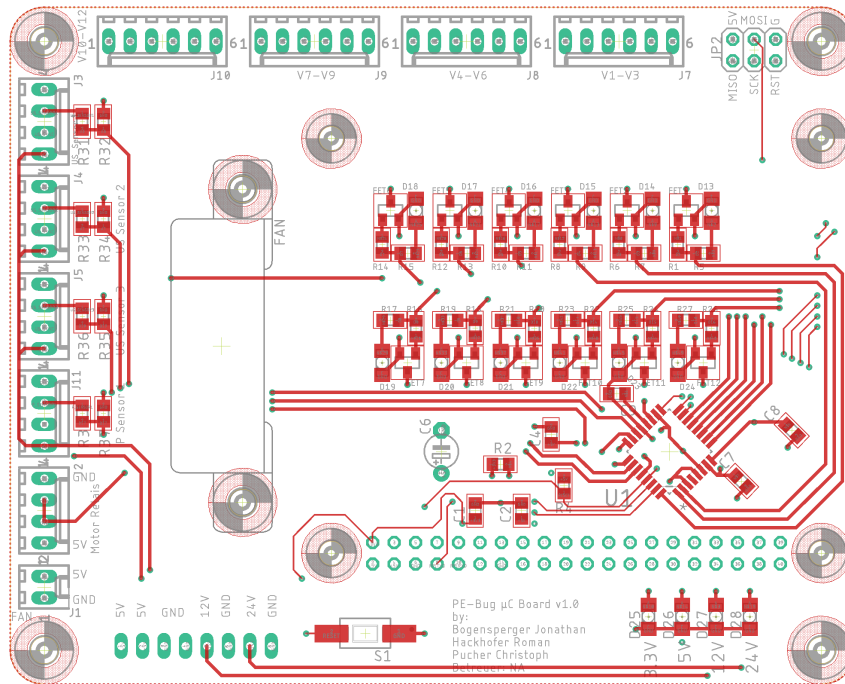


Abbildung B.1: Top Seite der Leiterplatte

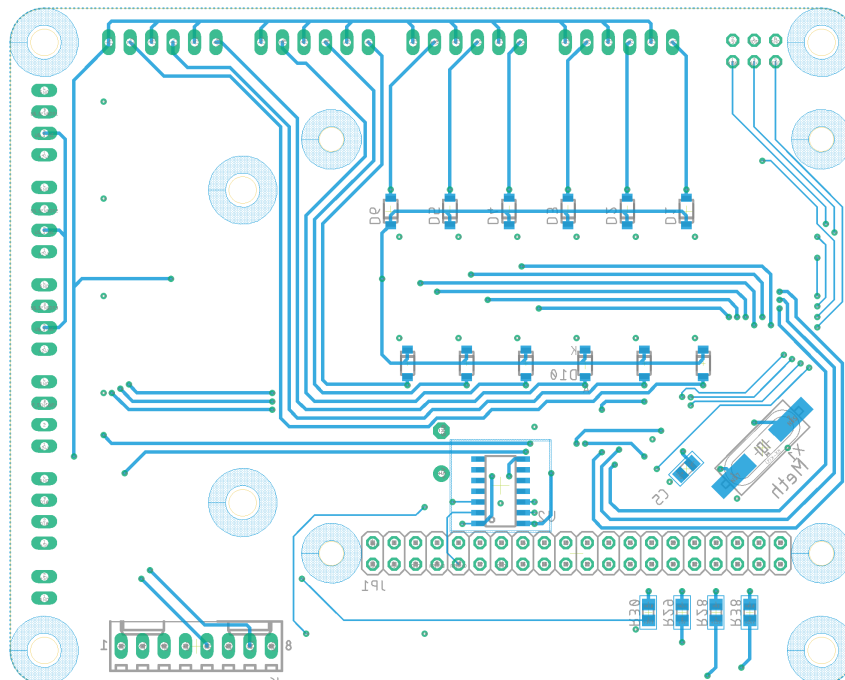


Abbildung B.2: Bottom Seite der Leiterplatte

Anhang C

3D gedruckte Befestigungen

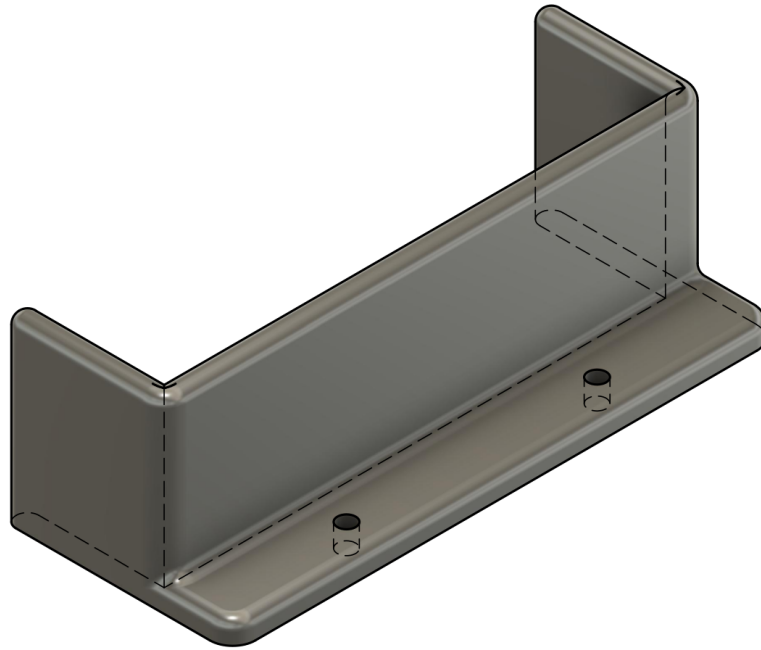


Abbildung C.1: 3D gedruckte Befestigung für den Akku

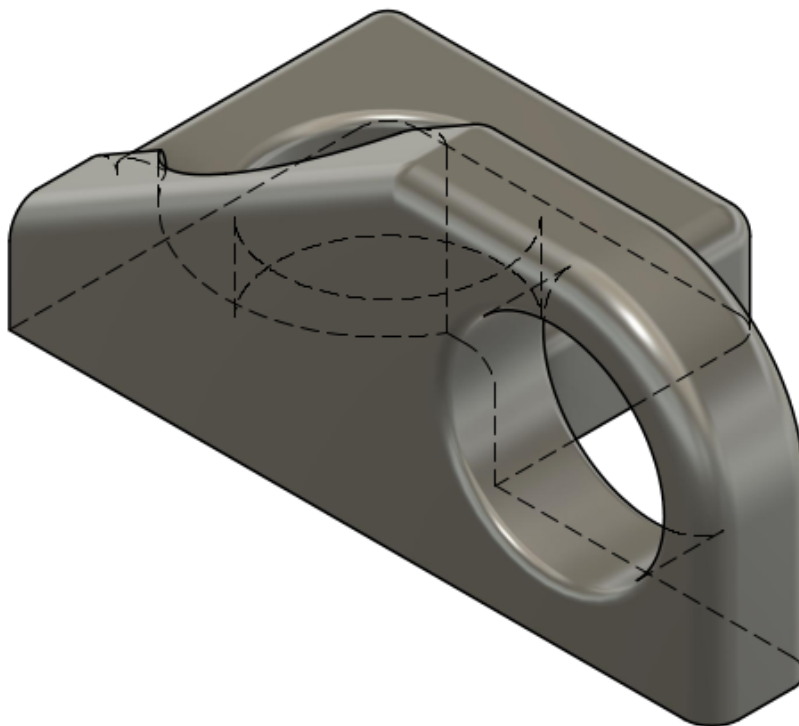
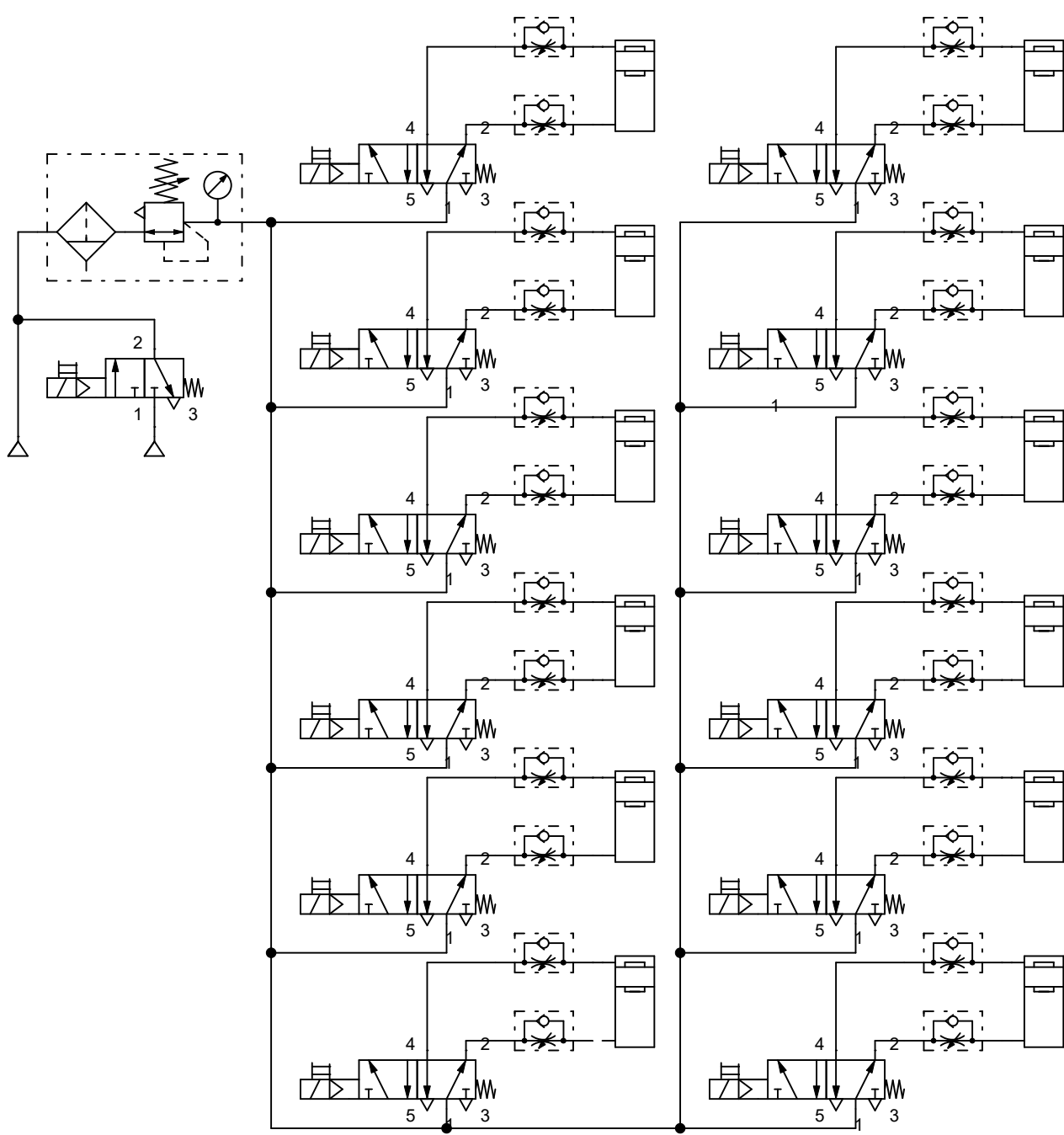


Abbildung C.2: 3D gedruckte Befestigung für den Lüfter

Anhang D

Pneumatische Schaltung



Anhang E

Golang Server

E.1 main.go

```
1 package main
2
3 import (
4     "bufio"
5     "fmt"
6     "log"
7     "net/http"
8     "os"
9     "strconv"
10    "strings"
11    "sync"
12
13    "github.com/gorilla/mux"
14    "go.bug.st/serial"
15 )
16
17 type sensorVals struct {
18     Left    int `json:"left"`
19     Middle  int `json:"middle"`
20     Right   int `json:"right"`
21 }
22
23 var (
24     ackmutex sync.Mutex
25     ackc     chan interface{}
26     running  bool
27     ready    bool
```

```

28     serialOut chan string
29     sensor    sensorVals
30 )
31
32 func main() {
33     ackmutex = sync.Mutex{}
34     ackc = make(chan interface{})
35     serialOut = make(chan string)
36     sensor = sensorVals{}
37     serialPortPath := os.Args[1]
38     ready = true
39
40     m := serial.Mode{
41         BaudRate: 9600,
42     }
43
44     serialPort, err := serial.Open(serialPortPath, &m)
45     if err != nil {
46         fmt.Printf("Something went wrong:\n%v", err)
47         os.Exit(1)
48     }
49     serialPort.Write([]byte("Conf:Delay:100\n"))
50     go func() {
51         for {
52             select {
53             case msg := <-serialOut:
54                 serialPort.Write([]byte(msg))
55             }
56
57         }
58     }()
59
60     reader := bufio.NewReader(serialPort)
61
62     go func() {
63         for {
64             str, err := reader.ReadString('\n')
65             str = strings.ReplaceAll(str, "\n", "")
66             if err != nil {
67                 fmt.Printf("Error: %v\n", err)
68                 os.Exit(1)
69             }

```

```

70     fmt.Println(str)
71     sstr := strings.Split(str, ":")
72     switch sstr[0] {
73     case "ACK":
74         switch sstr[1] {
75         case "Received":
76             ackc <- 0
77         case "Done":
78             ready = true
79         }
80         break
81     case "Return":
82         switch sstr[1] {
83         case "Sensor":
84             values := sstr[2:]
85             sensor.Left, _ = strconv.Atoi(values[0])
86             sensor.Middle, _ = strconv.Atoi(values[1])
87             sensor.Right, _ = strconv.Atoi(values[2])
88
89         }
90         break
91     }
92 }
93 }()
94
95 r := mux.NewRouter().StrictSlash(true)
96 r.PathPrefix("/p/").Handler(http.StripPrefix("/p/", http.FileServer(
97     http.Dir("./page/"))))
98 r.HandleFunc("/", redirectRoot)
99 r.HandleFunc("/right", move("Right"))
100 r.HandleFunc("/forward", move("Forward"))
101 r.HandleFunc("/left", move("Left"))
102 r.HandleFunc("/back", move("Back"))
103 r.HandleFunc("/tright", turn("Right"))
104 r.HandleFunc("/tleft", turn("Left"))
105 r.HandleFunc("/sensors", sensors)
106 log.Fatal(http.ListenAndServe(":8420", r))

```

E.2 routes.go

```

1 package main

```

```

2
3 import (
4     "encoding/json"
5     "net/http"
6 )
7
8 func move(dir string) func(w http.ResponseWriter, r *http.Request) {
9     return func(w http.ResponseWriter, r *http.Request) {
10         if !ready {
11             w.WriteHeader(http.StatusBadRequest)
12         }
13         ackmutex.Lock()
14         serialOut <- "Move:" + dir
15         _ = <-ackc
16         ready = false
17         ackmutex.Unlock()
18         w.WriteHeader(200)
19     }
20 }
21
22 func turn(dir string) func(w http.ResponseWriter, r *http.Request) {
23     return func(w http.ResponseWriter, r *http.Request) {
24         if !ready {
25             w.WriteHeader(http.StatusBadRequest)
26         }
27         ackmutex.Lock()
28         serialOut <- "Turn:" + dir
29         _ = <-ackc
30         ready = false
31         ackmutex.Unlock()
32         w.WriteHeader(200)
33     }
34 }
35
36 func sensors(w http.ResponseWriter, r *http.Request) {
37     b, err := json.Marshal(sensor)
38     if err != nil {
39         w.WriteHeader(500)
40         return
41     }
42     w.Write(b)
43 }

```

```
44 ||
45 | func redirectRoot(w http.ResponseWriter, r *http.Request) {
46 |     http.Redirect(w, r, "/p/", http.StatusFound)
47 | }
```

Anhang F

Website Code

```

1  <!DOCTYPE html>
2  <html lang="en">
3    <head>
4      <meta charset="UTF-8" />
5      <meta name="viewport" content="width=device-width, initial-scale=1.0
        " />
6      <title>PE-Bug Remote</title>
7      <link rel="stylesheet" href="style.css" />
8    </head>
9    <body>
10     <div class="buttonmatrix">
11       <div class="button" id="tleft">
12         <div class="buttoninner">Turn Left</div>
13       </div>
14       <div class="button" id="forward">
15         <div class="buttoninner">Forward</div>
16       </div>
17       <div class="button" id="tright">
18         <div class="buttoninner">Turn Right</div>
19       </div>
20       <div class="button" id="left"><div class="buttoninner">Left</div><
        /div>
21       <div class="button" id="back"><div class="buttoninner">Back</div><
        /div>
22       <div class="button" id="right"><div class="buttoninner">Right</div>
        </div>
23     </div>
24     <script src="script.js"></script>
25   </body>
26 </html>

```

Abbildung F.1: index.html

```

1  body {
2      font-family: sans-serif;
3      display: grid;
4      align-items: center;
5      height: 100vh;
6      width: 100vw;
7      margin: 0;
8      overflow: hidden;
9  }
10 .buttonmatrix {
11     display: grid;
12     width: 60vh;
13     height: 40vh;
14     grid-template-rows: 20vh 20vh;
15     grid-template-columns: 20vh 20vh 20vh;
16     margin: auto;
17 }
18 .button {
19     margin: auto;
20     width: 90%;
21     height: 90%;
22     vertical-align: middle;
23     border: solid black 1px;
24     border-radius: 1vh;
25     transition: all 0.2s;
26 }
27 .buttoninner {
28     height: 100%;
29     line-height: 18vh;
30     vertical-align: middle;
31     text-align: center;
32     margin: auto;
33     user-select: none;
34 }
35 .button:active {
36     transform: scale(0.98) translate(-5px, -5px);
37 }

```

Abbildung F.2: style.css

```
1 Array.from(document.getElementsByClassName("button")).forEach((element)
    => {
2   element.onclick = () => {
3     fetch("/") + element.id);
4   };
5 });
```

Abbildung F.3: script.js

Anhang G

Alte Firmware

G.1 Pinbelegung

```
1  #include <Arduino.h>
2  /*****
3      right | leg [nr] | turn/elevate
4      left  | leg [nr] | turn/elevate
5  *****/
6
7  #define r_l1_t 2
8  #define r_l1_e 3
9  #define r_l2_t 4
10 #define r_l2_e 5
11 #define r_l3_t 6
12 #define r_l3_e 7
13 #define l_l1_t 8
14 #define l_l1_e 9
15 #define l_l2_t 10
16 #define l_l2_e 14
17 #define l_l3_t 15
18 #define l_l3_e 16
19
20 /*****
21     compressor | [nr]
22 *****/
23
24 #define c 18
25
26 /*****
27     pressure sensor | [nr]
```

```

28  *****/
29
30  #define ps A3
31
32  /******
33      ultrasonic sensor / [position]
34  *****/
35
36  #define us_m A5
37  #define us_l A6
38  #define us_r A7

```

G.2 Einstellungen

```

1  /******
2      timings in ms
3  *****/
4
5  unsigned long Delay = 100;
6  unsigned long SensorTimeout = 10000;
7  unsigned long CompressorTimeout = 100000;
8
9  /******
10     baudrate
11  *****/
12
13  #define baud 9600

```

G.3 Hauptprogramm

```

1  #include <Arduino.h>
2  #include <settings.h>
3  #include <gpio.h>
4  #include <string.h>
5
6  /******
7      enum definition
8  *****/
9  #pragma region enum
10 enum side
11 {
12     left,

```

```

13     right
14 };
15 enum position
16 {
17     front,
18     middle,
19     back
20 };
21 enum piston
22 {
23     elevation,
24     rotation
25 };
26 enum compressor
27 {
28     on,
29     off,
30     aut
31 };
32 #pragma endregion
33
34 /*****
35     var definition
36 *****/
37 #pragma region var
38 // serial vars
39 char *category = nullptr;
40 char *option = nullptr;
41 char *value = nullptr;
42 const char *serialState = "ready";
43 char serialRead[50] = {0};
44 int compressor = aut;
45 //instructionj vars
46 int index = 0;
47 int length = 0;
48 //time vars
49 unsigned long time = millis();
50 unsigned long sensorTime = millis();
51 unsigned long compressorTime = millis();
52
53 int legs[2][3][2] = {
54     {{l_11_e, l_11_t},

```

```

55     {l_12_e, l_12_t},
56     {l_13_e, l_13_t}},
57     {{r_11_e, r_11_t},
58     {r_12_e, r_12_t},
59     {r_13_e, r_13_t}}});
60
61 struct instruction
62 {
63     static const int size = sizeof(int) * 4 + sizeof(bool);
64     int side;
65     int position;
66     int piston;
67     bool direction;
68     unsigned long timeout;
69 };
70 #pragma endregion
71
72 /*****
73     function init
74 *****/
75 #pragma region functions
76 void read_command();
77 void movement();
78 void serial_read();
79 void serial_reset();
80 void get_sensor();
81 bool check_string(const char* string);
82 bool get_direction(instruction inst, bool direction);
83 //read sensor
84 void read_sensor_distance();
85 void read_sensor_pressure();
86 #pragma endregion
87
88 /*****
89     instructions
90 *****/
91 #pragma region instructions
92 /*****
93     instruction array
94 *****/
95 instruction *current_inst;
96

```

```

97  /*****
98      check instructions
99  *****/
100 instruction check[6] = {
101     {left, front, rotation},
102     {left, middle, rotation},
103     {left, back, rotation},
104     {right, front, rotation},
105     {right, middle, rotation},
106     {right, back, rotation}};
107
108  /*****
109      movement instructions
110  *****/
111 instruction move_forward_llr[13] = {
112     //elevate
113     {right, front, elevation, true, 0},
114     {right, back, elevation, true, 0},
115     {left, middle, elevation, true, 0},
116     //rotate
117     {left, front, rotation, false, 1},
118     {left, middle, rotation, true, 0},
119     {left, back, rotation, false, 0},
120     {right, front, rotation, true, 0},
121     {right, middle, rotation, false, 0},
122     {right, back, rotation, true, 0},
123     //elevate
124     {right, front, elevation, false, 10},
125     {right, back, elevation, false, 0},
126     {left, middle, elevation, false, 0},
127     {left, middle, elevation, false, 10}};
128 instruction move_forward_rrl[13] = {
129     //elevate
130     {left, front, elevation, true, 0},
131     {left, back, elevation, true, 0},
132     {right, middle, elevation, true, 0},
133     //rotate
134     {left, front, rotation, true, 1},
135     {left, middle, rotation, false, 0},
136     {left, back, rotation, true, 0},
137     {right, front, rotation, false, 0},
138     {right, middle, rotation, true, 0},

```

```

139     {right, back, rotation, false, 0},
140     //elevate
141     {left, front, elevation, false, 10},
142     {left, back, elevation, false, 0},
143     {right, middle, elevation, false, 0},
144     {right, middle, elevation, false, 10}};
145 instruction move_back_llr[13] = {
146     //elevate
147     {right, front, elevation, true, 0},
148     {right, back, elevation, true, 0},
149     {left, middle, elevation, true, 0},
150     //rotate
151     {left, front, rotation, true, 1},
152     {left, middle, rotation, false, 0},
153     {left, back, rotation, true, 0},
154     {right, front, rotation, false, 0},
155     {right, middle, rotation, true, 0},
156     {right, back, rotation, false, 0},
157     //elevate
158     {right, front, elevation, false, 10},
159     {right, back, elevation, false, 0},
160     {left, middle, elevation, false, 0},
161     {left, middle, elevation, false, 10}};
162 instruction move_back_rrl[13] = {
163     //elevate
164     {left, front, elevation, true, 0},
165     {left, back, elevation, true, 0},
166     {right, middle, elevation, true, 0},
167     //rotate
168     {left, front, rotation, false, 1},
169     {left, middle, rotation, true, 0},
170     {left, back, rotation, false, 0},
171     {right, front, rotation, true, 0},
172     {right, middle, rotation, false, 0},
173     {right, back, rotation, true, 0},
174     //elevate
175     {left, front, elevation, false, 10},
176     {left, back, elevation, false, 0},
177     {right, middle, elevation, false, 0},
178     {right, middle, elevation, false, 10}};
179 instruction move_turn_right[16] = {
180     //SETUP

```

```

181 //elevate
182 {left, front, elevation, true, 0},
183 {left, back, elevation, true, 0},
184 //rotate
185 {left, front, rotation, true, 1},
186 {left, back, rotation, true, 0},
187 //elevate
188 {left, front, elevation, false, 10},
189 {left, back, elevation, false, 0},
190 {left, back, elevation, false, 10},
191 //START
192 //elevation
193 {left, middle, elevation, true, 0},
194 {right, front, elevation, true, 0},
195 {right, back, elevation, true, 0},
196 //rotate
197 {left, front, rotation, false, 1},
198 {left, back, rotation, false, 0},
199 //elevation
200 {left, middle, elevation, false, 10},
201 {right, front, elevation, false, 0},
202 {right, back, elevation, false, 0},
203 {right, back, elevation, false, 10}};
204 instruction move_turn_left[16] = {
205 //SETUP
206 //elevate
207 {right, front, elevation, true, 0},
208 {right, back, elevation, true, 0},
209 //rotate
210 {right, front, rotation, true, 1},
211 {right, back, rotation, true, 0},
212 //elevate
213 {right, front, elevation, false, 10},
214 {right, back, elevation, false, 0},
215 {right, back, elevation, false, 10},
216 //START
217 //elevation
218 {right, middle, elevation, true, 0},
219 {left, front, elevation, true, 0},
220 {left, back, elevation, true, 0},
221 //rotate
222 {right, front, rotation, false, 1},

```

```

223     {right, back, rotation, false, 0},
224     //elevation
225     {right, middle, elevation, false, 10},
226     {left, front, elevation, false, 0},
227     {left, back, elevation, false, 0},
228     {left, back, elevation, false, 1}};
229 instruction move_rotation_right[27] = {
230     //SETUP llr
231     //elevation
232     {left, front, elevation, true, 0},
233     {left, back, elevation, true, 0},
234     {right, middle, elevation, true, 0},
235     //rotate
236     {left, front, rotation, true, 1},
237     {left, back, rotation, true, 0},
238     {right, middle, rotation, false, 0},
239     //elevation
240     {left, front, elevation, false, 10},
241     {left, back, elevation, false, 0},
242     {right, middle, elevation, false, 0},
243     {right, middle, elevation, false, 10},
244     //SETUP rrl
245     //elevation
246     {right, front, elevation, true, 0},
247     {right, back, elevation, true, 0},
248     {left, middle, elevation, true, 0},
249     //rotate
250     {right, front, rotation, false, 1},
251     {right, back, rotation, false, 0},
252     {left, middle, rotation, true, 0},
253     //elevation
254     {right, front, elevation, false, 10},
255     {right, back, elevation, false, 0},
256     {left, middle, elevation, false, 0},
257     {left, middle, elevation, false, 10},
258     //START
259     //rotation
260     {left, front, rotation, false, 0},
261     {left, middle, rotation, false, 0},
262     {left, back, rotation, false, 0},
263     {right, front, rotation, true, 0},
264     {right, middle, rotation, true, 0},

```

```

265     {right, back, rotation, true, 0},
266     {right, back, rotation, true, 10}};
267 instruction move_rotation_left[27] = {
268     //SETUP rrl
269     //elevation
270     {right, front, elevation, true, 0},
271     {right, back, elevation, true, 0},
272     {left, middle, elevation, true, 0},
273     //rotate
274     {right, front, rotation, true, 1},
275     {right, back, rotation, true, 0},
276     {left, middle, rotation, false, 0},
277     //elevation
278     {right, front, elevation, false, 10},
279     {right, back, elevation, false, 0},
280     {left, middle, elevation, false, 0},
281     {left, middle, elevation, false, 10},
282     //SETUP llr
283     //elevation
284     {left, front, elevation, true, 0},
285     {left, back, elevation, true, 0},
286     {right, middle, elevation, true, 0},
287     //rotate
288     {left, front, rotation, false, 1},
289     {left, back, rotation, false, 0},
290     {right, middle, rotation, true, 0},
291     //elevation
292     {left, front, elevation, false, 10},
293     {left, back, elevation, false, 0},
294     {right, middle, elevation, false, 0},
295     {right, middle, elevation, false, 10},
296     //START
297     //rotation
298     {left, front, rotation, true, 0},
299     {left, middle, rotation, true, 0},
300     {left, back, rotation, true, 0},
301     {right, front, rotation, false, 0},
302     {right, middle, rotation, false, 0},
303     {right, back, rotation, false, 0},
304     {right, back, rotation, false, 10}};
305 #pragma endregion
306

```

```

307  *****
308      setup
309  *****
310  void setup()
311  {
312      //MOS-FETs
313      pinMode(r_l1_e, OUTPUT);
314      pinMode(r_l1_t, OUTPUT);
315      pinMode(r_l2_e, OUTPUT);
316      pinMode(r_l2_t, OUTPUT);
317      pinMode(r_l3_e, OUTPUT);
318      pinMode(r_l3_t, OUTPUT);
319      pinMode(l_l1_e, OUTPUT);
320      pinMode(l_l1_t, OUTPUT);
321      pinMode(l_l2_e, OUTPUT);
322      pinMode(l_l2_t, OUTPUT);
323      pinMode(l_l3_e, OUTPUT);
324      pinMode(l_l3_t, OUTPUT);
325      //set elevation low
326      digitalWrite(r_l1_e, LOW);
327      digitalWrite(r_l2_e, LOW);
328      digitalWrite(r_l3_e, LOW);
329      digitalWrite(l_l1_e, LOW);
330      digitalWrite(l_l2_e, LOW);
331      digitalWrite(l_l3_e, LOW);
332      //ultrasonic sensors
333      pinMode(us_l, INPUT);
334      pinMode(us_m, INPUT);
335      pinMode(us_r, INPUT);
336      //pressure sensor
337      pinMode(ps, INPUT);
338      //compressors
339      pinMode(c, OUTPUT);
340      //serial start-up
341      Serial.begin(baud);
342  }
343
344  *****
345      main
346  *****
347  void loop()
348  {

```

```

349     get_sensor();
350     serial_read();
351     read_command();
352     movement();
353     serial_reset();
354 }
355
356 /*****
357     functions
358 *****/
359 bool check_string(const char* string)
360 {
361     for(unsigned int character = 0; character < strlen(string); character
        ++){
362         if(string[character] < 48 || string[character] > 57) {
363             return false;
364         }
365     }
366     return true;
367 }
368
369 void serial_reset()
370 {
371     category = nullptr;
372     option = nullptr;
373     value = nullptr;
374     memset(&serialRead, 0, 50);
375 }
376
377 void serial_read()
378 {
379     if (Serial.available() > 0)
380     {
381         Serial.readBytesUntil('\n', serialRead, 50);
382         category = strtok(serialRead, ":");
383         option = strtok(NULL, ":");
384         value = strtok(NULL, "\n");
385     }
386 }
387
388 void read_command()
389 {

```

```

390 if (strcmp(category, "Conf") == 0)
391 {
392     if (strcmp(option, "Delay") == 0)
393     {
394         Serial.print("ACK:Recived\n");
395         if (check_string(value))
396         {
397             Delay = atoi(value);
398         }
399     }
400     else if (strcmp(option, "SensorTimeout") == 0)
401     {
402         Serial.print("ACK:Recived\n");
403         if (check_string(value))
404         {
405             SensorTimeout = atoi(value);
406         }
407     }
408     else if (strcmp(option, "Compressor") == 0)
409     {
410         if (strcmp(value, "On") == 0)
411         {
412             Serial.print("ACK:Recived\n");
413             compressor = on;
414         }
415         else if (strcmp(value, "Off") == 0)
416         {
417             Serial.print("ACK:Recived\n");
418             compressor = off;
419         }
420         else if (strcmp(value, "Auto") == 0)
421         {
422             Serial.print("ACK:Recived\n");
423             compressor = aut;
424         }
425     }
426     else if (strcmp(option, "CompressorTimeout") == 0)
427     {
428         Serial.print("ACK:Recived\n");
429         if (check_string(value))
430         {
431             CompressorTimeout = atoi(value);

```

```

432     }
433 }
434 }if (strcmp(category, "Move") == 0 && strcmp(serialState, "ready") ==
    0)
435 {
436     if (strcmp(option, "Forward") == 0)
437     {
438         Serial.print("ACK:Recived\n");
439         if (
440             get_direction(check[1], digitalRead(legs[check[1].side][check
                [1].posistion][check[1].piston])) == false &&
441             get_direction(check[3], digitalRead(legs[check[3].side][check
                [3].posistion][check[3].piston])) == false &&
442             get_direction(check[5], digitalRead(legs[check[5].side][check
                [5].posistion][check[5].piston])) == false)
443         {
444             current_inst = (instruction *)move_forward_llr;
445             length = 13;
446             index = 0;
447         }
448         else
449         {
450             current_inst = (instruction *)move_forward_rrl;
451             length = 13;
452             index = 0;
453         }
454     }
455     else if (strcmp(option, "Back") == 0)
456     {
457         Serial.print("ACK:Recived\n");
458         //ckeck for llr
459         if (
460             get_direction(check[0], digitalRead(legs[check[0].side][check
                [0].posistion][check[0].piston])) == false &&
461             get_direction(check[2], digitalRead(legs[check[2].side][check
                [2].posistion][check[2].piston])) == false &&
462             get_direction(check[4], digitalRead(legs[check[4].side][check
                [4].posistion][check[4].piston])) == false)
463         {
464             current_inst = (instruction *)move_back_llr;
465             length = 13;
466             index = 0;

```

```

467     }
468     else
469     {
470         current_inst = (instruction *)move_back_rrl;
471         length = 13;
472         index = 0;
473     }
474 }
475 else if (strcmp(option, "Left") == 0)
476 {
477     Serial.print("ACK:Recived\n");
478     //check turn left
479     if (!(
480         get_direction(check[3], digitalRead(legs[check[3].side][
481             check[3].posistion][check[3].piston])) == true &&
482         get_direction(check[5], digitalRead(legs[check[5].side][
483             check[5].posistion][check[5].piston])) == true))
484     {
485         current_inst = (instruction *)move_turn_left;
486         length = 16;
487         index = 0;
488     }
489     else
490     {
491         current_inst = (instruction *)move_turn_left + 6;
492         length = 9;
493         index = 0;
494     }
495 }
496 else if (strcmp(option, "Right") == 0)
497 {
498     Serial.print("ACK:Recived\n");
499     //check turn right
500     if (!(
501         get_direction(check[0], digitalRead(legs[check[0].side][
502             check[0].posistion][check[0].piston])) == true &&
503         get_direction(check[2], digitalRead(legs[check[2].side][
504             check[2].posistion][check[2].piston])) == true))
505     {
506         current_inst = (instruction *)move_turn_right;
507         length = 16;
508         index = 0;

```

```

505     }
506     else
507     {
508         current_inst = (instruction *)move_turn_right + 6;
509         length = 9;
510         index = 0;
511     }
512 }
513 }
514 else if (strcmp(category, "Turn") == 0 && strcmp(serialState, "ready")
515         == 0)
516 {
517     if (strcmp(option, "Left") == 0)
518     {
519         Serial.print("ACK:Recived\n");
520         //check rotation rrl
521         if (
522             get_direction(check[1], digitalRead(legs[check[1].side][
523                 check[1].posistion][check[1].piston])) == true &&
524             get_direction(check[3], digitalRead(legs[check[3].side][
525                 check[3].posistion][check[3].piston])) == false &&
526             get_direction(check[5], digitalRead(legs[check[5].side][
527                 check[5].posistion][check[5].piston])) == false)
528         {
529             current_inst = (instruction *)move_rotation_left;
530             length = 27;
531             index = 0;
532         }
533     else
534     {
535         //turn left
536         current_inst = (instruction *)move_rotation_left + 9;
537         length = 17;
538         index = 0;
539     }
540 }
541 else if (strcmp(option, "Right") == 0)
542 {
543     Serial.print("ACK:Recived\n");
544     //check rotation llr
545     if (
546         get_direction(check[0], digitalRead(legs[check[0].side][

```

```

        check[0].posistion[check[0].piston])) == false &&
543     get_direction(check[2], digitalRead(legs[check[2].side][
        check[2].posistion[check[2].piston])) == false &&
544     get_direction(check[4], digitalRead(legs[check[4].side][
        check[4].posistion[check[4].piston])) == true)
545     {
546         current_inst = (instruction *)move_rotation_right;
547         length = 27;
548         index = 0;
549     }
550     else
551     {
552         //turn right
553         current_inst = (instruction *)move_rotation_right + 9;
554         length = 17;
555         index = 0;
556     }
557 }
558 }
559 else if (strcmp(category, "Sensor") == 0)
560 {
561     if (strcmp(option, "Distance") == 0)
562     {
563         if (strcmp(value, "Get") == 0)
564         {
565             Serial.print("ACK:Recived\n");
566             read_sensor_distance();
567         }
568     }
569     else if (strcmp(option, "Pressure"))
570     {
571         if (strcmp(value, "Get") == 0)
572         {
573             Serial.print("ACK:Recived\n");
574             read_sensor_pressure();
575         }
576     }
577 }
578 }
579
580 bool get_direction(instruction inst, bool direction)
581 {

```

```

582     if (legs[inst.side][inst.posistion][inst.piston] == legs[left][front][
        rotation])
583     {
584         return !direction;
585     }
586     else if (legs[inst.side][inst.posistion][inst.piston] == legs[left][
        back][rotation])
587     {
588         return !direction;
589     }
590     else if (legs[inst.side][inst.posistion][inst.piston] == legs[right][
        middle][rotation])
591     {
592         return !direction;
593     }
594     else
595     {
596         return direction;
597     }
598 }
599
600 void movement()
601 {
602     if (index < length)
603     {
604         instruction i = current_inst[index];
605         if ((i.timeout * Delay + time) <= millis())
606         {
607             serialState = "working";
608             compressorTime = millis();
609             digitalWrite(legs[i.side][i.posistion][i.piston], get_direction(i,
                i.direction));
610             time = millis();
611             index++;
612             if(index == length){
613                 serialState = "ready";
614                 Serial.print("ACK:Done\n");
615             }
616         }
617     }
618 }
619

```

```

620  /*****
621      sensor commands
622  *****/
623  void get_sensor()
624  {
625      if (SensorTimeout != 0 && (SensorTimeout + sensorTime <=millis()))
626      {
627          read_sensor_distance();
628          sensorTime = millis();
629      }
630      if (compressor == on)
631      {
632          digitalWrite(c, true);
633      }
634      else if (compressor == off)
635      {
636          digitalWrite(c, false);
637      }
638      else if (compressor == aut)
639      {
640          if (compressorTime > CompressorTimeout)
641          {
642              digitalWrite(c, false);
643          }
644          else if (strcmp(serialState, "working") == 0)
645          {
646              digitalWrite(c, true);
647          }
648      }
649  }
650
651  void read_sensor_distance()
652  {
653      Serial.print("Return:Distance:");
654      Serial.print(analogRead(us_l));
655      Serial.print(";");
656      Serial.print(analogRead(us_m));
657      Serial.print(";");
658      Serial.print(analogRead(us_r));
659      Serial.print("\n");
660  }
661

```

```
662 void read_sensor_pressure()  
663 {  
664     Serial.print("Return:Pressure:");  
665     Serial.print(analogRead(ps));  
666     Serial.print("\n");  
667 }
```

Anhang H

Neue Firmware

H.1 Pinbelegung

```
1  #include <Arduino.h>
2  /*****
3      right / leg [nr] / turn/elevate
4      left  / leg [nr] / turn/elevate
5  *****/
6
7  #define r_l1_t 2
8  #define r_l1_e 3
9  #define r_l2_t 4
10 #define r_l2_e 5
11 #define r_l3_t 6
12 #define r_l3_e 7
13 #define l_l1_t 8
14 #define l_l1_e 9
15 #define l_l2_t 10
16 #define l_l2_e 14
17 #define l_l3_t 15
18 #define l_l3_e 16
19
20 /*****
21     compressor / [nr]
22 *****/
23
24 #define c 18
25
26 /*****
27     pressure sensor / [nr]
```

```

28  *****/
29
30  #define ps A3
31
32  /******
33      ultrasonic sensor / [position]
34  *****/
35
36  #define us_m A5
37  #define us_l A6
38  #define us_r A7

```

H.2 Einstellungen

```

1  /******
2      timings in ms
3  *****/
4
5  unsigned long Delay = 100;
6  unsigned long SensorTimeout = 10000;
7  unsigned long CompressorTimeout = 100000;
8
9  /******
10     baudrate
11  *****/
12
13  #define baud 9600

```

H.3 Instruktionen

```

1  unsigned short move_forward_llr[13] =
2  {
3      0b1100010100000000,
4      0b1110010100000000,
5      0b1001010100000001,
6      0b1000100100000000,
7      0b1001110100000000,
8      0b1010100100000000,
9      0b1100110100000000,
10     0b1101100100000000,
11     0b1110110100001010,
12     0b1100000100000000,

```

```

13      0b1110000100000000 ,
14      0b1001000100001010 ,
15      0b0000000000000000
16 };
17 unsigned short move_forward_rrl[13] =
18 {
19      0b1000010100000000 ,
20      0b1010010100000000 ,
21      0b1101010100000001 ,
22      0b1000110100000000 ,
23      0b1001100100000000 ,
24      0b1010110100000000 ,
25      0b1100100100000000 ,
26      0b1101110100000000 ,
27      0b1110100100001010 ,
28      0b1000000100000000 ,
29      0b1010000100000000 ,
30      0b1101000100001010 ,
31      0b0000000000000000
32 };
33 unsigned short move_back_llr[13] =
34 {
35      0b1100010100000000 ,
36      0b1110010100000000 ,
37      0b1001010100000001 ,
38      0b1000110100000000 ,
39      0b1001100100000000 ,
40      0b1010110100000000 ,
41      0b1100100100000000 ,
42      0b1101110100000000 ,
43      0b1110100100001010 ,
44      0b1100000100000000 ,
45      0b1110000100000000 ,
46      0b1001000100001010 ,
47      0b0000000000000000
48 };
49 unsigned short move_back_rrl[13] =
50 {
51      0b1000010100000000 ,
52      0b1010010100000000 ,
53      0b1101010100000001 ,
54      0b1000100100000000 ,

```

```

55     0b1001110100000000 ,
56     0b1010100100000000 ,
57     0b1100110100000000 ,
58     0b1101100100000000 ,
59     0b1110110100001010 ,
60     0b1000000100000000 ,
61     0b1010000100000000 ,
62     0b1101000100001010 ,
63     0b0000000000000000
64 };
65 unsigned short move_right[15] =
66 {
67     0b1000010100000000 ,
68     0b1010010100000001 ,
69     0b1000110100000000 ,
70     0b1010110100001010 ,
71     0b1000000100000000 ,
72     0b1010000100001010 ,
73     0b1001010100000000 ,
74     0b1100010100000000 ,
75     0b1110010100000001 ,
76     0b1000100100000000 ,
77     0b1010100100001010 ,
78     0b1001000100000000 ,
79     0b1100000100000000 ,
80     0b1110000100001010 ,
81     0b0000000000000000
82 };
83 unsigned short move_left[15] =
84 {
85     0b1100010100000000 ,
86     0b1110010100000001 ,
87     0b1100110100000000 ,
88     0b1110110100001010 ,
89     0b1100000100000000 ,
90     0b1110000100001010 ,
91     0b1101010100000000 ,
92     0b1000010100000000 ,
93     0b1010010100000001 ,
94     0b1100100100000000 ,
95     0b1110100100001010 ,
96     0b1101000100000000 ,

```

```

97     0b100000001000000000 ,
98     0b101000001000001010 ,
99     0b000000000000000000
100 };
101 unsigned short turn_right[25] =
102 {
103     0b10000101000000000 ,
104     0b10100101000000000 ,
105     0b11010101000000001 ,
106     0b10001101000000000 ,
107     0b10101101000000000 ,
108     0b1101100100001010 ,
109     0b10000001000000000 ,
110     0b10100001000000000 ,
111     0b1101000100001010 ,
112     0b11000101000000000 ,
113     0b11100101000000000 ,
114     0b10010101000000001 ,
115     0b11001001000000000 ,
116     0b11101001000000000 ,
117     0b1001110100001010 ,
118     0b11000001000000000 ,
119     0b11100001000000000 ,
120     0b1001000100001010 ,
121     0b10001001000000000 ,
122     0b10011001000000000 ,
123     0b10101001000000000 ,
124     0b11001101000000000 ,
125     0b11011101000000000 ,
126     0b1110110100001010 ,
127     0b00000000000000000
128 };
129 unsigned short turn_left[25] =
130 {
131     0b11000101000000000 ,
132     0b11100101000000000 ,
133     0b10010101000000001 ,
134     0b11001101000000000 ,
135     0b11101101000000000 ,
136     0b1001100100001010 ,
137     0b11000001000000000 ,
138     0b11100001000000000 ,

```

```

139     0b1001000100001010 ,
140     0b1000010100000000 ,
141     0b1010010100000000 ,
142     0b1101010100000001 ,
143     0b1000100100000000 ,
144     0b1010100100000000 ,
145     0b1101110100001010 ,
146     0b1000000100000000 ,
147     0b1010000100000000 ,
148     0b1101000100001010 ,
149     0b1000110100000000 ,
150     0b1001110100000000 ,
151     0b1010110100000000 ,
152     0b1100100100000000 ,
153     0b1101100100000000 ,
154     0b1110100100001010 ,
155     0b0000000000000000
156 };

```

H.4 Hauptprogramm

```

1  #include <Arduino.h>
2  #include <string.h>
3
4  #include <settings.h>
5  #include <pins.h>
6  #include <instructions.h>
7
8  //#define DEBUG
9
10 /******
11 enum definition
12 ******/
13 enum side
14 {
15     left,
16     right
17 };
18 enum position
19 {
20     front,
21     middle,

```

```

22     back
23 };
24 enum piston
25 {
26     elevation,
27     rotation
28 };
29 enum mode
30 {
31     standard,
32     manual
33 };
34
35 /*****
36     var definition
37 *****/
38 #pragma region var
39 unsigned long time;
40 unsigned long sensorTime;
41 unsigned long timeout;
42 bool check = 0;
43 unsigned short inst[25];
44 int instIndex = 0;
45 int usermode;
46 int legs[2][3][2] =
47 {
48     {
49         {l_l1_e, l_l1_t},
50         {l_l2_e, l_l2_t},
51         {l_l3_e, l_l3_t}
52     },
53     {
54         {r_l1_e, r_l1_t},
55         {r_l2_e, r_l2_t},
56         {r_l3_e, r_l3_t}
57     }
58 };
59
60 struct command
61 {
62     bool resume;
63     bool side;

```

```

64     short posistion;
65     bool piston;
66     bool direction;
67     bool getsensor;
68     bool compressor;
69     unsigned long timeout;
70 };
71 #pragma endregion
72
73 /*****
74     function definition
75 *****/
76 #pragma region functions
77 void Standard();
78 void Manual();
79
80 bool CheckString(char *input);
81
82 void SerialRead(char **input);
83 void SerialCommand(char **input, unsigned short *inst);
84 void CommandMode(char **input);
85 void CommandConf(char **input);
86 void CommandMove(char **input, unsigned short *inst);
87 void CommandTurn(char **input, unsigned short *inst);
88 void CommandSensor(char **input);
89
90 void ManualSerialCommand(unsigned short value, unsigned short *inst);
91
92 void ServiceRoutine(unsigned short *inst);
93 command InstructionRead(unsigned short inst);
94 bool Movement(command inst);
95 bool GetDirection(int pin, bool direction);
96
97 void CopyInstruction(unsigned short *inst, unsigned short *destination);
98
99 void SensorDistance();
100 void ReadSensorDistance();
101
102 #ifdef DEBUG
103 void Debug(int value);
104 void Debug2();
105 #endif

```

```

106 #pragma endregion
107
108 /*****
109     main
110 *****/
111 void setup()
112 {
113     //MOS-FETs
114     pinMode(r_l1_e, OUTPUT);
115     pinMode(r_l1_t, OUTPUT);
116     pinMode(r_l2_e, OUTPUT);
117     pinMode(r_l2_t, OUTPUT);
118     pinMode(r_l3_e, OUTPUT);
119     pinMode(r_l3_t, OUTPUT);
120     pinMode(l_l1_e, OUTPUT);
121     pinMode(l_l1_t, OUTPUT);
122     pinMode(l_l2_e, OUTPUT);
123     pinMode(l_l2_t, OUTPUT);
124     pinMode(l_l3_e, OUTPUT);
125     pinMode(l_l3_t, OUTPUT);
126     //ultrasonic sensors
127     pinMode(us_l, INPUT);
128     pinMode(us_m, INPUT);
129     pinMode(us_r, INPUT);
130     //pressure sensor
131     pinMode(ps, INPUT);
132     //compressors
133     pinMode(c, OUTPUT);
134
135     Serial.begin(9600);
136     usermode = standard;
137     time = millis();
138     sensorTime = millis();
139     timeout = millis();
140 }
141
142 void loop()
143 {
144     if (usermode == standard)
145     {
146         Standard();
147     }

```

```

148     else if(usermode == manual)
149     {
150         Manual();
151     }
152 }
153
154
155 bool CheckString(char *input)
156 {
157     for (unsigned int character = 0; character < strlen(input); character
158         ++))
159     {
160         if (input[character] < 48 || input[character] > 57)
161         {
162             return false;
163         }
164     }
165     return true;
166 }
167
168 *****
169 modes
170 *****
171 void Standard()
172 {
173     char *cInput[3];
174
175     SensorDistance();
176     if (Serial.available() > 0)
177     {
178         SerialRead(cInput);
179         SerialCommand(cInput, inst);
180     }
181     ServiceRoutine(inst);
182 }
183
184 void Manual()
185 {
186     unsigned short iInput = 0;
187
188     if(Serial.available() > 0)
189     {

```

```

189     iInput = Serial.read();
190     ManualSerialCommand(iInput, inst);
191
192 }
193 ServiceRoutine(inst);
194 }
195
196 /******
197 standard serial commands
198 ******/
199 void SerialRead(char **input)
200 {
201     char temp[64];
202
203     memset(&temp, 0, 64);
204     for (int instIndex; instIndex < 3; instIndex++)
205     {
206         input[instIndex] = nullptr;
207     }
208
209     Serial.readBytesUntil('\n', temp, 6);
210     input[0] = strtok(temp, ":");
211     input[1] = strtok(NULL, ":");
212     input[2] = strtok(NULL, "\n");
213 }
214
215 void SerialCommand(char **input, unsigned short *inst)
216 {
217     if (strcmp(input[0], "Mode") == 0)
218     {
219         CommandMode(input);
220     }
221     else if (strcmp(input[0], "Conf") == 0)
222     {
223         CommandConf(input);
224     }
225     else if (strcmp(input[0], "Move") == 0 && check == 0)
226     {
227         CommandMove(input, inst);
228     }
229     else if (strcmp(input[0], "Turn") == 0 && check == 0)
230     {

```

```

231     CommandTurn(input, inst);
232 }
233 else if (strcmp(input[0], "Sensor") == 0)
234 {
235     CommandSensor(input);
236 }
237 }
238
239 /*****
240     standard commands
241 *****/
242 void CommandMode(char **input)
243 {
244     if (strcmp(input[1], "Eib1") == 0)
245     {
246         usermode = manual;
247         Serial.print("ACK:Received\n");
248     }
249 }
250
251 void CommandConf(char **input)
252 {
253     if (strcmp(input[1], "Delay") == 0)
254     {
255         Serial.print("ACK:Received\n");
256         if (CheckString(input[2]))
257         {
258             Delay = atoi(input[2]);
259         }
260     }
261     else if (strcmp(input[1], "SensorTimeout") == 0)
262     {
263         Serial.print("ACK:Received\n");
264         if (CheckString(input[2]))
265         {
266             SensorTimeout = atoi(input[2]);
267         }
268     }
269     else if (strcmp(input[1], "Compressor") == 0)
270     {
271         unsigned long oldCompressorTimeout;
272         if (strcmp(input[2], "On") == 0)

```

```

273     {
274         Serial.print("ACK:Received\n");
275         digitalWrite(c, true);
276         if (CompressorTimeout != 0)
277         {
278             oldCompressorTimeout = CompressorTimeout;
279             CompressorTimeout = 0;
280         }
281     }
282     else if (strcmp(input[2], "Off") == 0)
283     {
284         Serial.print("ACK:Received\n");
285         digitalWrite(c, false);
286         if (CompressorTimeout != 0)
287         {
288             oldCompressorTimeout = CompressorTimeout;
289             CompressorTimeout = 0;
290         }
291     }
292     else if (strcmp(input[2], "Auto") == 0)
293     {
294         Serial.print("ACK:Received\n");
295         CompressorTimeout = oldCompressorTimeout;
296     }
297 }
298 else if (strcmp(input[1], "CompressorTimeout") == 0)
299 {
300     Serial.print("ACK:Received\n");
301     if (CheckString(input[2]))
302     {
303         CompressorTimeout = atoi(input[2]);
304     }
305 }
306 }
307
308 void CommandMove(char **input, unsigned short *inst)
309 {
310     if (strcmp(input[1], "Forward") == 0)
311     {
312         Serial.print("ACK:Received\n");
313
314         if (GetDirection(l_12_t, digitalRead(l_12_t)) == 0 && GetDirection(

```

```

        r_l1_t, digitalRead(r_l1_t)) == 0 && GetDirection(r_l3_t,
        digitalRead(r_l3_t)) == 0)
315     {
316         CopyInstruction(move_forward_llr, inst);
317     }
318     else
319     {
320         CopyInstruction(move_forward_rrl, inst);
321     }
322 }
323 else if (strcmp(input[1], "Back") == 0)
324 {
325     Serial.print("ACK:Received\n");
326     if (GetDirection(l_l1_t, digitalRead(l_l1_t)) == 0 && GetDirection(
        l_l3_t, digitalRead(l_l3_t)) == 0 && GetDirection(r_l2_t,
        digitalRead(r_l2_t)) == 0)
327     {
328         CopyInstruction(move_back_llr, inst);
329     }
330     else
331     {
332         CopyInstruction(move_back_rrl, inst);
333     }
334 }
335 else if (strcmp(input[1], "Left") == 0)
336 {
337     Serial.print("ACK:Received\n");
338     if (GetDirection(r_l1_t, digitalRead(r_l1_t)) == 0 || GetDirection(
        r_l3_t, digitalRead(r_l3_t)) == 0)
339     {
340         CopyInstruction(move_left, inst);
341     }
342     else
343     {
344         CopyInstruction(move_left + 6, inst);
345     }
346 }
347 else if (strcmp(input[1], "Right") == 0)
348 {
349     Serial.print("ACK:Received\n");
350     if (GetDirection(l_l1_t, digitalRead(l_l1_t)) == 0 || GetDirection(
        l_l3_t, digitalRead(l_l3_t)) == 0)

```

```

351     {
352         CopyInstruction(move_right, inst);
353     }
354     else
355     {
356         CopyInstruction(move_right + 6, inst);
357     }
358 }
359 }
360
361 void CommandTurn(char **input, unsigned short *inst)
362 {
363     if (strcmp(input[1], "Left") == 0)
364     {
365         Serial.print("ACK:Received\n");
366         if (GetDirection(l_l2_t, true) == 0 || GetDirection(r_l1_t, false)
            == 0 || GetDirection(r_l3_t, false) == 0)
367         {
368             CopyInstruction(turn_left, inst);
369         }
370         else
371         {
372             CopyInstruction(turn_left + 9, inst);
373         }
374     }
375     else if (strcmp(input[1], "Right") == 0)
376     {
377         Serial.print("ACK:Received\n");
378         if (GetDirection(l_l1_t, false) == 0 || GetDirection(l_l3_t, false)
            == 0 || GetDirection(r_l2_t, true) == 0)
379         {
380             CopyInstruction(turn_right, inst);
381         }
382         else
383         {
384             CopyInstruction(turn_right + 9, inst);
385         }
386     }
387 }
388
389 void CommandSensor(char **input)
390 {

```

```

391     if (strcmp(input[1], "Distance") == 0)
392     {
393         if (strcmp(input[2], "Get") == 0)
394         {
395             Serial.print("ACK:Received\n");
396             ReadSensorDistance();
397         }
398     }
399 }
400
401 *****
402 manual serial commands
403 *****
404 void ManualSerialCommand(unsigned short value, unsigned short *inst)
405 {
406     if ((value & 0x80) == 0)
407     {
408         usermode = standard;
409     }
410     else
411     {
412         inst[0] = value << 8;
413         inst[1] = 0;
414     }
415 }
416
417 *****
418 control
419 *****
420 void ServiceRoutine(unsigned short *inst)
421 {
422     command current_inst = InstructionRead(inst[instIndex]);
423
424     if (current_inst.resume != 1)
425     {
426         instIndex = 0;
427         inst[instIndex] = 0;
428
429         if (usermode == standard)
430         {
431             if (CompressorTimeout != 0 && time > CompressorTimeout)
432             {

```

```

433         digitalWrite(c, false);
434     }
435     if (check == 1)
436     {
437         Serial.print("ACK:Done\n");
438     }
439 }
440
441     check = 0;
442 }
443 else
444 {
445     check = Movement(current_inst);
446     if (check == true) {instIndex++;}
447     if (current_inst.getsensor == 1) {ReadSensorDistance();}
448     digitalWrite(c, current_inst.compressor);
449     check = 1;
450 }
451 }
452
453 command InstructionRead(unsigned short inst)
454 {
455     command out;
456     out.resume = (inst & 0x8000) >> 15;
457     out.side = (inst & 0x4000) >> 14;
458     out.posistion = (inst & 0x3000) >> 12;
459     out.piston = (inst & 0x0800) >> 11;
460     out.direction = (inst & 0x0400) >> 10;
461     out.getsensor = (inst & 0x0200) >> 9;
462     out.compressor = (inst & 0x0100) >> 8;
463     out.timeout = inst & 0x00FF;
464     return out;
465 }
466
467 bool Movement(command inst)
468 {
469     if (timeout * Delay + time <= millis())
470     {
471         digitalWrite(legs[inst.side][inst.posistion][inst.piston],
472             GetDirection(legs[inst.side][inst.posistion][inst.piston], inst.
473                 direction));

```

```

473     #ifdef DEBUG
474     Debug(legs[inst.side][inst.posistion][inst.piston]);
475     Serial.print("┐->┐");
476     Serial.print(GetDirection(legs[inst.side][inst.posistion][inst.
        piston], inst.direction));
477     Serial.print("\n");
478     #endif
479
480     time = millis();
481     timeout = inst.timeout;
482     return true;
483 }
484 return false;
485 }
486
487 bool GetDirection(int pin, bool direction)
488 {
489     if (pin == l_l1_t)
490     {
491         return !direction;
492     }
493     else if (pin == l_l3_t)
494     {
495         return !direction;
496     }
497     else if (pin == r_l2_t)
498     {
499         return !direction;
500     }
501     else
502     {
503         return direction;
504     }
505 }
506
507 void CopyInstruction(unsigned short *inst, unsigned short *destination)
508 {
509     int i = 0;
510     while (inst[i])
511     {
512         destination[i] = inst[i];
513         i++;

```

```

514     }
515     destination[i] = '\0';
516 }
517
518 /*****
519     sensor commands
520 *****/
521 void SensorDistance()
522 {
523     if (SensorTimeout != 0 && (SensorTimeout + sensorTime <= millis()))
524     {
525         ReadSensorDistance();
526         sensorTime = millis();
527     }
528 }
529
530 void ReadSensorDistance()
531 {
532     Serial.print("Return:Distance:");
533     Serial.print(analogRead(us_l));
534     Serial.print(";");
535     Serial.print(analogRead(us_m));
536     Serial.print(";");
537     Serial.print(analogRead(us_r));
538     Serial.print("\n");
539 }
540
541
542
543
544 #ifdef DEBUG
545 void Debug(int value)
546 {
547     char *test[17] = {
548         "x",
549         "x",
550         "r_l1_t",
551         "r_l1_e",
552         "r_l2_t",
553         "r_l2_e",
554         "r_l3_t",
555         "r_l3_e",

```

```

556     "l_11_t",
557     "l_11_e",
558     "l_12_t",
559     "x",
560     "x",
561     "x",
562     "l_12_e",
563     "l_13_t",
564     "l_13_e"
565 };
566     Serial.print(test[value]);
567 }
568
569 void Debug2()
570 {
571     Serial.print("r_11_t->");
572     Serial.println(GetDirection(digitalRead(r_11_t), false));
573     Serial.print("r_12_t->");
574     Serial.println(GetDirection(digitalRead(r_12_t), false));
575     Serial.print("r_13_t->");
576     Serial.println(GetDirection(digitalRead(r_13_t), false));
577     Serial.print("l_11_t->");
578     Serial.println(GetDirection(digitalRead(l_11_t), false));
579     Serial.print("l_12_t->");
580     Serial.println(GetDirection(digitalRead(l_12_t), false));
581     Serial.print("l_13_t->");
582     Serial.println(GetDirection(digitalRead(l_13_t), false));
583 }
584 #endif

```

Anhang I

Programm zur Instruktionswandlung

I.1 Hauptprogramm

```
1  #include <stdio.h>
2
3  enum side
4  {
5      left,
6      right
7  };
8  enum position
9  {
10     front,
11     middle,
12     back
13 };
14 enum piston
15 {
16     elevation,
17     rotation
18 };
19
20 struct instruction
21 {
22     int side;
23     int posistion;
24     int piston;
25     bool direction;
26     unsigned long timeout;
27 };
```

```

28
29 *****
30 movement instructions
31 *****/
32 instruction move_forward_llr[12] = {
33     //elevate
34     {right, front, elevation, true, 0},
35     {right, back, elevation, true, 0},
36     {left, middle, elevation, true, 1},
37     //rotate
38     {left, front, rotation, false, 0},
39     {left, middle, rotation, true, 0},
40     {left, back, rotation, false, 0},
41     {right, front, rotation, true, 0},
42     {right, middle, rotation, false, 0},
43     {right, back, rotation, true, 10},
44     //elevate
45     {right, front, elevation, false, 0},
46     {right, back, elevation, false, 0},
47     {left, middle, elevation, false, 10} };
48 instruction move_forward_rrl[12] = {
49     //elevate
50     {left, front, elevation, true, 0},
51     {left, back, elevation, true, 0},
52     {right, middle, elevation, true, 1},
53     //rotate
54     {left, front, rotation, true, 0},
55     {left, middle, rotation, false, 0},
56     {left, back, rotation, true, 0},
57     {right, front, rotation, false, 0},
58     {right, middle, rotation, true, 0},
59     {right, back, rotation, false, 10},
60     //elevate
61     {left, front, elevation, false, 0},
62     {left, back, elevation, false, 0},
63     {right, middle, elevation, false, 10} };
64 instruction move_back_llr[12] = {
65     //elevate
66     {right, front, elevation, true, 0},
67     {right, back, elevation, true, 0},
68     {left, middle, elevation, true, 1},
69     //rotate

```

```

70     {left, front, rotation, true, 0},
71     {left, middle, rotation, false, 0},
72     {left, back, rotation, true, 0},
73     {right, front, rotation, false, 0},
74     {right, middle, rotation, true, 0},
75     {right, back, rotation, false, 10},
76     //elevate
77     {right, front, elevation, false, 0},
78     {right, back, elevation, false, 0},
79     {left, middle, elevation, false, 10} };
80 instruction move_back_rrl[12] = {
81     //elevate
82     {left, front, elevation, true, 0},
83     {left, back, elevation, true, 0},
84     {right, middle, elevation, true, 1},
85     //rotate
86     {left, front, rotation, false, 0},
87     {left, middle, rotation, true, 0},
88     {left, back, rotation, false, 0},
89     {right, front, rotation, true, 0},
90     {right, middle, rotation, false, 0},
91     {right, back, rotation, true, 10},
92     //elevate
93     {left, front, elevation, false, 0},
94     {left, back, elevation, false, 0},
95     {right, middle, elevation, false, 10} };
96 instruction move_turn_right[14] = {
97     //SETUP
98     //elevate
99     {left, front, elevation, true, 0},
100    {left, back, elevation, true, 1},
101    //rotate
102    {left, front, rotation, true, 0},
103    {left, back, rotation, true, 10},
104    //elevate
105    {left, front, elevation, false, 0},
106    {left, back, elevation, false, 10},
107    //START
108    //elevation
109    {left, middle, elevation, true, 0},
110    {right, front, elevation, true, 0},
111    {right, back, elevation, true, 1},

```

```

112 //rotate
113 {left, front, rotation, false, 0},
114 {left, back, rotation, false, 10},
115 //elevation
116 {left, middle, elevation, false, 0},
117 {right, front, elevation, false, 0},
118 {right, back, elevation, false, 10} };
119 instruction move_turn_left[14] = {
120 //SETUP
121 //elevate
122 {right, front, elevation, true, 0},
123 {right, back, elevation, true, 1},
124 //rotate
125 {right, front, rotation, true, 0},
126 {right, back, rotation, true, 10},
127 //elevate
128 {right, front, elevation, false, 0},
129 {right, back, elevation, false, 10},
130 //START
131 //elevation
132 {right, middle, elevation, true, 0},
133 {left, front, elevation, true, 0},
134 {left, back, elevation, true, 1},
135 //rotate
136 {right, front, rotation, false, 0},
137 {right, back, rotation, false, 10},
138 //elevation
139 {right, middle, elevation, false, 0},
140 {left, front, elevation, false, 0},
141 {left, back, elevation, false, 10} };
142 instruction move_rotation_right[24] = {
143 //SETUP llr
144 //elevation
145 {left, front, elevation, true, 0},
146 {left, back, elevation, true, 0},
147 {right, middle, elevation, true, 1},
148 //rotate
149 {left, front, rotation, true, 0},
150 {left, back, rotation, true, 0},
151 {right, middle, rotation, false, 10},
152 //elevation
153 {left, front, elevation, false, 0},

```

```

154     {left, back, elevation, false, 0},
155     {right, middle, elevation, false, 10},
156     //SETUP rrl
157     //elevation
158     {right, front, elevation, true, 0},
159     {right, back, elevation, true, 0},
160     {left, middle, elevation, true, 1},
161     //rotate
162     {right, front, rotation, false, 0},
163     {right, back, rotation, false, 0},
164     {left, middle, rotation, true, 10},
165     //elevation
166     {right, front, elevation, false, 0},
167     {right, back, elevation, false, 0},
168     {left, middle, elevation, false, 10},
169     //START
170     //rotation
171     {left, front, rotation, false, 0},
172     {left, middle, rotation, false, 0},
173     {left, back, rotation, false, 0},
174     {right, front, rotation, true, 0},
175     {right, middle, rotation, true, 0},
176     {right, back, rotation, true, 10} };
177 instruction move_rotation_left[24] = {
178     //SETUP rrl
179     //elevation
180     {right, front, elevation, true, 0},
181     {right, back, elevation, true, 0},
182     {left, middle, elevation, true, 1},
183     //rotate
184     {right, front, rotation, true, 0},
185     {right, back, rotation, true, 0},
186     {left, middle, rotation, false, 10},
187     //elevation
188     {right, front, elevation, false, 0},
189     {right, back, elevation, false, 0},
190     {left, middle, elevation, false, 10},
191     //SETUP llr
192     //elevation
193     {left, front, elevation, true, 0},
194     {left, back, elevation, true, 0},
195     {right, middle, elevation, true, 1},

```

```

196 //rotate
197 {left, front, rotation, false, 0},
198 {left, back, rotation, false, 0},
199 {right, middle, rotation, true, 10},
200 //elevation
201 {left, front, elevation, false, 0},
202 {left, back, elevation, false, 0},
203 {right, middle, elevation, false, 10},
204 //START
205 //rotation
206 {left, front, rotation, true, 0},
207 {left, middle, rotation, true, 0},
208 {left, back, rotation, true, 0},
209 {right, front, rotation, false, 0},
210 {right, middle, rotation, false, 0},
211 {right, back, rotation, false, 10} };
212
213 unsigned short bitMagic(instruction i);
214 void dec2bin(unsigned short num);
215 void write(instruction *inst, int length);
216
217 int main() {
218
219     printf("unsigned_short_move_forward_llr [13] _=_\n{\n");
220     write((instruction *)move_forward_llr, 12);
221     printf("};\n");
222
223     printf("unsigned_short_move_forward_rrl [13] _=_\n{\n");
224     write((instruction*)move_forward_rrl, 12);
225     printf("};\n");
226
227     printf("unsigned_short_move_back_llr [13] _=_\n{\n");
228     write((instruction*)move_back_llr, 12);
229     printf("};\n");
230
231     printf("unsigned_short_move_back_rrl [13] _=_\n{\n");
232     write((instruction*)move_back_rrl, 12);
233     printf("};\n");
234
235     printf("unsigned_short_move_turn_right [15] _=_\n{\n");
236     write((instruction*)move_turn_right, 14);
237     printf("};\n");

```

```

238
239     printf("unsigned_short_move_turn_left[15]_=_\n{\n");
240     write((instruction*)move_turn_left, 14);
241     printf("};\n");
242
243     printf("unsigned_short_move_rotation_right[25]_=_\n{\n");
244     write((instruction*)move_rotation_right, 24);
245     printf("};\n");
246
247     printf("unsigned_short_move_rotation_left[25]_=_\n{\n");
248     write((instruction*)move_rotation_left, 24);
249     printf("};\n");
250
251     return 0;
252 }
253
254 void write(instruction *inst, int length) {
255     for (int i = 0; i < length; i++) {
256         printf("___");
257         dec2bin(bitMagic(inst[i]));
258         printf(",\n");
259     }
260     printf("___0b0000000000000000\n");
261 }
262
263 unsigned short bitMagic(instruction i){
264     unsigned short out = 1;
265     out = out << 15;
266     out += i.side << 14;
267     out += i.posistion << 12;
268     out += i.piston << 11;
269     out += i.direction << 10;
270     out += 0 << 9; // sensor request
271     out += 1 << 8; // motor
272     out += i.timeout;
273     return out;
274 }
275
276 void dec2bin(unsigned short num) {
277     unsigned short mask = 0b1000000000000000;
278
279     printf("0b");

```

```
280     for (int i = 0; i < 16; i++) {
281         printf("%d", (num & mask ? 1:0));
282         mask = mask >> 1;
283     }
284 }
```

Literaturverzeichnis

- [1] Arduino. Arduino as ISP and Arduino Bootloaders. <https://www.arduino.cc/en/Tutorial/BuiltInExamples/ArduinoISP>, 2018.
- [2] Arturo Guadalupi. NanoV3.3. https://content.arduino.cc/assets/NanoV3.3_sch.pdf, 2019. Arduino Nano Schaltplan.
- [3] Diodes Incorporated. 1N4148W. <https://www.diodes.com/assets/Datasheets/ds30086.pdf>, 2017.
- [4] DI Thomas Eibel. Die BULME Skulptur . https://lehrer.bulme.at/~eb/bulme_kultur/archiv/2000_01/bulme_skulptur.html.
- [5] Honeywell. Series 947 analogue. <https://datasheet.ciiva.com/18452/947-f4v-2d-1c0-180e-18452627.pdf>.
- [6] jimbojr. hat-board-mechanical. <https://github.com/raspberrypi/hats/blob/master/hat-board-mechanical.pdf>, 2014.
- [7] KNF. NPK 09 Series. https://knf.com/fileadmin/Global_files/Downloads/Product_OEM_Process/Datasheets/KNF_Datasheet_NPK_09_12-2020_web.pdf.
- [8] LITE-ON. Ltw-150tk. <https://docs.rs-online.com/fd34/0900766b80e28d04.pdf>. Datenblatt der weißen LED.
- [9] Martin Rowan. Raspberry Pi 4 Cases, Temperature and CPU Throttling Under Load. <https://www.martinrowan.co.uk/2019/09/raspberry-pi-4-cases-temperature-and-cpu-throttling-under-load/#conclusion>, 2019.
- [10] Mean Well. PSD-15 Datasheet. <https://www.meanwell.com/Upload/PDF/PSD-15/PSD-15-SPEC.PDF>, 2020.
- [11] Microchip Technology. megaAVR® Data Sheet. <https://ww1.microchip.com/downloads/en/DeviceDoc/>

- ATmega48A-PA-88A-PA-168A-PA-328-P-DS-DS40002061B.pdf, 2019. ATmega328P Datenblatt.
- [12] Molex. MICRO-FIT (3.0) DUAL ROW RIGHT ANGLE THRU HOLE HEADER ASSEMBLY. https://www.molex.com/pdm_docs/sd/430450600_sd.pdf, 2000. Molex Mirco-Fit 3.0 Buchse Datenblatt.
- [13] Raspberry Pi Foundation. Raspberry Pi 4 Tech Specs. <https://www.raspberrypi.org/products/raspberry-pi-4-model-b/specifications/>.
- [14] Raspberry Pi Foundation. Raspberry Pi Operating Systems. <https://www.raspberrypi.org/software/operating-systems/>.
- [15] Raspberry Pi Foundation. Raspbian About. <http://raspbian.org/RaspbianAbout>.
- [16] Raspberry Pi Foundation. UART configuration. <https://www.raspberrypi.org/documentation/configuration/uart.md>, 2017.
- [17] Yakov Rekhter and Tony Li. The Secure Shell (SSH) Protocol Architecture. RFC 4251, IETF, January 2006.
- [18] RS. Datasheet Sealed Lead-Acid Battery 150-1560. <https://docs.rs-online.com/cd87/0900766b8163f4c1.pdf>.
- [19] Science Direct. Insect Walking Animation. <https://mae.princeton.edu/sites/default/files/AyaliBBC-FD-GH-C0IS15.pdf>, 2015.
- [20] Tony DiCola. Raspberry Pi Analog to Digital Converters. <https://learn.adafruit.com/raspberry-pi-analog-to-digital-converters>, 2016.
- [21] Vishay Intertechnology. SI2300DS. <https://www.vishay.com/docs/65701/si2300ds.pdf>, 2010.
- [22] Bruno Wildbach. [Beschreibung der Skulptur]. <http://offsite.kulturserver-graz.at/werke/1680>, 2001.