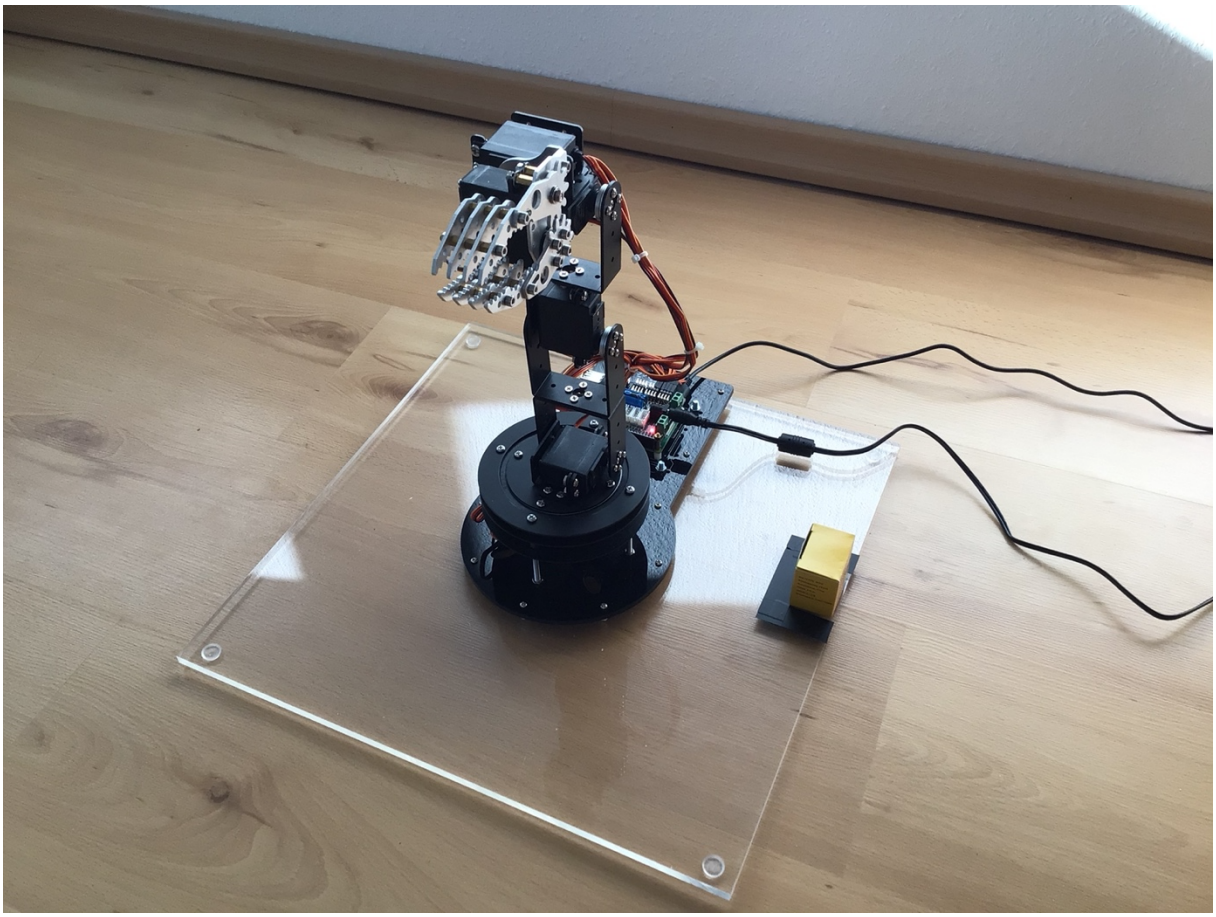
Python-Programmierung zur Fernsteuerung eines Roboters



Florian Schmitz

Höhere Berufsfachschule, BBS1	Klasse HBF IT 19a
Schuljahr 2020/2021	
Projektzeitraum 01.02.2021 – 02.03.2021	Abgabetermin 02.03.2021 13 Uhr

Gesamtinhaltsverzeichnis

Vorwort

Als Thema für die praktische Abschlussarbeit habe ich mir die Programmierung der Steuerung eines Roboter-Arms ausgesucht.

Einer meiner Beweggründe dafür ist mein Interesse an der zunehmenden Digitalisierung in allen Lebensbereichen. Mehrfach habe ich im Internet den Begriff „Internet der Dinge“ gesehen und darüber gelesen.

Das Thema meiner Abschlussarbeit hat es mir erlaubt die Software-Entwicklung mit Python, die mir sehr viel Spaß macht und mit der ich mich auch in meiner Freizeit beschäftige, mit der Steuerung eines Roboters und den damit zusammenhängenden Aufgaben im Bereich Mechanik und Elektronik zu kombinieren.

Außerdem kann ich bei dieser Aufgabenstellung meine Kenntnisse über die Entwicklung von Smartphone Apps einbringen. Diese konnte ich mir zuvor in meiner Freizeit und im Rahmen meiner Praktika aneignen. Sie haben mir es ermöglicht, bei der Abschlussarbeit eine Smartphone App zur Bedienung des Roboters zu entwickeln.

Projektbezeichnung	Python-Programmierung zur Fernsteuerung eines Roboters
Projektleiter	Florian Schmitz
Erstellt am	01.02.2021
Letzte Änderung am	01.03.2021
Aktuelle Version	1.3

Lastenheft

Inhaltsverzeichnis

1. Zielbestimmung

Ziel des Projektes ist die Montage der mechatronischen Komponenten (Bausatz[1]), Installation und Konfiguration des Raspberry Pi, die Entwicklung einer Steuerungs- (Backend) und Bediensoftware (Frontend) und Inbetriebnahme eines 6-achsigen Roboter-Arms.

Als Antriebe kommen Servomotoren zum Einsatz, die mit Hilfe eines Raspberry Pi mit MOTOPI[2] Elektronik auf Basis von Pulsweitenmodulation gesteuert werden sollen.

Die Steuerungssoftware soll als Service im Backend auf dem Raspberry Pi laufen und mit Python entwickelt werden. Ausgangspunkt hierfür ist eine vom Hersteller der MOTOPI Elektronik bereitgestellte Python Library, die als Software Treiber eingesetzt wird. Die Steuerungssoftware soll als Service realisiert werden, die auf dem Raspberry ohne Tastatur und Monitor lauffähig ist („headless“).

Über eine zu implementierende RESTful API[3] soll eine zu entwickelnde Smartphone App als Frontend zur Bedienung des Roboters angebunden werden. Die Basis hierfür ist die Programmiersprache Dart und das Framework Flutter[4] von Google.

2. Produkteinsatz

Der Roboter soll als Demonstrationsobjekt für die Entwicklung und Test von Steuerungssoftware und Benutzerschnittstelle dienen, um das Zusammenspiel von Mechanik, Elektronik und Software zu erlernen.

[1] JOY-IT ROBO02 https://joy-it.net/de/products/Robot02
[2] JOY-IT MOTOPI https://joy-it.net/de/products/RB-Moto3
[3] RESTful API https://restfulapi.net/
[4] Google Flutter https://flutter.dev/

3. Funktionale Anforderungen

ID	Prio	Beschreibung
1	Muss	Alle Servomotoren können durch Raspberry Pi und der Moto-Pi Aufsatzplatine mit Hilfe von Pulsweitenmodulation (PWM) einzeln angesteuert werden. Laut Datenblatt des Servomotors, muss ein 50 Hz Rechtecksignal mit einer Pulslänge im Bereich 500 bis 2500 Mikrosekunden (µs) generiert werden. Der Servo-Typ hat einen Drehbereich von bis zu 180°.
2	Muss	Der erlaubte Drehbereich der Servomotoren kann einzeln eingestellt werden, um die mechanischen Vorgaben zu berücksichtigen. Um die Servomotoren testen und einstellen zu können, soll ein Servo-Testprogramm entwickelt werden.
3	Muss	Die Neutralposition kann man bei jedem Servomotor konfigurieren, um eine individuelle Start- bzw. Warteposition des Roboters zu ermöglichen. Die Neutralposition wird mit einem PWM-Signal von 1500 µs erzeugt.
4	Muss	Zur Steuerung Servomotor-Position wird die Pulsweite auf einen konfigurierbaren Positionswertebereich umgerechnet. Der Wertebereich wird auf die Steuerungselemente der App angepasst (z.B. von -1000 bis +1000).
5	Muss	Mit Hilfe von „Slidern“ als Steuerungselemente (Controller) sollen die Servo-Positionen verändert werden können.
6	Muss	Die erforderlichen Funktionen für die Steuerung des Roboter Arms können per REST-API aufgerufen werden.
7	Muss	Die Smartphone App nutzt die REST Schnittstelle, um mit dem Backend zu kommunizieren. Hierbei ist das Smartphone der Client und der REST API-Server das Backend.
8	Kann	Die Servomotoren können mit einem Button in der App auf ihre Neutralposition zurückgesetzt werden.
9	Kann	Der Roboter hat einen einprogrammierten Demo-Bewegungsablauf, der vom Anwender per App ausgeführt werden kann. Zum Beispiel soll der Roboter per Knopfdruck einen kleinen Gegenstand hochheben und zu einer anderen Position transportieren.

4. Nichtfunktionale Anforderungen

ID	Prio.	Beschreibung
1	Kann	Die wichtigsten Funktionen werden durch automatisierte Unit Tests getestet.
2	Muss	Es gibt eine Konfigurationsdatei, in der alle änderbaren Einstellungen vorgenommen werden können.
3	Kann	Die API ist dokumentiert, zum Beispiel mit Swagger UI[5].
4	Kann	In der App kann der User die IP-Adresse des Servers auswählen.
5	Muss	Connection Probleme werden abgefangen und der User wird durch einen Alert Dialog in der App gewarnt.
6	Muss	Das Smartphone kommuniziert entweder über WLAN oder Mobilen Hotspot mit dem Backend.
7	Muss	Durch eine Konfigurationsdatei merkt sich das Backend die erlaubten Drehbereiche, damit die Servos nicht an die mechanischen Grenzen fahren.
8	Muss	Die App soll auf einem Galaxy A21s Smartphone laufen (Android Betriebssystem).
9	Muss	Das Backend läuft auf einem Raspberry Pi 4 Model B mit 4 GB RAM. Betriebssystem: Raspbian, Version 10.
10	Kann	Das Backend startet bei einem Start oder Neustart automatisch (Service).

5. Abnahmekriterien

Zur Abnahme des Projekts sollen folgende Funktionen per Foto- oder Video-Dokumentation erläutert oder per Live-Vorführung demonstriert werden:

- Bewegung des Roboters in die Warte-/Neutralposition
- Bewegung aller sechs Roboterachsen mit Hilfe der Smartphone App
- Starten und Durchführung des Demo-Bewegungsablaufs

[5] https://swagger.io/tools/swagger-ui/

Projektbezeichnung	Python-Programmierung zur Fernsteuerung eines Roboters
Projektleiter	Florian Schmitz
Erstellt am	26.02.2021
Letzte Änderung am	01.03.2021
Aktuelle Version	1.1

Pflichtenheft

Inhaltsverzeichnis

1 Arbeitspaketdefinition

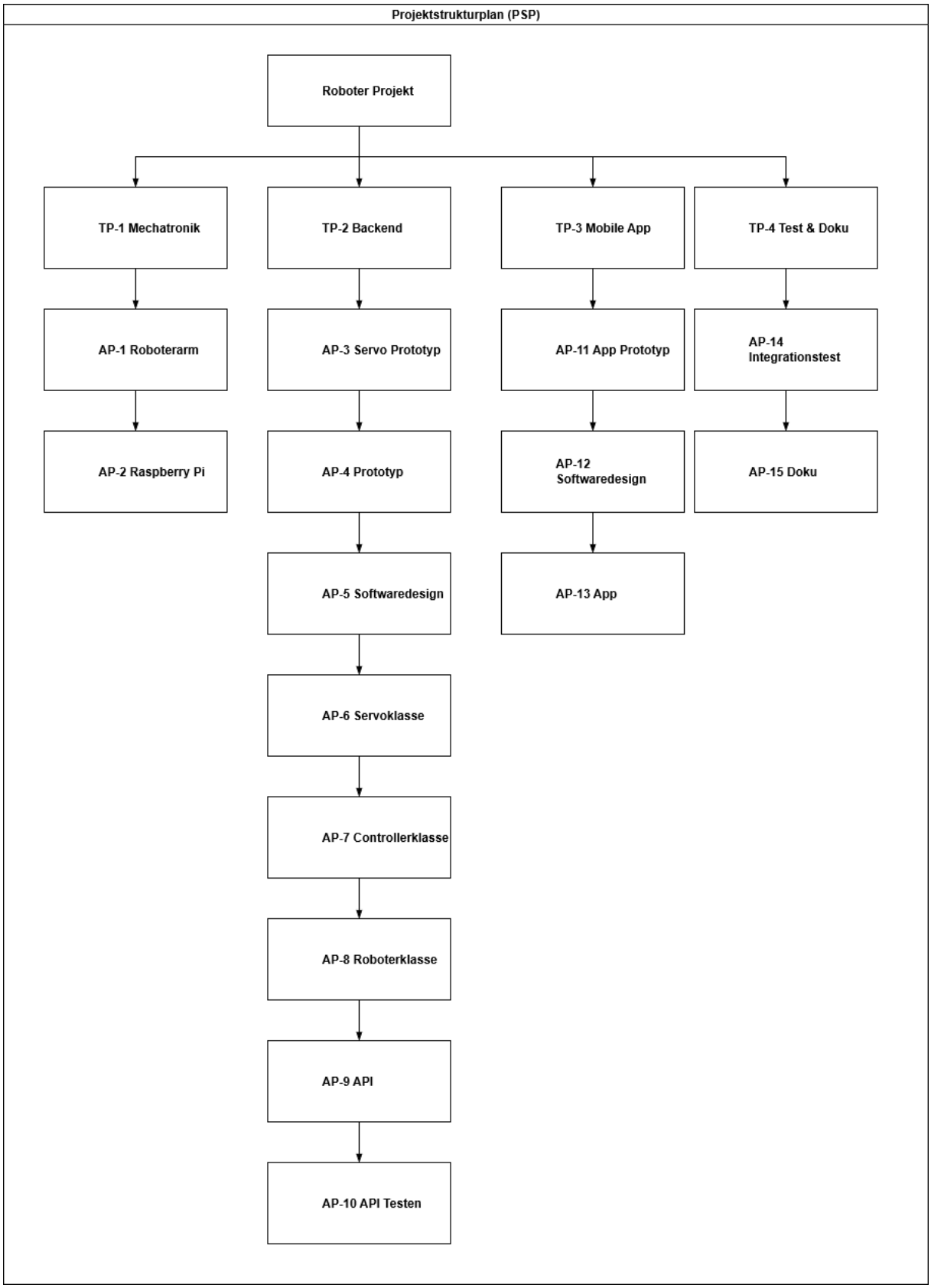
AP-Nr.	Bezeichnung	Kurzbeschreibung
1	Roboterarm zusammenbauen	Test der Servomotoren mithilfe einer Servo-Tester Hardware. Einstellung der Servo-Arme auf Neutralposition, Montage des Roboters, Befestigung des Raspberry Pi und Verkabelung der Servomotoren.
2	Raspberry Pi einrichten	Download und Speicherung des Raspbian Betriebssystems auf SD-Karte, Temporäres Anschließen von Tastatur und Maus, Booten und Konfiguration des Raspberry. Installation und Konfiguration zusätzlich benötigter Softwarekomponenten, wie zum Beispiel die Adafruit Bibliothek zur Ansteuerung des PWM-Controllers. Aktivierung des SSH Servers für Remote Zugriff.
3	Servo Software-Prototyp entwickeln	Zur Einarbeitung und Sicherstellung der Machbarkeit soll ein kleines Python Testprogramm entwickelt werden, das einzelne Servos ansteuern kann (Servo-Tester Software). Zuvor Entwicklungsumgebung Visual Studio Code[1] einrichten, so dass ein Remote Coding und Debugging auf dem Raspberry durchgeführt werden kann.
4	Api Software-Prototyp entwickeln	Zur Einarbeitung und Sicherstellung der Machbarkeit soll ein kleines Python Testprogramm entwickelt werden, das eine RESTful API[2] bereitstellt. API mit Hilfe einer API-Client Software testen (z.B. mit Postman[3]).
5	Softwaredesign Backend	Objektorientiertes Software-Design erarbeiten und mithilfe von UML Diagrammen[4] und dem Tool Draw.io[5] dokumentieren.
6	Servoklasse entwickeln	Entwicklung einer Python Klasse mit der Funktionalität zur Ansteuerung eines Servomotors und der dazugehörigen automatisierten Softwaretests (Unit Tests[6]).

[1] https://code.visualstudio.com/
[2] https://restfulapi.net/
[3] https://www.postman.com/product/api-client/
[4] https://www.ionos.de/digitalguide/websites/web-entwicklung/klassendiagramme-mit-uml-erstellen/
[5] https://app.diagrams.net/
[6] https://docs.python.org/3/library/unittest.html

7	Controllerklasse entwickeln	Entwicklung einer Python Klasse mit der Funktionalität zur Verarbeitung von Steuerbefehlen (Slider auf der Smartphone App).
8	Roboterklasse entwickeln	Entwicklung einer Python Klasse, die alle benötigten Softwarekomponenten miteinander verbindet.
9	Rest Api Server entwickeln	Entwicklung einer Python Klasse mit der Funktionalität die Service-Endpoints für GET und PUT bereitzustellen.
10	Api mit Postman testen	Manuelles Testen der API-Funktionen mithilfe des Tools Postman[7]
11	App Prototyp entwickeln	Zur Einarbeitung und Sicherstellung der Machbarkeit wird ein App Prototyp für Smartphones mit Android Betriebssystem mithilfe der Programmiersprache Dart und dem Flutter Framework entwickelt. Zuvor Entwicklungsumgebung mit Visual Studio Code für Dart[8] und Flutter[9] einrichten.
12	Softwaredesign Frontend	Erstellung des User Interface Designs (Mockup) mithilfe des Tools Draw.io.
13	App entwickeln	Entwicklung der App zur Steuerung des Roboter-Arms, die über die REST-API mit dem Backend kommuniziert.
14	Integrationstest	Testen des Zusammenspiels aller Systemkomponenten und des Gesamtablaufs.
15	Dokumentation	Fertigstellung aller geforderten Dokumente.

[7] https://www.postman.com/product/api-client/
[8] http://dartapps.de/
[9] Google Flutter https://flutter.dev/

2 Projektstrukturplan



3 Projektablaufplan

Das Projekt wird in vier Phasen gegliedert:

- Phase 1: Mechatronik
- Phase 2: Backend
- Phase 3: Frontend
- Phase 4: Integration

3.1 Phase 1: Teilprojekt Mechatronik

Ergebnis: Roboter-Arm ist mechanisch funktionsfähig, Elektronik ist verkabelt und konfiguriert.

3.2 Phase 2: Teilprojekt Backend

Ergebnis: Das Backend ist mithilfe von Python entwickelt und stellt einen funktionsfähigen REST API-Endpoint zur Verfügung.

3.3 Phase 3: Teilprojekt Frontend (App)

Ergebnis: Smartphone App ist mithilfe von Dart entwickelt, kommuniziert über das Backend und stellt eine benutzerfreundliche Anwenderschnittstelle (User Interface) zur Steuerung des Roboters zur Verfügung.

3.4 Phase 4: Integrationstest

Ergebnis: Das Zusammenspiel aller Systemkomponenten ist ausführlich getestet.

4 Durchführbarkeitsanalyse

Zur Einarbeitung und Sicherstellung der Machbarkeit sind drei Prototypen geplant.

4.1 AP-3 Servo Software-Prototyp

Mithilfe des Prototyps soll herausgefunden werden, wie die Servomotoren über Pulsweitenmodulation (PWM) gesteuert werden. Hierzu wird der Beispielcode zur PWM-Motorelektronik analysiert, angepasst und angewendet. Als Ergebnis soll eine Servotest Software entstehen, die im weiteren Verlauf des Projekt sinnvoll als Tool verwendet werden kann.

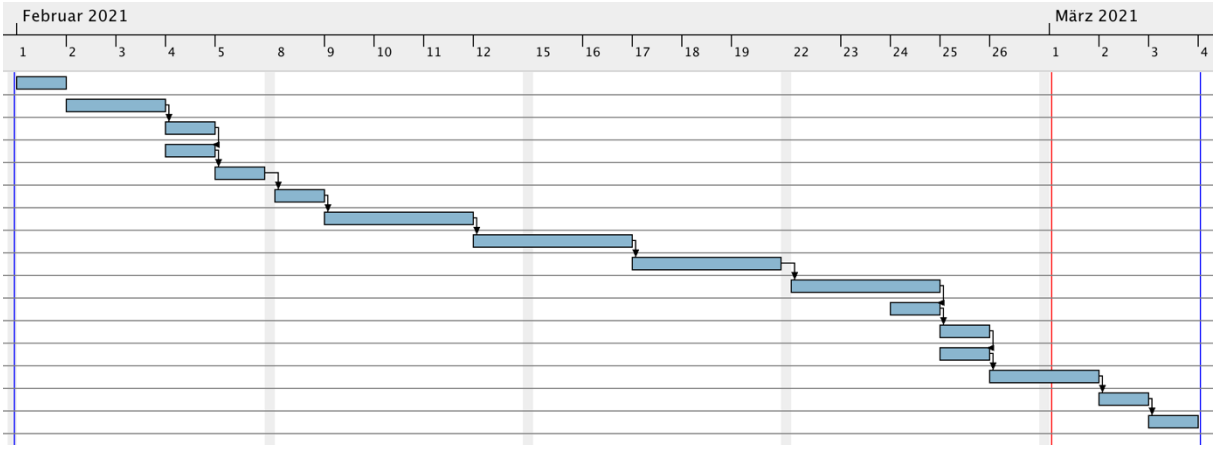
4.2 AP-4 API Prototyp

Mithilfe des Prototyps soll herausgefunden werden, wie auf Basis des Flask Frameworks eine API entwickelt wird. Dazu wird der Beispielcode aus der Flask Community analysiert, angepasst und angewendet. Als Tool zum Testen der API wird Postman verwendet.

4.3 AP-11 App Prototyp

Mithilfe des Prototyps soll herausgefunden werden, ob das Zusammenspiel der Flutter App mit der Flask API funktioniert.

Gantt Diagramm



Projektplanung	01.02.21	01.02.21
Roboterarm zusammenbauen	02.02.21	03.02.21
Raspberry Pi einrichten und mit Roboter verkabeln	04.02.21	04.02.21
Servo Software Prototyp entwickeln	04.02.21	04.02.21
Api Prototyp entwickeln	05.02.21	05.02.21
Softwaredesign Backend	08.02.21	08.02.21
Servoklasse entwickeln	09.02.21	11.02.21
Controllerklasse entwickeln	12.02.21	16.02.21
Roboterklasse entwickeln	17.02.21	19.02.21
Rest Api Server entwickeln	22.02.21	24.02.21
Api mit Postman testen	24.02.21	24.02.21
App Prototyp entwickeln	25.02.21	25.02.21
Softwaredesign Frontend	25.02.21	25.02.21
App entwickeln	26.02.21	01.03.21
Integrationstest	02.03.21	02.03.21
Dokumentation fertigstellen & Optimierung	03.03.21	03.03.21

Projektbezeichnung	Python-Programmierung zur Fernsteuerung eines Roboters
Projektleiter	Florian Schmitz
Erstellt am	26.02.2021
Letzte Änderung am	01.03.2021
Aktuelle Version	1.1

Projektablauf

Inhaltsverzeichnis

1 Wochenberichte

1.1 Woche 1 - Tag 1

Am ersten Tag wurde ein Ganttdiagramm und ein Lastenheft erstellt.

1.2 Woche 1 - Tag 2

Am zweiten Tag wurde die Montage des Roboterarms angefangen. Sie konnte an diesem Tag nicht wie geplant abgeschlossen werden, da in der Anleitung nicht genau beschrieben war, welche Schrauben man verwenden sollte oder in welcher Richtung die Teile montiert werden sollten. Dies hatte zur Folge, dass Montageschritte mehrfach wiederholt werden mussten.

1.3 Woche 1 - Tag 3

Am dritten Tag wurde der Roboterarm fertig montiert. Daraufhin wurde das Betriebssystem des Raspberry Pi installiert und eingerichtet. Während des Installationsvorgangs wurde damit begonnen das Lastenheft zu schreiben. Zum Schluss wurde der Raspberry Pi mit dem Roboterarm verkabelt.

1.4 Woche 1 - Tag 4

Am vierten Tag wurde eine Servo Testing-Software in Python geschrieben, mit der über den Raspberry Pi und der MotoPi Elektronik, die Servomotoren einzeln ansteuerbar sind. Hierfür wurde auf dem Software Beispiel der Adafruit Bibliothek aufgebaut. Zuvor musste recherchiert werden, was Pulsweitenmodulation bedeutet und welche Angaben im Datenblatt des Servomotor-Typs stehen.

Die Beispiel-Software wurde bis auf eine Funktion komplett umgeschrieben und für die Anforderungen dieses Projekts angepasst. Beim Start des Programmes werden alle angeschlossenen Servomotoren auf ihre Standardposition gefahren (1500 µs). Im Anschluss kann ein Servomotor ausgewählt werden und dieser auf die gewünschte Position durch Vorgabe der Pulsweite in Mikrosekunden (500 µs bis 2500 µs) gefahren werden. Mit q beendet man dann dieses Programm.

Mit dieser Funktionalität eignet sich das Programm sehr gut, um die korrekten Einstellungen der Servomotoren für die jeweilige Roboterachse herauszufinden und die mechanische Beweglichkeit des Roboters zu prüfen.

1.5 Woche 1 - Tag 5

Am fünften Tag wurde ein API-Prototyp mit Python entwickelt. Es ermöglicht es mit dem Browser einem API-Client (z.B. Postman) auf die API zuzugreifen. Als Rückgabewert wird ein JSON Objekt gesendet (GET Befehl).

Des Weiteren kann man einen POST Request an die API schicken und somit das JSON Objekt verändern.

Die beiden Funktionen werden zu einem späteren Projektzeitpunkt benötigt, um die Servomotor Positionen abzurufen und zu verändern.

1.6 Woche 2 - Tag 1

Am ersten Tag dieser Woche wurde ein Plan erstellt, wie das Backend aussehen soll und in welche Klassen man es aufteilt. Hierzu wurde ein UML Diagramm mithilfe von Draw.io erstellt.

1.7 Woche 2 - Tag 2

Am nächsten Tag wurde das geplante vom Vortag umgesetzt. Mit den Prototypen der Servo-Klasse wurde begonnen. Im Anschluss wurde der API-Prototyp mit den Prototypen der Servo-Klasse verbunden. Nach diesem Tag war das Backend in der Lage einen Servomotor mit Postman von einem anderen Gerät aus zu steuern.

1.8 Woche 2 - Tag 3

Am dritten Tag wurde angefangen das UML Diagramm umzusetzen. Der Servo Prototyp wurde verbessert und die API-Klasse wurde aufgeteilt, in eine Roboter- und Ressource Klasse.

1.9 Woche 2 - Tag 4

An diesem Tag wurde Fehler Behandlungen durchgeführt und vieles ausprobiert.

1.10 Woche 2 - Tag 5

Am letzten Tag dieser Woche ging das Backend dann. Man konnte mit Postman jeden beliebigen Servomotor steuern. Leider ging an diesem Tag der sechste Servomotor des Roboterarmes kaputt. Dieser wurde dann ersetzt und der Greifer etwas verändert. Dies musste getan werden, damit der Greifer richtig greifen konnte, denn die Zahnräder griffen nicht richtig ineinander und damit konnte der Greifer sich nicht bewegen.

1.11 Woche 3 - Tag 1
Am ersten Tag wurde eine JSON Datei erstellt, die 15 Servomotoren beinhaltet. Dies war notwendig da die MotoPi Aufsatzplatine 15 Steckmöglichkeiten besitzt. In dieser JSON Datei stehen Daten, wie Minimalwert, Maximalwert oder Standartwert. Mit dieser Datei konfiguriert man jeden Servomotor individuell.

1.12 Woche 3 - Tag 2
Am zweiten Tag wurde die Roboter Ressource Klasse verändert, da sonst die JSON Datei nicht korrekt eingelesen und weiterverarbeitet werden konnte.

1.13 Woche 3 - Tag 3 und Tag 4
Am dritten und vierten wurde die Umrechnung der Servomotor Daten entwickelt. Der Servomotor braucht Mikrosekunden, die von 500 bis 2500 Mikrosekunden gehen. Dies könnte man im Frontend auch so programmieren, aber da jeder Servomotor durch die Konfigurationsdatei anders konfiguriert ist, muss das Programm das automatisch richtig umrechnen (zu -1000 bis 1000). Zum Beispiel ist der Standartwert oder der Minimalwert von einem Servomotor anders und das müsste man an zwei Orten ändern. Durch die Umrechnung nur in der Konfigurationsdatei.

1.14 Woche 3 - Tag 5
Am letzten Tag dieser Woche wurden Testfälle in Python geschrieben, damit der Code automatisiert getestet werden kann.

1.15 Woche 4 - Tag 1, Tag 2 und Tag 3

An den ersten drei Tagen, wurde die Smartphone App entwickelt. Dies wurde mit dem Framework Flutter entwickelt. Das schwierigste während der App Entwicklung war, die API aufzurufen. Dies lag daran, dass die erforderlichen Kenntnisse noch nicht zur Verfügung standen und man am Anfang den Falschen „Header“ verwendet hatte.

1.16 Woche 4 – Tag 4

Am vierten Tag wurde das Backend so programmiert, dass der Roboter lernen konnte. Das heißt durch die App, kann man den Roboter an die Stelle Steuern, die man haben möchte. Im Anschluss betätigt man einen Button. Durch diesen Button wird ein PUT Request an das Backend geschickt und das Backend merkt sich dann die Positionen der Servomotoren. Dies kann man beliebig oft wiederholen. Wenn man im Anschluss den „play“ Button betätigt, wird die Sequenz ausgeführt.

1.17 Woche 4 - Tag 5

Am fünften wurde der Greifer des Roboterarms verbessert, indem das Kugellager gedreht und noch ein paar Unterlegscheiben hinzugefügt wurden. Dies war notwendig, da der Greifer instabil war. Leider wurde dies nicht so in der Anleitung beschrieben. Danach wurde mit der Dokumentation angefangen.

Zum Schluss wurde ein Video des Roboters und der App gedreht. Dies soll ein Backup sein, da die Servomotoren schnell kaputt gehen können und das während der Präsentation passieren kann. Falls das in Kraft tritt, habe ich die Möglichkeit, die Funktionen per Video[1] zu zeigen

[1] https://youtu.be/DA7x8Jc-tic
https://youtu.be/UnwspfrYE10

2 Arbeitspaketberichte

2.1 AP-1 Roboterarm zusammenbauen

2.1.1 Ziele

Aus den Teilen des Roboterbausatzes, wie z.B. Servomotoren, Metallgestänge, Schrauben und Basisplatte aus Plexiglas wird ein funktionsfähiger Roboterarm zusammengebaut. Die Elektronik, bestehend aus Raspberry Pi und der MotoPi Elektronik wird montiert und verkabelt. Anschließend wird die System-Software installiert und konfiguriert, so dass ein Remote-Zugriff über WLAN und SSH-Protokoll möglich ist. Dies ist notwendig, um über mein MacBook entwickeln zu können.

2.1.2 Voraussetzungen

Das folgende Material wird benötigt:

Pos.	Anzahl	Bezeichnung	Beschreibung
1	1	Roboterarm Bausatz	Name: Grab-It, Artikelnummer: Robot02, Hersteller: Joy-It
2	1	Plexiglasplatte, Plastikfüße, Befestigungsmaterial	Zur Stabilisierung des Roboters. Basisplatte aus 4mm starken Plexiglas, Abmessung: 40x40cm
3	1	Raspberry Pi 4	Ein kleiner Computer, um in diesem Fall ein Python Programm auszuführen
4	1	Raspberry Pi Netzteil	Stromversorgung für den Raspberry Pi
5	1	Moto-Pi PWM Controller	Aufsatzplatine für Raspberry Pi zur Ansteuerung der Servomotoren
6	1	Moto-Pi Netzteil	Leistungsstarkes Netzteil zur Stromversorgung der Servomotoren
7	7	Kabelbinder	Kabelbinder zur Montage des Kabelbaums
8	1	SD-Karte	SD-Karte für den Raspberry Pi, um das Betriebssystem und Daten zu speichern
9	1	Servotester Hardware	Ein Servotester (Hardware) ist ein kleines Gerät, dass die Servomotoren bewegen oder in Neutralstellung bringen kann
10		Werkzeug	Schraubendreher, Zange, Seitenschneider, Gabelschlüssel

2.1.3 Bericht

Nach Durchlesen der Montageanleitung wird der Roboter Schritt für Schritt montiert.

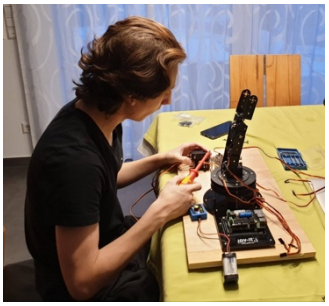


Abb. 1: Montage

Wichtig hierbei war das Überprüfen der Servomotoren mit Hilfe des Testgeräts und deren Kalibrierung auf die Neutralposition, bevor sie in den Roboter eingebaut wurden. Es wurde festgestellt, dass der Roboter nicht bei allen Achsen den vollen theoretischen Bewegungsradius der Servomotoren verwenden durfte. Daher wird der Bereich später durch die Software per Konfiguration beschränkt werden.

Nach der Montage der Roboterteile und der Servomotoren wird die MotoPi Elektronik auf den Raspberry Pi gesteckt. Zusammen werden die Komponenten auf der Roboter-Basisplatte festgeschraubt.

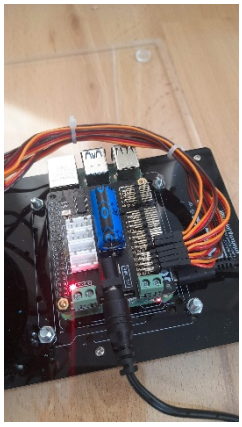


Abb. 2: Verkabelung

Schließlich wurde die gesamte Roboterarm-Einheit mitsamt der Roboter-Basisplatte mit Hilfe von vier stabilen Schrauben auf der Plexiglas-Platte befestigt.

Im nächsten Schritt wurden die Servomotor-Kabel an der MotoPi Aufsatzplatine angeschlossen und mit Hilfe von Kabelbindern ordentlich als Kabelbaum verlegt. Hierbei musste darauf geachtet werden, dass die Bewegung des Roboterarms nicht durch zu kurze Kabel eingeschränkt wird.

2.1.4 Ergebnisse

Der Roboterarm wurde erfolgreich montiert, befestigt und verkabelt. Herausfordernd war dabei die nicht immer ausreichend aussagekräftige Anleitung. Daher mussten einige Montageschritte aufgrund notwendiger Korrekturen wiederholt werden.

2.2 AP-2 Raspberry Pi einrichten

2.2.1 Ziele

Der Raspberry Pi soll mit dem neuesten Raspbian Betriebssystem laufen. Außerdem soll es möglich sein auf den Raspberry Pi mit Hilfe des SSH-Protokolls zuzugreifen. Zum Schluss soll die Adafruit Bibliothek (Software Treiber) installiert werden, um die Servomotoren ansteuern zu können.

2.2.2 Voraussetzungen

Pos.	Anzahl	Bezeichnung	Beschreibung
1	1	Raspberry Pi	Ein kleiner Computer, um in diesem Fall ein Python Programm auszuführen
2	1	SD-Karte	SD-Karte für den Raspberry Pi, um das Betriebssystem und Daten zu speichern
3	1	Software zum Installieren und Beschreiben der SD-Karte	Raspberry Pi Imager

2.2.3 Bericht

Nach der Übertragung des Betriebssystems auf die SD-Karte, wurde diese in den Raspberry Pi eingelegt. Für die anschließenden Konfigurationsschritte wurden temporär Tastatur und Monitor angeschlossen. Im Anschluss wurde der Raspberry Pi gebootet. Es wurde die Desktop Version installiert, so dass die Konfiguration mit der grafischen Oberfläche recht schnell durchführbar war. Während der Konfiguration, wurden die neusten Updates installiert und die deutsche Sprache etc. eingestellt. Schließlich wurde WLAN eingerichtet und der SSH-Server in der „raspi-config“ aktiviert. Dies ermöglicht den Betrieb des Raspberry in der Folge ohne Tastatur und Monitor („headless“) und den bequemen Zugriff über das MacBook als Entwicklungsrechner. Als nächstes wurde die Adafruit Bibliothek installiert, um die Servomotoren ansteuern zu können. Die mitgelieferte Testsoftware wurde erfolgreich ausgeführt. Zum Schluss wurde eingestellt, dass der Raspberry Pi nur noch im Konsolen Modus startet.

2.2.4 Ergebnisse

Das Raspbian Betriebssystem wurde erfolgreich installiert, gestartet und konfiguriert. Dazu wurde die Adafruit Bibliothek installiert und das Python Test Programm erfolgreich ausgeführt. Einzelne Servomotoren wurden das erste Mal mit einem Python Programm gesteuert.

2.3 AP-3 Servo Software-Prototyp entwickeln

2.3.1 Ziele

Die Servomotoren sollen durch ein Python Testprogramm einzeln angesteuert werden. Dies soll das Testen und später das Bestimmen der richtigen Servomotor Konfiguration für die jeweiligen Roboterachsen erleichtern. Im Anschluss soll eine Prototypklasse entwickelt werden, die die Servomotoren ansteuert und dessen Position etc. merken kann.

2.3.2 Voraussetzungen

Pos.	Anzahl	Bezeichnung	Beschreibung
1	1	Raspberry Pi	Ein kleiner Computer, um in diesem Fall ein Python Programm auszuführen
2	1	Adafruit Bibliothek	Eine Python Bibliothek zur Ansteuerung der Servomotoren
3	1	RPI.GPIO	Eine Python Bibliothek zur Ansteuerung der Servomotoren

2.3.3 Bericht

Das Testprogramm der Adafruit Bibliothek wurde analysiert. Im Anschluss wurde die Servotest Software mit Python entwickelt. Hierbei wurde die „set_servo_pulse“ Funktion des Testprogrammes von Adafruit übernommen.

Der Servo-Klasse Prototyp verwendet ebenfalls die „set_servo_pulse“ Funktion. Diese ermöglicht das Umrechnen der Pulsweite in Mikrosekunden in den korrekten Bit-Code zur Steuerung der Servomotor-Position. Zusätzlich hat die Klasse eine „reset“ Methode bekommen, die ermöglicht den Servomotor auf ihre Neutralposition zu bewegen. Die Klasse hat Attribute zur Begrenzung des Bewegungsradius der Servomotoren mit sinnvollen Standardwerten („pwm_min“, „pwm_max“). Dies ist sehr wichtig, damit die Servomotoren oder die Mechanik nicht durch Software-Fehler beschädigt werden. Später sollen diese Werte mit Hilfe einer Konfigurationsdatei individuell für jeden Servomotor einstellbar gemacht werden.

2.3.4 Ergebnisse

Das Servotest Programm und der Prototyp der Servo-Klasse wurde erfolgreich entwickelt. Außerdem konnten die Servomotoren jetzt mit Hilfe des Servotest-Programmes einzeln über Pulsweitenmodulation angesteuert werden.

2.4 AP-4 API-Prototyp entwickeln

2.4.1 Ziele

Es soll ein API-Prototyp basierend auf dem Flask-Framework entwickelt werden, der bei einem GET Befehl ein JSON-Objekt zurückgibt und bei einem PUT Befehl das JSON Objekt im Backend ändert.

2.4.2 Voraussetzungen

Pos.	Anzahl	Bezeichnung	Beschreibung
1	1	Flask Framework	Ein Framework um Websites und RESTful API Services zu entwickeln
2	1	Postman	Eine API-Client Software, die es ermöglicht die API mit GET und PUT Aufrufen zu testen, ohne dass bereits ein Frontend entwickelt worden ist

2.4.3 Bericht

Als erstes wurden Internet Quellen gesucht, um auf deren Basis einen Flask Server zu programmieren. Dies war notwendig, um mir das erforderliche Wissen anzueignen. Im Anschluss wurde ein Basis Flask Server mit RESTful API entwickelt. Dazu wurde dem Flask Server eine Ressource Klasse hinzugefügt und registriert, um per http-Protokoll empfangene PUT und GET Befehle zu verarbeiten.

Bei dem GET Befehl wurde zunächst ein leeres JSON Objekt zurückgegeben. Im Anschluss wurde eine JSON Objekt-Struktur definiert, die den Zustand von sechs Servomotoren mit Hilfe der Variablen „id“,“pos_min“,“pos_max“,“mpos_neutral“ und „pos“ abbilden kann. Das sind die Werte, die der Prototyp der Servo-Klasse braucht. Daraufhin wurde der PUT Befehl entwickelt. Dieser sorgt dafür, dass das JSON Objekt im Backend geändert und der Servo angesteuert wird. Als das erfolgreich war, wurde der Prototyp der Servo-Klasse eingebunden. Damit wurden die Servomotoren per REST-API ansteuerbar gemacht.

Als diese Klasse eingebunden war, konnte der API-Prototyp jeden Servomotor ansteuern und bewegen. Dies wurde mit dem Tool Postman getestet.

2.4.4 Ergebnisse

Der API-Prototyp wurde erfolgreich entwickelt. Im Anschluss wurden alle sechs Servomotoren gleichzeitig über API gesteuert.

Hierbei wurde herausgefunden, dass man nicht alle Servomotoren gleichzeitig ansteuern sollte, da dafür die Stromversorgung des Stecker Netzteils nicht ausreichend war und dadurch die MotoPi Elektronik instabil wird. Im weiteren Verlauf war dies jedoch kein Problem, da man nicht alle sechs Servomotoren zum selben Zeitpunkt bewegen muss.

2.5 AP-5 Softwaredesign Backend

2.5.1 Ziele

Ziel ist es einen Plan zu erstellen, indem deutlich wird, wie das Backend programmiert wird.

2.5.2 Voraussetzungen

Pos.	Anzahl	Bezeichnung	Beschreibung
1	1	Prototyp der Servo-Klasse	Ein Prototyp der einen Servomotor darstellen soll. Diese Klasse gibt die aktuelle Position etc. zurück. Außerdem kann diese Klasse die Servomotoren bewegen
2	1	Prototyp der API-Klasse	Ein Prototyp, der einen Flask Server zur Verfügung stellt und ein JSON Objekt verändern kann
3	1	Software Tool Daw.io	Mit dieser Software (als kostenfreie Web App verfügbar) lassen sich Zeichnungen, wie z.B. UML-Diagramme erstellen

2.5.3 Bericht

Um ein übersichtliches Software Design zu ermöglichen, wurde vor Beginn der Entwicklung überlegt, wie die Software mit Hilfe einer objektorientierten Modellierung strukturiert werden kann.

Als geeignete Methodik zur Darstellung wurde das UML Klassendiagramm ausgewählt. Nach Recherche und Einarbeitung in die Methodik und aufbauend auf dem während der Prototyp-Entwicklung erlernten Wissen, wurde das hier dargestellte Klassendiagramm erstellt.

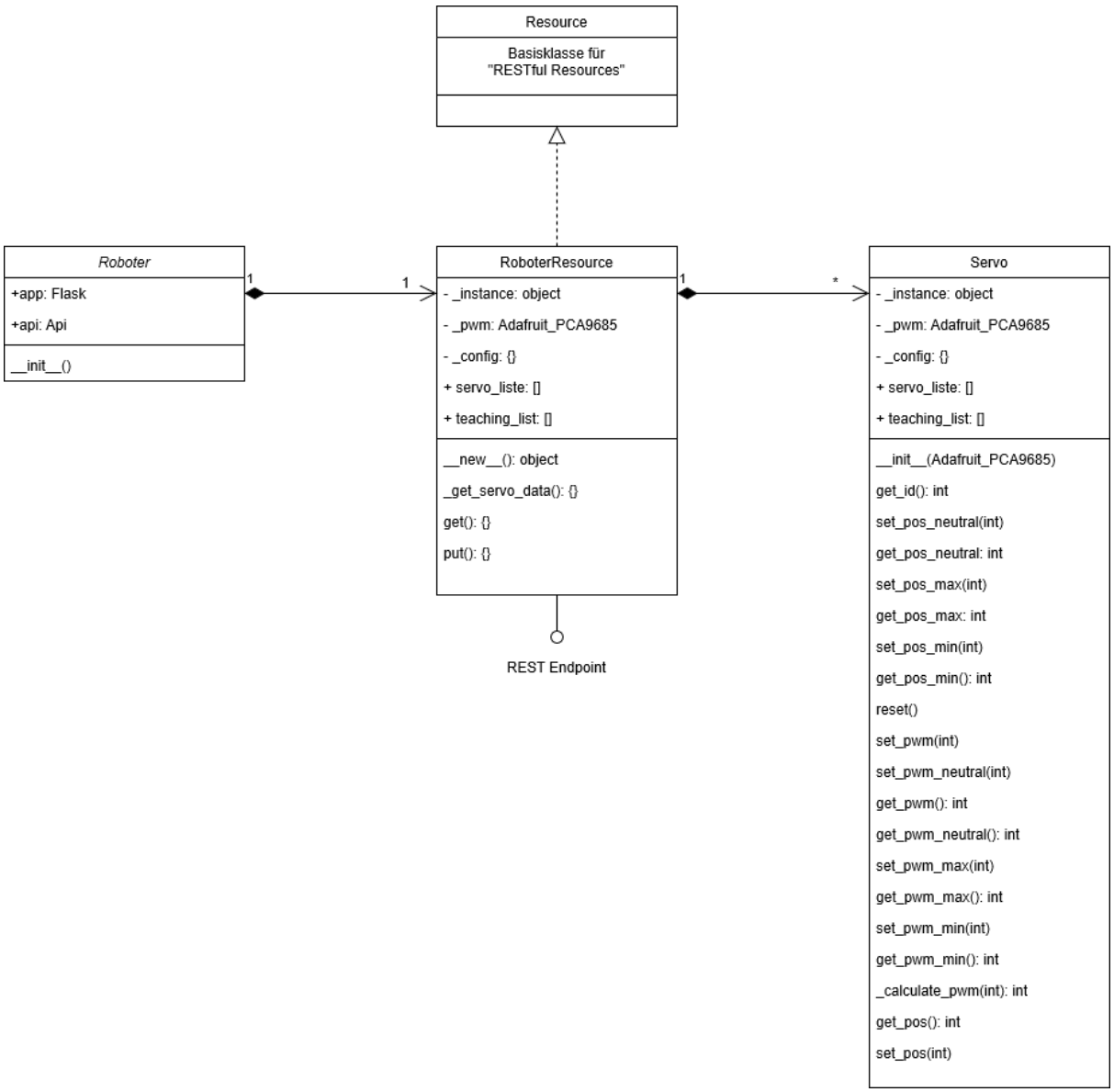


Abbildung 3: Klassen-Diagramm

Die Roboterklasse ist eigentlich nur eine Hülle, die alle benötigten Software-Komponenten initialisiert. Im Hauptprogramm muss lediglich ein Objekt dieser Klasse erzeugt werden.

Die RoboterResource Klasse soll den API-Service implementieren und dazu einen GET und einen PUT End Point mit entsprechenden Methoden zur Verfügung stellen.

Beim aller ersten API-Aufruf nach dem Service Start soll zuvor eine Konfigurationsdatei (JSON-Datei) eingelesen und anschließend 6 Servo-Objekte mit den in der Konfigurationsdatei gespeicherten Servo-spezifischen Parametern initialisiert werden. Der Aufbau der Konfigurationsdatei ist in der nebenstehenden Abbildung 4 gezeigt.

```
{
  "servos": {
    "0": {
      "id": 0,
      "pwm_min": 600,
      "pwm_max": 2400,
      "pwm_neutral": 1500,
      "pos_min": -1000,
      "pos_max": 1000,
      "pos_neutral": 0
    },
    "1": {
      "id": 1,
      "pwm_min": 1000,
      "pwm_max": 2300,
      "pwm_neutral": 1550,
      "pos_min": -1000,
      "pos_max": 1000,
      "pos_neutral": 0
    },
    "2": {
      "id": 2,
      "pwm_min": 600,
      "pwm_max": 2400,
      "pwm_neutral": 1450,
      "pos_min": -1000,
```

Abb. 4: Konfigurationsdatei

Schließlich musste noch definiert werden, wie die Transportobjekte aussehen, die per GET und PUT Request ausgetauscht werden. Auch hierfür wurde eine JSON Struktur definiert.

Im Abschnitt „servos“ werden für einen oder mehrere mit IDs gekennzeichneten Servomotoren in einer Liste die Positionswerte übertragen. Bei einem GET Aufruf werden die Ist-Positionen zurückgegeben. Bei einem PUT Aufruf enthält die Liste die Soll-Positionen.

Im Abschnitt „teach“ werden Befehle übertragen.

2.5.4 Ergebnisse

Das Software-Design des Backend wurde mittels Klassendiagramm visualisiert. Außerdem wurde die Struktur der Konfigurationsdatei und die Struktur der JSON-Transport-Objekte definiert.

```
{
  "servos": [
    {
      "id": 0,
      "pos": 1000
    },
    {
      "id": 3,
      "pos": 250
    }
  ],
  "teach": [
    {
      "teaching": true,
      "run": false,
      "reset": false,
      "example": false
    }
  ]
}
```

Abb. 5: JSON Transport Struktur

2.6 AP-6 Servo-Klasse entwickeln

2.6.1 Ziele

Das Ziel war es den Prototypen der Servo-Klasse weiter zu entwickeln und zu verbessern. Die Klasse soll mit Getter- und Setter-Methoden erweitert werden, um auf die Werte von Attributen zugreifen zu können oder um diese zu verändern. Beispiel: Min-Pulsweite, Neutral-Pulsweite und Max-Pulsweite, um den Bewegungsradius des Servomotors an die Mechanik anzupassen.

Ein weiteres Ziel ist die Bereitstellung einer Möglichkeit den Servomotor über einen Positionswert zu steuern, der intern in die dazugehörige Pulsweite umgerechnet wird. In der Roboter App sollen später Slider Widgets eingesetzt werden, um den Bediener die Steuerung der Position der Servomotoren zu ermöglichen. Abhängig von der Positionierung des Sliders, durch den Bediener wird ein entsprechender Positionswert generiert. Der Wertebereich des Positionswertes ist anders als der Wertebereich der Pulsweite.

2.6.2 Voraussetzungen

Pos.	Anzahl	Bezeichnung	Beschreibung
1	1	Prototyp der Servo-Klasse	Ein Prototyp der einen Servomotor darstellen soll. Diese Klasse gibt die aktuelle Position etc. zurück. Außerdem kann diese Klasse die Servomotoren bewegen

2.6.3 Bericht

Die Implementierung der Getter- und Setter-Methoden war sehr einfach. Viel komplizierter war die Implementierung der Umrechnung der Positionswerte des Controllers (Slider Widgets in der App) in die dazugehörigen Pulsweiten in Mikrosekunden. In den folgenden Abschnitten sind das Problem und die Lösung näher beschrieben.

2.6.3.1 Aufgabenstellung im Detail

Die Servomotoren werden durch Pulsweitenmodulation gesteuert, das heißt ihre Position ist abhängig vom Puls-Pausenverhältnis, das in Mikrosekunden vorgegeben wird. Der maximale Wertebereich hierfür ist abhängig vom Servomotor-Typ und ist im Datenblatt vorgegeben.

Für die im Projekt eingesetzten Servomotoren gelten laut Datenblatt folgende Grenzwerte:

Parameter	Wert
Pulsweite Minimum	500 µs
Pulsweite Neutral	1500 µs
Pulsweite Maximum	2500 µs

Durch den Einsatz des Servomotors als Antrieb einer Roboter-Achse ist der Bewegungsradius aus mechanischen Gründen gegebenenfalls einzuschränken. Auch die gewünschte Mittelstellung kann abweichend von der normalen Neutralposition sein. Für jeden Servomotor werden bei der Inbetriebnahme daher jeweils die drei tatsächlich zu verwendenden Werte ermittelt und in einer Konfigurationsdatei im Backend gespeichert.

Beispiel: Für den in Achse 2 eingesetzten Servomotor (Servo-Id:1) sind folgende Werte bei der Kalibrierung ermittelt worden:

Parameter	Wert
Pulsweite Minimum	1000 µs
Pulsweite Neutral	1550 µs
Pulsweite Maximum	2300 µs
Slider Position Minimum	-1000
Slider Position Neutral	0
Slider Position Maximum	1000

Benötigt wird daher eine Funktion im Backend in der Servo-Klasse, die Slider Positionswerte in die entsprechende Pulsweite umrechnen kann.

2.6.3.2 Analyse der Aufgabenstellung

Die Positionswerte des Sliders kann man als x-Werte in einem Koordinatensystem interpretieren. Die dazugehörigen Pulsweite Werte kann man als y-Werte interpretieren.

Die Werte-Tabelle zu dem im vorherigen Kapitel beschriebenen Beispiel sieht wie folgt aus:

x-Werte	-1000	0	1000
y-Werte	1000	1550	2300

Die Werte lassen sich in einem Koordinatensystem als Punkte darstellen.

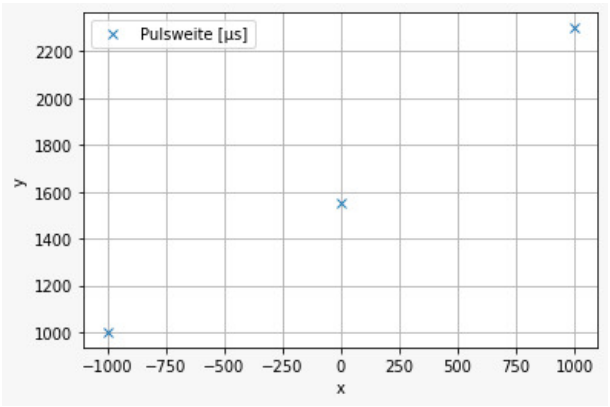


Abb. 6: Punkte im Koordinatensystem

Zwei Punkte definieren eine Gerade. Bei drei Punkten ergeben sich daher zwei Geraden. Das wird ersichtlich, wenn wir die Punkte linear miteinander verbinden. Beide Geraden haben in diesem Beispiel unterschiedliche Steigungen.

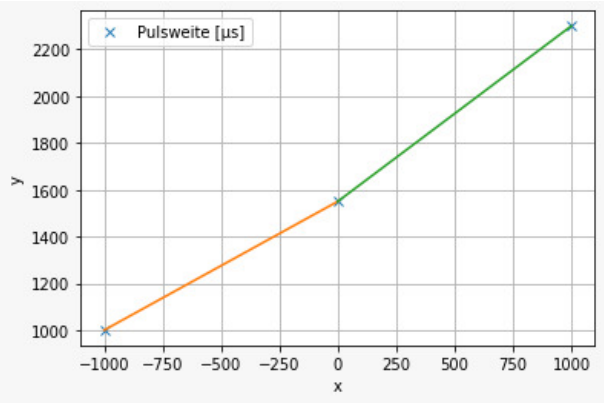


Abb. 7: Geraden im Koordinatensystem

2.6.3.3 Lösung

Um die Pulsweite zu beliebigen Slider Positionen manuell zu ermitteln, kann man die Werte aus der Grafik ablesen. Für Slider Positionswerte kleiner der Neutralposition muss dafür die erste Gerade verwendet werden. Für Werte größer gleich der Neutralposition die zweite Gerade. Für eine automatisierte Berechnung ist es erforderlich die beiden Geradengleichungen zu bestimmen.

Die allgemeine Geradengleichung lautet wie folgt:

y = m * x + b

Hierbei ist m die Steigung der Geraden und b der y-Achsenabschnitt.

Die Steigung m lässt sich mit Hilfe des Steigungsdreiecks[2] ermitteln.

Die Formel hierfür lautet:

m = (y2 - y1) / (x2 - x1)

Der y-Achsenabschnitt lässt sich durch Umformen der Geradengleichung und Einsetzen der Steigung m und eines Werte-Paars ermitteln.

Die Formel hierfür lautet:

b = y - m * x

Für das oben beschriebene Beispiel wurden anhand der dargestellten Formeln die Werte ermittelt:

m1 = 0,55

b1 = 1550

m2 = 0,75

b2 = 1550

---

[2] https://mathematik-wissen.de/klasse-7/lineare-funktion/steigung-einer-linearen-funktionermitteln-steigungsdreieck-und-zweipunkteform

Zur Überprüfung wurden für verschiedene x-Werte die dazugehörigen y-Werte errechnet und die Werte-Paare als Punkte in das Diagramm eingetragen:

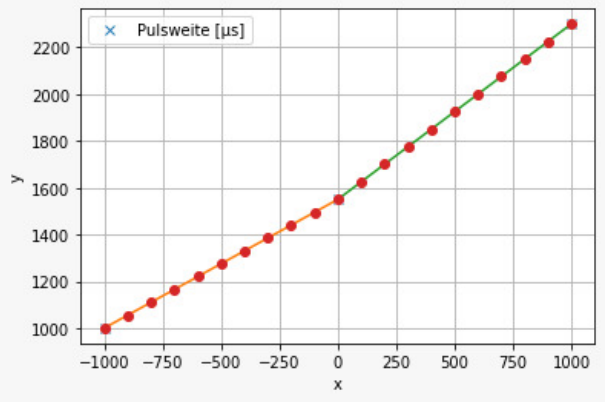


Abb. 8: Überprüfung der Geradengleichungen

Wie im Diagramm gut zu erkennen ist, liegen alle berechneten Punkte auf den beiden Geraden. Folglich funktioniert die Umrechnung der Positionswerte in die für die Servo-Steuerung benötigte Pulsweite.

Die oben gezeigten Diagramme und Berechnungen wurden mit Python mit Hilfe der Erweiterung „Jupyter Notebook[3]“ gemacht.

2.6.4 Ergebnisse

Die Servo-Klasse wurde wie geplant weiterentwickelt, so dass sie nun voll funktionstüchtig ist. Ein Objekt dieser Klasse ist das digitale Abbild eines Servomotors. Darüber hinaus wurden zur Überprüfung automatisierte Unit-Test-Fälle entwickelt.

[3] https://jupyter.org/

2.7 AP-7 Controllerklasse entwickeln

2.7.1 Ziele
Das Ziel der Controller Klasse war, die Umrechnung von der PWM in den Wertebereich -1000 bis 1000.

2.7.2 Voraussetzungen

Pos.	Anzahl	Bezeichnung	Beschreibung
1	1	Servo-Klasse	Ein Klasse die einen Servomotor digital abbildet. Diese Klasse gibt die aktuelle Position etc. zurück. Außerdem kann diese Klasse die Servomotoren bewegen

2.7.3 Bericht
Während der Bearbeitung von Arbeitspaket AP-5 wurde entschieden die Umrechnung mit in die Servo-Klasse (AP-6) zu integrieren und dafür keine extra Controller Klasse zu entwickeln.

2.7.4 Ergebnisse
Die Umrechnung wurde bereits erfolgreich in AP-6 entwickelt.

2.8 AP-8 Roboterklasse entwickeln

2.8.1 Ziele
Die Roboterklasse soll alle erforderlichen Objekte initialisieren.

2.8.2 Voraussetzungen

Pos.	Anzahl	Bezeichnung	Beschreibung
1	1	Servo-Klasse	Ein Klasse die einen Servomotor digital abbildet. Diese Klasse gibt die aktuelle Position etc. zurück. Außerdem kann diese Klasse die Servomotoren bewegen
2	1	Prototyp der API-Klasse	Ein Prototyp, der einen Flask Server zur Verfügung stellt und ein JSON Objekt verändern kann

2.8.3 Bericht
Im Arbeitspaket AP-5 wurde entschieden, den Flask Server in der Roboterklasse zu starten. Die RoboterRessource Klasse, die in AP-9 entwickelt wird, muss im Flask Server registriert werden. Dieser erzeugt dann bei einem API-Request ein Objekt der Ressource Klasse.

2.8.4 Ergebnisse
Die Klasse wurde erfolgreich entwickelt.

2.9 AP-9 Rest API-Server entwickeln

2.9.1 Ziele
Das Ziel ist eine RoboterRessource Klasse zu entwickeln, die API-Requests mit PUT und GET Befehlen verarbeiten kann. Bei einem GET Befehl sollen die im Servo-Objekt gespeicherten Positionswerte der Servomotoren zurückgegeben und bei einem PUT Befehl die Werte verändert und die Servomotoren bewegt werden.

Des Weiteren soll eine Konfigurationsdatei eingelesen werden, die jeden Servomotor individuell einstellbar macht, um damit den Drehbereich abhängig von den mechanischen Gegebenheiten einschränken zu können. Dies ist sehr wichtig, um Beschädigungen an den Servomotoren zu verhindern.

Außerdem soll wie im Lastenheft gefordert eine Funktion entwickelt werden, die eine vordefinierte Bewegungsabfolge des Roboters ausführt. Hierbei soll der Roboter eine Box (kleiner Karton) greifen und von einer festgelegten Position zu einer anderen transportieren.

Zusätzlich zu den im Lastenheft definierten Anforderungen wurde entschieden eine Teach-Funktion zu implementieren, so dass der Roboter einfach eine neue Bewegungsabfolge erlernen und ausführen kann.

2.9.2 Voraussetzungen

Pos.	Anzahl	Bezeichnung	Beschreibung
1	1	Servo-Klasse	Ein Klasse die einen Servomotor digital abbildet. Diese Klasse gibt die aktuelle Position etc. zurück. Außerdem kann diese Klasse die Servomotoren bewegen
2	1	Roboter Klasse	Eine Klasse, die alle benötigten Objekte erzeugt.
3	1	Prototyp der API-Klasse	Ein Prototyp, der einen Flask Server zur Verfügung stellt und ein JSON Objekt verändern kann

2.9.3 Bericht

Um die Roboter Ressource Klasse zu entwickeln, wurde ein Teil des API-Prototypen übernommen. Zusätzlich wurde eine JSON Konfigurationsdatei erstellt. Diese wird beim Start des Programmes eingelesen. In der Konfigurationsdatei sind 16 Servomotoren aufgelistet, da die MotoPi Elektronik 16 Anschlussmöglichkeiten besitzt. Bei jedem Servomotor kann man den Drehbereich und die Neutralposition konfigurieren.

Bei der Implementierung ist folgendes Problem aufgetreten: Standardmäßig erzeugt das Flask Framework bei jedem API-Request eine neue Instanz der RoboterRessource Klasse. Das würde dazu führen, dass jedes Mal die Servo-Objekte neu erzeugt werden. Das kann nicht funktionieren.

Um das Problem zu lösen, wurde nach einer Recherche das sogenannte „Singleton“ Design-Pattern[4] angewandt. Es prüft vor der Erzeugung eines neuen Objekts in der __new__ Methode (Konstruktor), ob es bereits eine Objekt-Instanz gibt. Falls nein, dann wird ein neues Objekt erzeugt und initialisiert. Falls ja, dann wird nur das bereits vorhandene Objekt zurückgegeben.

[4] https://www.ionos.de/digitalguide/websites/web-entwicklung/was-ist-das-singleton-pattern/

Anschließend wurde die Funktion zum Ausführen einer Beispiel-Schrittfolge entwickelt. Die Roboter Schrittfolge wurde als Liste von Positionsvektoren aller 16 Servomotoren realisiert. Bei der Ausführung wird durch die Liste iteriert und die Servomotoren in die jeweiligen Positionen gesteuert. Zwischen den Schritten wird eine Pause von 0,7s abgewartet. Dieser Wert wurde durch praktische Messungen ermittelt und gibt den Servomotoren ausreichend Zeit, um sich in die vorgegebene Zielposition zu bewegen, bevor der nächste Schritt ausgeführt wird.

Zum Schluss wurde die zusätzliche Teach-Funktion entwickelt: Das Lernen einer neuen Schrittfolge. Der Grund hierfür ist, dass es sehr faszinierend zu sehen ist, wie der Roboter neue Schrittfolgen lernen kann.

Dazu wurde noch eine Teach Methode hinzugefügt, die neue Schritte zu einer Abfolge hinzufügen kann, die der Roboter auf Anforderung ausführt. Aus Zeitgründen wurde keine Speicherfunktion entwickelt, d.h. die erlernte Abfolge ist nach einem Neustart verloren.

2.9.4 Ergebnisse

Die Roboter Ressource Klasse wurde erfolgreich entwickelt. Die Konfigurationsdatei wird erfolgreich eingelesen und der Servo-Klasse übergeben. Die Ausführung einer Demo Schrittfolge funktioniert ebenfalls. Zusätzlich zu den im Lastenheft geforderten Funktionen wurde noch eine Teach-Funktion implementiert.

2.10 AP-10 API mit Postman testen

2.10.1 Ziele

Alle Software Komponenten arbeiten korrekt zusammen. Die API Requests werden erfolgreich verarbeitet und der http Statuscode 200 zurückgegeben. Das bedeutet, dass alles in Ordnung ist. Alle Servomotoren sollen sich bewegen können. Der Roboter soll eine Beispielabfolge abspielen können und die zuvor gelernten Schrittfolgen speichern und diese wieder ablaufen lassen können.

2.10.2 Voraussetzungen

Pos.	Anzahl	Bezeichnung	Beschreibung
1	1	Servo-Klasse	Ein Klasse die einen Servomotor digital abbildet. Diese Klasse gibt die aktuelle Position etc. zurück. Außerdem kann diese Klasse die Servomotoren bewegen
2	1	Roboter Klasse	Eine Klasse, die alle benötigten Objekte erzeugt.
3	1	Roboter Ressource Klasse	Eine Klasse, die GET und PUT Requests verarbeitet
4	1	Sequenz Klasse	Eine Klasse die einen Ablauf an Servomotor Positionen in einer Liste gespeichert hat
5	1	Konfigurationsdatei	Eine JSON Datei, die einstellbare Werte speichert

2.10.3 Bericht

Mit Hilfe des Tools Postman wurde ein JSON Objekt gesendet, dass verschiedene Befehle beinhaltet hat.

2.10.4 Ergebnisse

Die Tests haben alle den Statuscode 200 zurückgegeben. Das heißt der Test war erfolgreich. Alle Funktionen funktionieren.

2.11 AP-11 App Prototyp entwickeln

2.11.1 Ziele
Die App soll sechs Schieberegler besitzen und mit diesen die Servomotoren bewege können.

2.11.2 Voraussetzungen

Pos.	Anzahl	Bezeichnung	Beschreibung
1	1	Servo-Klasse	Ein Klasse die einen Servomotor digital abbildet. Diese Klasse gibt die aktuelle Position etc. zurück. Außerdem kann diese Klasse die Servomotoren bewegen
2	1	Roboter Klasse	Eine Klasse, die alle benötigten Objekte erzeugt.
3	1	Roboter Ressource Klasse	Eine Klasse, die GET und PUT Requests verarbeitet
4	1	Sequenz Klasse	Eine Klasse die einen Ablauf an Servomotor Positionen in einer Liste gespeichert hat
5	1	Konfigurationsdatei	Eine JSON Datei, die einstellbare Werte speichert

2.11.3 Bericht
In der App wurde ein Statefulwidget erstellt, das eine „Card“ mit einem Schieberegler enthält. Des Weiteren enthält es eine GET und eine PUT Methode. Jedes Mal, wenn der Schieberegler einen größeren Schritt als 500 macht wird ein PUT Befehl ausgeführt. Ebenso wenn 80 Millisekunden vergangen sind. Dies sorgt dafür, dass der Roboter sich nicht ruckartig bewegt und das Backend nicht mit PUT Requests überschwemmt wird.

2.11.4 Ergebnisse
Die Servomotoren können mit Hilfe der Schieberegler bewegt werden.

2.12 AP-12 Softwaredesign Frontend

2.12.1 Ziele
Ziel ist es einen Plan zu erstellen, indem deutlich wird, wie das Frontend programmiert wird.

2.12.2 Voraussetzungen

Pos.	Anzahl	Bezeichnung	Beschreibung
1	1	App Prototyp	Eine Prototyp App, die alle sechs Servomotoren ansteuern kann

2.12.3 Bericht
Dieser Schritt wurde aus Zeitgründen ausgelassen, um das Gesamtziel nicht zu gefährden. Stattdessen wurde die App im Arbeitspaket AP-13 direkt entwickelt.

2.12.4 Ergebnisse
Keine

2.13 AP-13 App entwickeln

2.13.1 Ziele

Die App soll ein schönes Design bekommen und alle vom Backend bereitgestellten Funktionen für den Bediener verfügbar machen.

2.13.2 Voraussetzungen

Pos.	Anzahl	Bezeichnung	Beschreibung
1	1	Servo-Klasse	Ein Klasse die einen Servomotor digital abbildet. Diese Klasse gibt die aktuelle Position etc. zurück. Außerdem kann diese Klasse die Servomotoren bewegen
2	1	Roboter Klasse	Eine Klasse, die alle benötigten Objekte erzeugt.
3	1	Roboter Ressource Klasse	Eine Klasse, die GET und PUT Requests verarbeitet
4	1	Sequenz Klasse	Eine Klasse die einen Ablauf an Servomotor Positionen in einer Liste gespeichert hat
5	1	Konfigurationsdatei	Eine JSON Datei, die einstellbare Werte speichert
6	1	App Prototyp	Eine Prototyp App, die alle sechs Servomotoren ansteuern kann

2.13.3 Bericht

Als erstes wurde die Haupt-Page mit den Control Widgets für die Steuerung der Servomotoren entwickelt. Hierzu wurden Schiebregler (Slider) verwendet.

Eine Eventhandler Routine prüft bei jeder Bewegung wieviel Zeit im Vergleich zum letzten Ereignis vergangen ist und wieviel der Slider bewegt worden ist. Abhängig von den Differenzwerten entscheidet die Methode, ob ein API-Request durchgeführt wird. Das sorgt dafür, dass sich die Slider gut bedienen lassen und das Backend nicht mit zu vielen Requests überflutet wird.

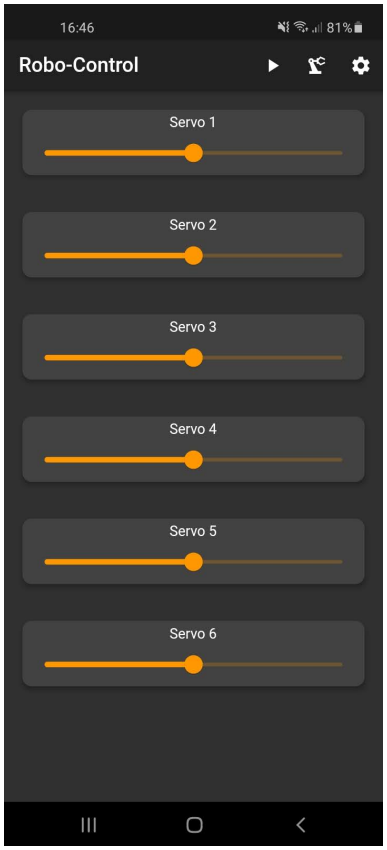


Abb. 10: Main Page

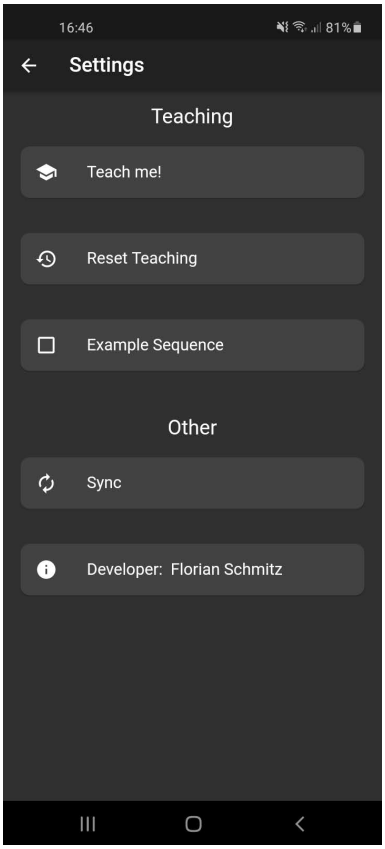


Abb. 9: Settings Page

Als nächstes wurde eine Reset Funktion entwickelt, die alle Servomotoren auf die Standartposition setzt.

Abb. 11 "reset" Button

Im Anschluss wurde die „Settings Page“ entwickelt. In dieser Seite sind die Buttons „Teachen“,“Example Sequence“ und „Sync“ enthalten. Diese sind mit den jeweiligen Funktionen des Backends verknüpft. Bei dem Synchronisationsknopf wird ein GET Reqest ausgeführt, der die Schieberegler auf den neusten Stand bringt. Dies kann z.B. dann nützlich sein, wenn der Roboter mit mehreren Smartphones gleichzeitig gesteuert oder die App neu gestartet wird. Bei den anderen Funktionen werden PUT Requests ausgeführt, die die jeweilige Funktion aktivieren.

Abb. 12 Settings Page Button

Mit dem „Play Button“ in der Haupt-Page lässt sich eine zuvor erlernte Schrittfolge ausführen

Abb. 13 "run"/“play“ Button

2.13.4 Ergebnisse

Die App ist funktionsfähig und sieht gut aus. Alle Buttons funktionieren.

2.14 AP-14 Integrationstest

2.14.1 Ziele
Durch ausführliche manuelle Tests soll die Gesamtfunktion des Systems überprüft werden.

2.14.2 Voraussetzungen
Der Roboter ist vollständig implementiert.

2.14.3 Bericht
Da die einzelnen Software Klassen bereits mit automatisierten Unit Tests basierend auf dem Python Unittest Modul gründlich getestet worden sind, konnte sich der Integrationstest auf das Zusammenspiel aller Systemkomponenten konzentrieren.

Neben den Soll-Funktionen wurden auch Sonderfälle, wie z.B. eine unterbrochene Netzwerkverbindung getestet.

2.14.4 Ergebnisse
Die Tests waren erfolgreich. Es wurde herausgefunden, dass für einen „autarken“ Aufbau ohne Nutzung einer vorhandenen Netzwerk-Infrastruktur, das WLAN-Netzwerk durch die Hotspot-Funktion eines zweiten Smartphones bereitgestellt werden muss. Das Smartphone mit der Robo-Control App kann den Hotspot nicht bereitstellen, da die App dann keine Verbindung zum Backend bekommt.

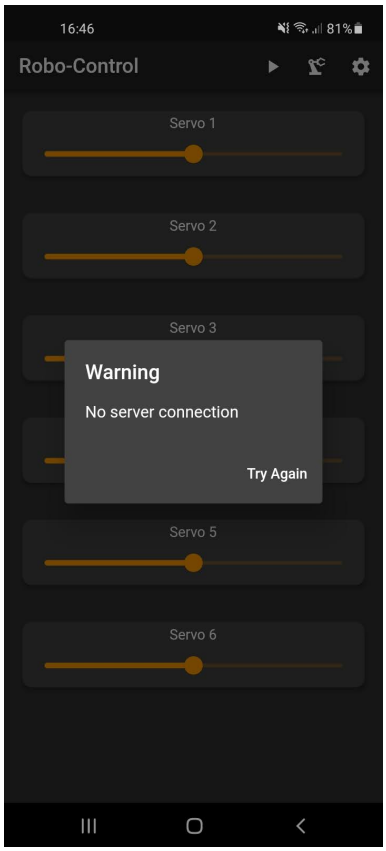


Abb. 14: Fehlerbehandlung Verbindungsproblem

2.15 AP-15 Dokumentation

2.15.1 Ziele
Das Schreiben von Pflichtenheft, Arbeitspaketberichten, Projektergebnissen, Fazit, Quellenverzeichnis und Anhang

2.15.2 Voraussetzungen
Das Projekt ist fertig

2.15.3 Bericht
Alle erforderlichen Dokumente wurden mit Word geschrieben

2.15.4 Ergebnisse
Alle erforderlichen Dokumente wurden erfolgreich geschrieben

3 Projektergebnis

In diesem Kapitel wird im Abgleich mit den Anforderungen des Lastenhefts überprüft, ob alle MUSS-Anforderungen erfolgreich umgesetzt worden ist.

3.1 Funktionale Anforderungen

3.1.1 ID 1

3.1.1.1 Priorität

Erforderlich

3.1.1.2 Beschreibung

Alle Servomotoren können durch Raspberry Pi und der MotoPi Aufsatzplatine mit Hilfe von Pulsweitenmodulation (PWM) einzeln angesteuert werden.

Laut Datenblatt des Servomotors, muss ein 50 Hz Rechtecksignal generiert werden mit einer Pulslänge im Bereich 500 bis 2500 Mikrosekunden (µs). Der Servotyp hat einen Drehbereich von bis zu 180°.

3.1.1.3 Ergebnis

Alle Servomotoren können angesteuert werden.	

3.1.2 ID 2

3.1.2.1 Priorität

Erforderlich

3.1.2.2 Beschreibung

Der erlaubte Drehbereich der Servomotoren kann einzeln eingestellt werden. Um die Servomotoren testen und einstellen zu können, soll ein Servotestprogramm entwickelt werden.

3.1.2.3 Ergebnis

Der erlaubte Drehbereich kann individuell in einer Konfigurationsdatei eingestellt werden. Das Testprogramm wurde erfolgreich entwickelt.	

3.1.3 ID 3

3.1.3.1 Priorität

Erforderlich

3.1.3.2 Beschreibung

Die neutrale Position kann man bei jedem Servomotor anpassen, um den Roboter einen individuelle Start- oder Warteposition zu geben.

3.1.3.3 Ergebnis

Die Neutralposition ist individuell in einer Konfigurationsdatei einstellbar.	

3.1.4 ID 4

3.1.4.1 Priorität

Erforderlich

3.1.4.2 Beschreibung

Der Wertebereich für die Servomotor-Position wird auf einen konfigurierbaren Wertebereich normiert. Der Wertebereich wird auf das Frontend der App angepasst. (von -1000 bis +1000)

3.1.4.3 Ergebnis

Der Wertebereich wird in der Servo-Klasse umgerechnet.	

3.1.5 ID 5

3.1.5.1 Priorität

Erforderlich

3.1.5.2 Beschreibung

Mit Hilfe von „Slidern“ oder virtuellen Controllern sollen die Servo-Positionen verändert werden.

3.1.5.3 Ergebnis

Es werden sechs Schieberegler (Slider) in der App erzeugt, mit denen die Servomotoren gesteuert werden.	

3.1.5.4 ID 6

3.1.5.5 Priorität

Erforderlich

3.1.5.6 Beschreibung

Die erforderlichen Funktionen für die Steuerung des Roboter Arms können per REST-API aufgerufen werden.

3.1.5.7 Ergebnis

Die Funktionen können per REST-API mit Hilfe eines API-Client (Postman) aufgerufen werden.	

3.1.6 ID 7

3.1.6.1 Priorität

Erforderlich

3.1.6.2 Beschreibung

Die Smartphone App nutzt die REST Schnittstelle, um mit dem Backend zu kommunizieren. Hierbei ist das Smartphone der Client und der REST API-Server das Backend.

3.1.6.3 Ergebnis

Die App kommuniziert über die REST Schnittstelle.	

3.1.7 ID 8

3.1.7.1 Priorität

Nicht erforderlich

3.1.7.2 Beschreibung

Die Servomotoren, können mit einem Button in der App auf ihre Neutralposition zurückgesetzt werden.

3.1.7.3 Ergebnis

Die Servomotoren können über einen Button in der App-Bar den Roboter zurücksetzen.	

3.1.8 ID 9

3.1.8.1 Priorität

Nicht erforderlich

3.1.8.2 Beschreibung

Der Roboter hat einen einprogrammierten Ablauf, der per Button in der App ausgeführt werden können. Zum Beispiel, dass der Roboter per Knopfdruck einen kleinen Gegenstand hochhebt und diesen in eine Schachtel legt.

3.1.8.3 Ergebnis

In der App auf der Einstellungsseite ist ein Button, mit dem man einen Ablauf starten kann. Dieser trägt dann eine kleine Box von dem Startpunkt bis zum Endpunkt	

3.2 Nichtfunktionale Anforderungen

3.2.1 ID 1

3.2.1.1 Priorität

Nicht erforderlich

3.2.1.2 Beschreibung

Die wichtigsten Funktionen werden durch automatisierte Unit Tests getestet.

3.2.1.3 Ergebnis

Einige (die Wichtigsten), aber nicht alle Funktionen werden mit automatisierten Tests getestet. Dies musste ich auch Zeitgründen begrenzen. Zum Beispiel wird die Lernfunktion des Roboters nicht getestet. Dafür wird die Servo-Klasse und ein paar Methoden der API-Klasse getestet.	

3.2.2 ID 2

3.2.2.1 Priorität

Erforderlich

3.2.2.2 Beschreibung

Es gibt eine Konfigurationsdatei, in der alle änderbaren Einstellungen vorgenommen werden können.

3.2.2.3 Ergebnis

In dieser Datei können bis zu 16 Servomotoren individuell eingestellt werden.	

3.2.3 ID 3

3.2.3.1 Priorität

Nicht erforderlich

3.2.3.2 Beschreibung

Die API ist dokumentiert, zum Beispiel mit Swagger UI.

3.2.3.3 Ergebnis

Aus Zeitgründen musste diese Anforderung gestrichen werden.	

3.2.4 ID 4

3.2.4.1 Priorität

Nicht erforderlich

3.2.4.2 Beschreibung

In der App kann der User die IP-Adresse des Servers auswählen.

3.2.4.3 Ergebnis

Aus Zeitgründen musste diese Anforderung gestrichen werden.	

3.2.5 ID 5

3.2.5.1 Priorität

Erforderlich

3.2.5.2 Beschreibung

Connection Probleme werden abgefangen und der User wird durch einen Alert Dialog in der App gewarnt.	

3.2.5.3 Ergebnis

Der User wird, wenn die App den Server nicht erreichen kann durch einen Alert Dialog gewarnt.

3.2.6 ID 6

3.2.6.1 Priorität

Erforderlich

3.2.6.2 Beschreibung

Das Smartphone kommuniziert entweder über WLAN oder Mobilen Hotspot mit dem Backend

3.2.6.3 Ergebnis

Das Smartphone kann mit WLAN oder Mobilen Hotspot mit dem Backend kommunizieren.	

3.2.7 ID 7

3.2.7.1 Priorität

Erforderlich

3.2.7.2 Beschreibung

Durch eine Konfigurationsdatei merkt sich das Backend die erlaubten Drehbereiche damit sie nicht an die mechanische Grenze fahren.

3.2.7.3 Ergebnis

Die Konfigurationsdatei merkt sich alles, für jeden Servomotor	

3.2.8 ID 8

3.2.8.1 Priorität

Erforderlich

3.2.8.2 Beschreibung

Die App wird auf einen Galaxy A21s laufen

3.2.8.3 Ergebnis

Die App wurde erfolgreich auf einem Galaxy A21s getestet	

3.2.9 ID 9

3.2.9.1 Priorität

Erforderlich

3.2.9.2 Beschreibung

Das Backend läuft auf einem Raspberry Pi 4 Model B mit 4 GB RAM.
Betriebssystem: Raspbian. Version: 10

3.2.9.3 Ergebnis

Das Backend läuft auf dem geforderten Gerät.	

3.2.10 ID 10

3.2.10.1 Priorität

Nicht erforderlich

3.2.10.2 Beschreibung

Backend startet bei einem Start oder Neustart automatisch

3.2.10.3 Ergebnis

Das Backend wurde als System-Service installiert und startet automatisch nach dem Reboot.	

3.2.11 ID 11

3.2.11.1 Priorität

Nicht erforderlich

3.2.11.2 Beschreibung

Roboter speichert eine Servomotor Positionsabfolge und startet diese

3.2.11.3 Ergebnis

Der Roboter kann über die App eine Abfolge an Servomotoren Positionen lernen und diese mit einem Button ausführen. Dies wurde zusätzlich entwickelt, da es eine sinnvolle Funktion ist und es mir Spaß gemacht hat, diese zu entwickeln.	

Fazit

Ich konnte alle Ziele des Projektes erreichen. Darüber bin ich sehr froh, da es sich gezeigt hat, dass die Aufgabenstellung im Detail sehr herausfordernd war und ich sie nur durch großen Arbeitseinsatz zum Abgabetermin abschließen konnte.

Ich fand es sehr motivierend während des Projekts, dass ich Technologien anwenden konnte, wie z.B. RESTful APIs und dass ich nach jedem Schritt als Ergebnis etwas Vorzeigbares hatte. Mit Hilfe der Prototypen bekam ich sehr schnell ein Gefühl dafür, wie es funktioniert.

Die Entwicklung der automatisierten Unit-Tests hat zwar Zeit gekostet. Sie haben es aber ermöglicht, die Tests nach jeder Software-Änderung erneut auszuführen und so eine stabile Software zu entwickeln.

Durch meine Vorkenntnisse, die ich mir in den beiden Praktika aneignen konnte und die ich mir selbst beigebracht habe, war ich in schnell in der Lage eine Smartphone App zu entwickeln.

Die größten Herausforderungen gab es bei der Entwicklung der Python-basierten Steuerungs-Software im Backend:

- Anwendung des „Singleton-Patterns“, um zu verhindern, dass bei jedem API-Request neue RoboterRessource Objekte vom Flask-Server erzeugt werden
- Umrechnung der Positionswerte der Schieberegler in die für die Servos benötigen Pulsweiten

Außerdem gab es unvorhergesehene Ereignisse, wie zum Beispiel der Defekt eines Servomotors, wofür ein Ersatz beschafft und eingebaut werden musste. Die Erstellung der Dokumentation war ebenfalls aufwändiger als gedacht.

Diese Herausforderungen haben meinen Zeitplan etwas durcheinandergeworfen. Mein persönliches Fazit ist daher, bei der Planung mehr Puffer vorzusehen und sich besser ein paar Anforderungen weniger vorzunehmen. Außerdem habe ich viel gelernt, was Programmierung, Planung und Dokumentation angeht.

Dennoch können sich die Ergebnisse sehen lassen. Ich habe sie zusätzlich als Videos dokumentiert:

- https://youtu.be/DA7x8Jc-tic
- https://youtu.be/UnwspfrYE10

Nach Abschluss des Projekts werde ich eventuell in meiner Freizeit den Roboter noch etwas weiter verbessern. Folgende Themen fallen mir dazu ein:

- Verbesserung der API: Meine erste Implementierung einer API entspricht nicht allen Design-Regeln und beinhaltet auch keine integrierte Dokumentation (Swagger UI)
- Security: Verhindern, dass nicht jeder den Roboter steuern kann
- Sofern machbar: Kombination mit Bildverarbeitung (Objekterkennung)

Quellen

Link	Datum
https://www.projektmanagementhandbuch.de/handbuch/projektplanung/lastenheft/	01.02.2021
http://www.gm.fh-koeln.de/~afomusoe/WS2011_12/LastenheftAlumni.pdf	01.02.2021
https://www.ionos.de/digitalguide/websites/web-entwicklung/lastenheft/	01.02.2021
https://joy-it.net/de/products/Robot02	02.02.2021
https://joy-it.net/files/files/Produkte/Robot02/Robot02_Aufbauanleitung.pdf	02.02. - 03.02.2021
https://joy-it.net/de/products/COM-Motor02 03.02	03.02.2021
https://joy-it.net/de/products/RB-Moto3	03.02.2021
https://www.raspberrypi.org/software/	03.02.2021
https://joy-it.net/files/files/Produkte/RB-Moto3/RB-Moto3-Anleitung-20200416.pdf	03.02.2021
https://www.elektronik-kompendium.de/sites/raspberry-pi/2107071.htm	03.02.2021
https://www.elektronik-kompendium.de/sites/raspberry-pi/1906271.htm	03.03.2021
https://joy-it.net/files/files/Produkte/Robot02/Robot02_Kalibrierung.pdf	03.03.2021
https://code.visualstudio.com/	04.02.2021
https://blog.miguelgrinberg.com/post/designing-a-restful-api-with-python-and-flask	04.02.2021
https://restfulapi.net/	04.02.2021

https://realpython.com/api-integration-in-python/	04.02.2021
https://michal.karzynski.pl/blog/2016/06/19/building-beautiful-restful-apis-using-flask-swagger-ui-flask-restplus/	04.02.2021
https://swagger.io/tools/swagger-ui/	04.02.2021
https://flask-restful.readthedocs.io/en/latest/api.html	04.02.2021
https://flask-restplus.readthedocs.io/en/stable/quickstart.html#full-example	04.02.2021
https://www.postman.com/product/api-client/	04.02.2021
https://docs.python.org/3/tutorial/datastructures.html	04.02.2021
https://www.youtube.com/watch?v=GMppyAPbLYk	05.02.2021
https://www.elektronik-kompendium.de/sites/raspberry-pi/2006071.htm	05.02.2021
https://praxistipps.chip.de/ram-info-unter-linux-so-bestimmen-sie-ihren-arbeitsspeicher_44953	05.02.2021
https://bitreporter.de/raspberrypi/welche-version-von-raspbian-ist-installiert/	05.02.2021
https://git-scm.com/book/de/v2/Git-Grundlagen-Ein-Git-Repository-anlegen	08.02.2021
https://flask-restful.readthedocs.io/en/latest/installation.html	08.02.2021
https://www.ionos.de/digitalguide/websites/web-entwicklung/klassendiagramme-mit-uml-erstellen/	08.02.2021
https://app.diagrams.net/	08.02.2021
http://www.devedge.de/python/datentyp-ausgeben-python	09.02.2021
https://www.w3schools.com/python/ref_func_isinstance.asp	09.02.2021
https://stackoverflow.com/questions/10116373/git-push-error-repository-not-found	10.02.2021
https://github.community/t/fatal-repository-not-found/1180	10.02.2021

https://help.sentry.io/hc/en-us/articles/115006788707-GitHub-Why-am-I-getting-a-404-Not-Found-error-from-GitHub-when-trying-to-add-a-repository	10.02.2021
https://docs.github.com/en/github/creating-cloning-and-archiving-repositories/error-repository-not-found	10.02.2021
https://stackoverflow.com/questions/42237752/single-instance-of-class-in-python	10.02.2021
https://de.wikipedia.org/wiki/Singleton_(Entwurfsmuster)	10.02.2021
https://riptutorial.com/python/example/17869/singleton-class	10.02.2021
https://www.ionos.de/digitalguide/websites/web-entwicklung/was-ist-das-singleton-pattern/	10.02.2021
https://stackoverflow.com/questions/31875/is-there-a-simple-elegant-way-to-define-singletons	10.02.2021
https://deparkes.co.uk/2018/08/11/flask-restful-api-json/	11.02.2021
https://stackoverflow.com/questions/37813568/git-remote-repository-not-found	11.02.2021
https://stackoverflow.com/questions/56269686/git-repository-not-found-error-for-private-repository-on-github	11.02.2021
https://docs.github.com/en/github/authenticating-to-github/creating-a-personal-access-token	11.02.2021
https://stackoverflow.com/questions/25927914/git-error-please-make-sure-you-have-the-correct-access-rights-and-the-reposito	11.02.2021
https://stackoverflow.com/questions/15240815/git-fatal-the-remote-end-hung-up-unexpectedly	11.02.2021
https://joy-it.net/files/files/Produkte/Robot02/Robot02_Aufbauanleitung.pdf	12.02.2021
https://hellocoding.de/blog/coding-language/python/json-verwenden	15.02.2021

https://m.heise.de/developer/artikel/Analysieren-von-JSON-Dateien-mit-Python-3240513.html?seite=all	15.02.2021
https://www.delftstack.com/de/howto/python/how-to-pretty-print-a-json-file/	15.02.2021
https://www.python-forum.de/viewtopic.php?t=40728	15.02.2021
https://stackoverflow.com/questions/31875/is-there-a-simple-elegant-way-to-define-singletons	16.02.2021
https://stackoverflow.com/questions/52351312/singleton-pattern-in-python	16.02.2021
https://python-patterns.guide/gang-of-four/singleton/	16.02.2021
https://www.gutefrage.net/frage/wertebereich-abbilden	17.02.2021
https://www.matheboard.de/archive/514696/thread.html	17.02.2021
https://matheplanet.com/default3.html?call=viewtopic.php?topic=32140&ref=https%3A%2F%2Fwww.google.com%2F	17.02.2021
https://www.studyhelp.de/online-lernen/mathe/lineare-funktionen/	17.02.2021
https://mathematik-wissen.de/klasse-7/lineare-funktion/steigung-einer-linearen-funktion-ermitteln-steigungsdreieck-und-zweipunkteform	18.02.2021
https://mathematik-wissen.de/klasse-7/lineare-funktion/y-achsenabschnitt-schnittpunkt-der-y-achse-mit-dem-graphen	18.02.2021
https://jupyter.org/	18.02.2021
https://energie.labs.fhv.at/~kr/python/Juypter_Notebooks.html	18.02.2021
https://gertingold.github.io/pythonnawi/graphics.html	18.02.2021
http://www.biophysik.org/~wille/python/_build/html/VL12_IPython_Matplotlib.html	18.02.2021
https://www.youtube.com/watch?v=HW29067qVWk	18.02.2021
https://realpython.com/pytest-python-testing/	19.02.2021

https://stackoverflow.com/questions/56780892/python-unit-test-module-throws-modulenotfounderror-no-module-named-tests-test	19.02.2021
https://stackoverflow.com/questions/1896918/running-unittest-with-typical-test-directory-structure	19.02.2021
https://stackoverflow.com/questions/48329498/python-modulenotfounderror-testsuite-import-error	19.02.2021
https://de.wikipedia.org/wiki/Symbolische_Verknüpfung	19.02.2021
https://www.howtogeek.com/297721/how-to-create-and-use-symbolic-links-aka-symlinks-on-a-mac/	19.02.2021
https://docs.python.org/3/library/unittest.html	19.02.2021
https://stackoverflow.com/questions/6854658/explain-the-setup-and-teardown-python-methods-used-in-test-cases	19.02.2021
https://kapeli.com/cheat_sheets/Python_unittest_Assertions.docset/Contents/Resources/Documents/index	19.02.2021
https://stackoverflow.com/questions/33657463/python-test-to-check-instance-type	19.02.2021
https://stackoverflow.com/questions/48577606/how-to-manupulate-python-unittest-output-e-and-f	19.02.2021
https://docs.python.org/3/library/unittest.html#unittest.main	19.02.2021
https://www.byte-artist.de/blog/how-to-probleme-beim-push-in-ein-gitrepo	19.02.2021
https://www.codegrepper.com/code-examples/shell/error%3A+Fehler+beim+Versenden+einiger+Referenzen+nach+%27git%40git	19.02.2021
https://www.geeksforgeeks.org/get-post-requests-using-python/	19.02.2021
https://realpython.com/intro-to-python-threading/	19.02.2021
https://stackoverflow.com/questions/31264826/start-a-flask-application-in-separate-thread	19.02.2021

https://stackoverflow.com/questions/15115328/python-requests-no-connection-adapters	19.02.2021
https://stackoverflow.com/questions/9559963/unit-testing-a-python-app-that-uses-the-requests-library	19.02.2021
https://www.geeksforgeeks.org/python-different-ways-to-kill-a-thread/	19.02.2021
https://www.ontestautomation.com/writing-tests-for-restful-apis-in-python-using-requests-part-1-basic-tests/	19.02.2021
https://kapeli.com/cheat_sheets/Python_unittest_Assertions.docset/Contents/Resources/Documents/index	19.02.2021
https://flutter.dev/	22.02.2021
https://stackoverflow.com/questions/56391114/how-to-get-json-data-from-an-api-in-flutter	22.02.2021
https://pub.dev/packages/http	22.02.2021
https://flutter.dev/docs/cookbook/networking/fetch-data	22.02.2021
https://www.appsdeveloperblog.com/http-get-request-in-flutter/	22.02.2021
https://www.digitalocean.com/community/tutorials/flutter-flutter-http	22.02.2021
https://dart.dev/tutorials/web/fetch-data	22.02.2021
https://flutter.dev/docs/cookbook/networking/background-parsing	22.02.2021
https://www.youtube.com/watch?v=xfdG8e9mgU4	22.02.2021
https://stackoverflow.com/questions/64305059/postman-is-working-but-http-is-giving-404-error-in-flutter	22.02.2021
https://stackoverflow.com/questions/55671441/flutter-formatexception-unexpected-character-at-character-1	22.02.2021
https://www.kindacode.com/article/flutter-formatexception-unexpected-character-at-character-1/	22.02.2021

https://stackoverflow.com/questions/52734323/http-put-method-in-dart	22.02.2021
https://api.flutter.dev/flutter/material/Card-class.html	22.02.2021
https://stackoverflow.com/questions/50115311/flutter-how-to-force-an-application-restart-in-production-mode	23.02.2021
https://pub.dev/packages/flutter_phoenix	23.02.2021
https://material.io/resources/icons/?style=baseline	23.02.2021
https://hellocoding.de/blog/coding-language/python/json-verwenden	23.02.2021
https://www.data-science-architect.de/dictionaries-python/	23.02.2021
https://stackoverflow.com/questions/30362391/how-do-you-find-the-first-key-in-a-dictionary/39292086	23.02.2021
https://flutter.dev/	23.02.2021
https://flutter.de/artikel/alert-dialog-flutter.html	23.02.2021
https://stackoverflow.com/questions/49648022/check-whether-there-is-an-internet-connection-available-on-flutter-app	23.02.2021
https://material.io/resources/icons/?style=baseline	23.02.2021
https://pub.dev/packages/flutter_launcher_icons	24.02.2021
https://romannurik.github.io/AndroidAssetStudio/icons-launcher.html	24.02.2021
https://material.io/resources/icons/?icon=psychology&style=baseline	24.02.2021
https://www.stickpng.com/img/bots-and-robots/bot-arm-icon	24.02.2021
https://stackoverflow.com/questions/50654820/flutter-could-not-prepare-to-run-the-isolate	24.02.2021
https://stackoverflow.com/questions/43879103/adding-a-splash-screen-to-flutter-apps	24.02.2021

https://stackoverflow.com/questions/54165549/navigate-to-a-new-screen-in-flutter	24.02.2021
https://api.flutter.dev/flutter/material/Card-class.html	24.02.2021
https://material.io/resources/icons/	24.02.2021
https://pub.dev/packages/shared_preferences	24.02.2021
https://medium.com/flutterdevs/using-sharedpreferences-in-flutter-251755f07127	24.02.2021
https://stackoverflow.com/questions/49441187/how-to-change-status-bar-and-app-bar-color-in-flutter	24.02.2021
https://medium.com/flutter-community/beginners-guide-to-text-styling-in-flutter-3939085d6607	25.02.2021
https://joy-it.net/files/files/Produkte/Robot02/Robot02_Aufbauanleitung.pdf	25.02.2021
https://www.raspberrypi-spy.co.uk/2015/02/how-to-autorun-a-python-script-on-raspberry-pi-boot/	25.02.2021
https://askubuntu.com/questions/1157538/python-script-on-boot-module-import-error	25.02.2021
https://snippets.khromov.se/show-open-ports-and-what-program-is-using-them/	25.02.2021
https://stackoverflow.com/questions/57031864/running-flask-app-automatically-after-boot-does-not-work-correctly	25.02.2021
https://www.debinux.de/2014/11/copy-paste-mit-nano/	25.02.2021
https://stackoverflow.com/questions/52569602/flutter-run-function-every-x-amount-of-seconds	26.02.2021
https://stackoverflow.com/questions/1358540/how-can-i-count-all-the-lines-of-code-in-a-directory-recursively	27.02.2021

https://stackoverflow.com/questions/5905054/how-can-i-recursively-find-all-files-in-current-and-subfolders-based-on-wildcard	27.02.2021
https://www.bimminger.at/content/bereich/tips_linux/work_tipp_linux_datei_anzahl.html	27.02.2021
https://www.scribbr.de/aufbau-und-gliederung/vorwort-beispiel/	28.02.2021
https://joy-it.net/de/products/Robot02	28.02.2021
https://www.elektronik-kompendium.de/sites/raspberry-pi/2107051.htm	28.02.2021
https://raspberrypi.stackexchange.com/questions/4284/how-to-run-a-command-with-alias	28.02.2021
https://upload.wikimedia.org/wikipedia/commons/b/b0/Light_green_check.svg	01.03.2021
https://futurezone.at/digital-life/was-das-rote-x-bei-nutzernamen-auf-twitter-bedeutet/400058045	01.03.2021

Erklärung

Ich versichere, dass ich die vorliegende Projektarbeit in allen Teilen selbstständig und ohne fremde Hilfe verfasst und keine anderen als die angegebenen Hilfsmittel verwendet habe sowie dass alle wörtlichen und sinngemäßen Übernahmen aus anderen Quellen als solche kenntlich gemacht wurden.

Zornheim, den 01.03.2021

Ort, Datum

Unterschrift

Anhang

Inhaltsverzeichnis

1 Frontend Code

1.1 Main.dart

```dart
import 'package:flutter/material.dart';
import 'package:flutter_phoenix/flutter_phoenix.dart';
import 'package:robo_app/HomePage/homePage.dart';

void main() {
  runApp(Phoenix(child: MyApp()));
}

class MyApp extends StatelessWidget {
  @override
  Widget build(BuildContext context) {
    return MaterialApp(
      title: 'Robo-Control',
      // Farben auf Schwarz und Orange gesetzt
      theme: ThemeData(
        primarySwatch: Colors.orange,
        brightness: Brightness.dark,
        visualDensity: VisualDensity.adaptivePlatformDensity,
      ),
      debugShowCheckedModeBanner: false,
      // Startseite ist die HomePage
      home: HomePage(),
    );
  }
}
```

1.2 Home Page

1.2.1 homePage.dart

```
import 'dart:convert';
import 'package:flutter/material.dart';
import 'package:flutter/services.dart';
import 'package:flutter_phoenix/flutter_phoenix.dart';
import 'package:robo_app/HomePage/noConnection.dart';
import 'package:robo_app/HomePage/slider.dart';
import 'package:robo_app/settings/settingsPage.dart';
import 'package:http/http.dart' as http;

class HomePage extends StatefulWidget {
  @override
  _HomePageState createState() => _HomePageState();
}

class _HomePageState extends State<HomePage> {
  // variablen werden deklariert
  final String url = "http://robopi:5000/servo";
  var slider1;
  var slider2;
  var slider3;
  var slider4;
  var slider5;
  var slider6;
  List slider;

  @override
  void initState() {
    super.initState();
    // Stellt die Systembar dunkel
    SystemChrome.setSystemUIOverlayStyle(SystemUiOverlayStyle.dark);
    checkConnection();

    // 6 Slider erstellt und diese in die Liste slider geschrieben
    slider1 = ControlSlider("Servo 1", "0");
    slider2 = ControlSlider("Servo 2", "1");
    slider3 = ControlSlider("Servo 3", "2");
    slider4 = ControlSlider("Servo 4", "3");
    slider5 = ControlSlider("Servo 5", "4");
    slider6 = ControlSlider("Servo 6", "5");

    slider = [slider1, slider2, slider3, slider4, slider5, slider6];
  }
```

```
// Diese Funktion sendet einen GET Request um herauszufinden
// ob der Server erreichbar ist. Wenn nciht wird ein Alert Dialog geöffnet
void checkConnection() async {
  try {
    var response = await http.get(Uri.encodeFull("http://robopi:5000/servo"),
        headers: {"Accept": "application/json"});
    print("GET Status: ${response.statusCode}");
  } catch (e) {
    print(e);
    showDialog(context: context, builder: (_) => NoConnectionDialog());
  }
}

// Die JSON Map für den Befehl run
Map jsonMap = {
  "servos": [],
  "teach": [
    {"teaching": false, "run": true, "reset": false, "example": false}
  ]
};

// Sende PUT Request
void putJsonData() {
  String jsonStr = jsonEncode(jsonMap);
  http.put(Uri.encodeFull(url),
      body: jsonStr,
      headers: {"Content-Type": "application/json"}).then((result) {
    print("PUT Status: ${result.statusCode}");
  });
}

@override
Widget build(BuildContext context) {
  return Scaffold(
    appBar: AppBar(
      title: Text("Robo-Control"),
      automaticallyImplyLeading: false,
      actions: [
        // Button um die gelernte Sequenz abzuspielen
        IconButton(
          icon: Icon(Icons.play_arrow),
          tooltip: "Sequence",
          onPressed: () {
            putJsonData();
          },
        ),
```

```
          // Button um den Roboter auf die Standartposition zurück zu setzten
          IconButton(
            icon: Icon(
              Icons.precision_manufacturing_sharp,
            ),
            tooltip: "Reset",
            onPressed: () {
              // Setzt jeden Servo zurück
              for (var i in slider) {
                i.resetJsonMap();
              }
              // Lädt die App neu um einen GET Request zu senden und die Slider
              // auf den neusten Stand zu bringen
              // Das die App neu läd, bemerkt der User nicht
              Phoenix.rebirth(context);
            },
          ),
          // Button um die SettingsPage zu öffnen
          IconButton(
            icon: Icon(Icons.settings),
            onPressed: () {
              Navigator.push(
                context,
                MaterialPageRoute(builder: (context) => SettingsPage()),
              );
            },
          )
        ],
      ),
      // Eine SingleChildScrollView um das Scrollen der Seite zu ermöglichen.
      // Dies wird gebraucht um die App responsive zu gestalten
      // Eine Column wurde verwendet um die Slider Senkrecht zu plazieren
      body: SingleChildScrollView(
        child: Column(
          children: [slider1, slider2, slider3, slider4, slider5, slider6],
        ),
      ),
    );
  }
}
```

1.2.2 slider.dart

```
import 'dart:convert';
import 'package:flutter/material.dart';
import 'package:http/http.dart' as http;

class ControlSlider extends StatefulWidget {
  final String url = "http://robopi:5000/servo";
  final String name;
  final String identifier;
  Map jsonMap;

  // Ein Konstruktor um diesem Widget Variablen übergeben zu können
  ControlSlider(this.name, this.identifier);

  // Setzt den jeweiligen Servo auf Position 0
  void resetJsonMap() {
    jsonMap = {
      "servos": [
        {"id": identifier, "pos": 0}
      ],
      "teach": []
    };
    putJsonData();
  }

  // Sendet PUT Request
  void putJsonData() {
    String jsonStr = jsonEncode(jsonMap);
    http.put(Uri.encodeFull(url),
        body: jsonStr,
        headers: {"Content-Type": "application/json"}).then((result) {
      print("PUT Status: ${result.statusCode}");
    });
  }

  @override
  _ControlSliderState createState() => _ControlSliderState();
}

class _ControlSliderState extends State<ControlSlider> {
  double currentSliderValue = 0;
  int now;
  int last = 301;
  double nowSlider;
  double lastSlider = 0;
```

```
@override
void initState() {
  super.initState();
  // Beim Laden dieser Seite wird ein GET Request ausgeführt
  getJsonData();
}

// Sendet GET Request und aktualisiert die Slider
void getJsonData() async {
  try {
    var response = await http.get(Uri.encodeFull(widget.url),
        headers: {"Accept": "application/json"});

    // Dekodiert die Daten
    var convertDataToJson = json.decode(response.body);

    // Holt die ID und die Position vom Servo
    var data = convertDataToJson["servos"][int.parse(widget.identifier)];
    var position =
        convertDataToJson["servos"][int.parse(widget.identifier)]["pos"];
    print("$data || Status: ${response.statusCode}");
    // Aktualisiert die Slider
    setState(() {
      currentSliderValue = position.toDouble();
    });
  } catch (e) {
    print("Fehler: $e");
  }
}

// Sendet PUT Request
void putJsonData() {
  String jsonStr = jsonEncode(widget.jsonMap);
  http.put(Uri.encodeFull(widget.url),
      body: jsonStr,
      headers: {"Content-Type": "application/json"}).then((result) {
    print("PUT Status: ${result.statusCode}");
  });
}

// Updatet bei einer Bewegung der Slider die JSON Map
void updateJsonMap() {
  // Speichert aktuelle Zeit und aktueller Slider Wert
  now = DateTime.now().millisecondsSinceEpoch;
  nowSlider = currentSliderValue;
```

```
    setState(() {
      // Aktualisiert die JSON Map
      widget.jsonMap = {
        "servos": [
          {"id": widget.identifier, "pos": currentSliderValue}
        ],
        "teach": []
      };
      var delta = now - last;
      var deltaSlider = nowSlider - lastSlider;
      // Wenn mehr als 80 Millisekunden vergangen sind oder die Bewegung größer
      // als 500 ider kleiner als -500 war sendet er enen PUT Request
      if (delta > 80 || deltaSlider > 500 || deltaSlider < -500) {
        putJsonData();
        lastSlider = currentSliderValue;
        // Startet Timer neu
        last = DateTime.now().millisecondsSinceEpoch;
      }
    });
  }

  @override
  Widget build(BuildContext context) {
    return Container(
      child: Column(
        children: <Widget>[
          // Eine Card um eine schöne Box um den Slider zu setzten
          Card(
            // Hält 20 Pixel Abstand
            margin: EdgeInsets.all(20.0),
            // Sorgt für einen Schatten
            elevation: 4,
            // Rundet die Box ab
            shape:
                RoundedRectangleBorder(borderRadius: BorderRadius.circular(10)),
            color: Colors.grey[70],
            child: Column(
              children: <Widget>[
                Padding(
                  // Hält oben 5 Pixel Abstand um den Text besser darzustellen
                  padding: EdgeInsets.only(top: 5),
                  // Ein Text der die aktuelle Slider Nummer ausgibt
                  child: Text(
                    widget.name,
                    softWrap: false,
                    overflow: TextOverflow.fade,
                  ),
                ),
```

```
                Slider(
                  // Aktueller Wert
                  value: currentSliderValue,
                  min: -1000,
                  max: 1000,
                  onChanged: (double value) {
                    setState(() {
                      // Wenn der Slider bewegt wurde, ändert er die Variable
                      // und führt die Funktion updateJsonMap aus
                      currentSliderValue = value;
                    });
                    updateJsonMap();
                  },
                ),
              ],
            ),
          ),
        ],
      ),
    );
  }
}
```

1.2.3 noConnection.dart

```
import 'package:flutter/material.dart';
import 'package:flutter_phoenix/flutter_phoenix.dart';

// Ein Alert Dialog um den USer zu Warnen,
// falls keine Verbindung zum Server besteht
class NoConnectionDialog extends StatelessWidget {
  @override
  Widget build(BuildContext context) {
    return AlertDialog(
      title: Text("Warning"),
      content: Text("No server connection"),
      actions: [
        FlatButton(
          child: Text("Try Again"),
          onPressed: () {
            Navigator.of(context).pop();
            // App wird neu geladen
            Phoenix.rebirth(context);
          },
        )
      ],
    );
  }
}
```

1.3 Settings Page

1.3.1 settingsPage.dart

```dart
import 'package:flutter/material.dart';
import 'package:robo_app/settings/settingButtons/exampleSequence.dart';
import 'package:robo_app/settings/settingButtons/infoButton.dart';
import 'package:robo_app/settings/settingButtons/resetTeachButton.dart';
import 'package:robo_app/settings/settingButtons/syncButton.dart';
import 'package:robo_app/settings/settingButtons/teachButton.dart';
import 'package:robo_app/settings/settingsTopicText/otherText.dart';
import 'package:robo_app/settings/settingsTopicText/teachText.dart';


// Die SettingsPage, die per Button aufgerufen werden kann

class SettingsPage extends StatelessWidget {
  @override
  Widget build(BuildContext context) {
    return Scaffold(
        appBar: AppBar(
          title: Text("Settings"),
        ),
        // Grey 70 Container, um die Farbe einzustellen
        body: Container(
            color: Colors.grey[70],
            // Eine List View, die das Scrollen ermöglicht
            child: ListView(children: [
              // Eine Column die 7 Widgets Anzeigt
              Column(
                children: [
                  TeachText(),
                  TeachButton(),
                  ResetTeachButton(),
                  ExampleSequence(),
                  OtherText(),
                  SyncButton(),
                  InfoButton()
                ],
              ),
            ])));
  }
}
```

1.3.2 teachButton.dart

```
import 'dart:convert';
import 'package:flutter/material.dart';
import 'package:http/http.dart' as http;
import 'package:shared_preferences/shared_preferences.dart';

class TeachButton extends StatefulWidget {
  @override
  _TeachButtonState createState() => _TeachButtonState();
}

class _TeachButtonState extends State<TeachButton> {
  final String url = "http://robopi:5000/servo";
  bool teaching = false;

  // Speichert den Boolean in den Shared Preferences,
  // damit bei erneutem aufrufen der Seite der Button status gemerkt wird
  addBoolToSP() async {
    SharedPreferences prefs = await SharedPreferences.getInstance();
    prefs.setBool('teaching', teaching);
  }

  // Holt einen Boolean aus den Shared Preferences.
  // Ist dort keiner vorhanden wird false zurückgegeben
  getBoolValuesSP() async {
    SharedPreferences prefs = await SharedPreferences.getInstance();
    // setzt die Variable
    setState(() {
      teaching = prefs.getBool('teaching') ?? false;
    });
  }

  // JSON Map die gesendet wird
  Map jsonMap = {
    "servos": [],
    "teach": [
      {"teaching": true, "run": false, "reset": false, "example": false}
    ]
  };

  // PUT Request
  void putJsonData() {
    String jsonStr = jsonEncode(jsonMap);
    http.put(Uri.encodeFull(url),
        body: jsonStr,
        headers: {"Content-Type": "application/json"}).then((result) {
      print("PUT Status: ${result.statusCode}");
    });
  }
```

```
  @override
  void initState() {
    super.initState();
    // Beim Aufrufen der Seite wir die Methode getBoolValuesSP aufgerufen
    getBoolValuesSP();
  }

  @override
  Widget build(BuildContext context) {
    return Padding(
      padding: EdgeInsets.all(15),
      // Eine Card die 2 List Tiles enthält
      child: Card(
        color: Colors.grey[70],
        shape: RoundedRectangleBorder(borderRadius: BorderRadius.circular(10)),
        child: teaching
            // Wenn teaching true ist wird diese ListTile angezeigt
            ? ListTile(
                leading: Icon(Icons.school),
                title: Text("Stop teaching me"),
                onTap: () {
                  setState(() {
                    // Schickt den PUT Request und speichert die Variable
                    // in den Shared Preferences
                    teaching = false;
                    addBoolToSP();
                    putJsonData();
                  });
                },
              )
            // Wenn teaching false ist wird diese ListTile angezeigt
            : ListTile(
                leading: Icon(Icons.school),
                title: Text("Teach me!"),
                onTap: () {
                  setState(() {
                    // Speichert die VAriable in den Shared Preferences
                    teaching = true;
                    addBoolToSP();
                  });
                },
              ),
      ),
    );
  }
}
```

1.3.3 exampleSequence.dart

```
import 'dart:convert';
import 'package:flutter/material.dart';
import 'package:http/http.dart' as http;

class ExampleSequence extends StatelessWidget {
  final String url = "http://robopi:5000/servo";

  // JSON Map um dem Backend zu sagen, dass das Beispiel ausgeführt werden soll
  final Map jsonMap = {
    "servos": [],
    "teach": [
      {"teaching": false, "run": false, "reset": false, "example": true}
    ]
  };

  // PUT Request
  void putJsonData() {
    String jsonStr = jsonEncode(jsonMap);
    http.put(Uri.encodeFull(url),
        body: jsonStr,
        headers: {"Content-Type": "application/json"}).then((result) {
      print("PUT Status: ${result.statusCode}");
    });
  }

  @override
  Widget build(BuildContext context) {
    return Padding(
      // 15 Pixel Abstand
      padding: EdgeInsets.all(15),
      // Grey 70 Card um eine Box zu erschaffen die eine ListTile enthält
      child: Card(
        color: Colors.grey[70],
        shape: RoundedRectangleBorder(borderRadius: BorderRadius.circular(10)),
        child: ListTile(
          leading: Icon(Icons.crop_din),
          title: Text("Example Sequence"),
          onTap: () {
            putJsonData();
          },
        ),
      ),
    );
  }
}
```

1.3.4 resetTeachButton.dart

```
import 'dart:convert';
import 'package:flutter/material.dart';
import 'package:http/http.dart' as http;

class ResetTeachButton extends StatelessWidget {
  final String url = "http://robopi:5000/servo";

  // JSON Map die gesendet wird
  final Map jsonMap = {
    "servos": [],
    "teach": [
      {"teaching": false, "run": false, "reset": true, "example": false}
    ]
  };

  // PUT Request
  void putJsonData() {
    String jsonStr = jsonEncode(jsonMap);
    http.put(Uri.encodeFull(url),
        body: jsonStr,
        headers: {"Content-Type": "application/json"}).then((result) {
      print("PUT Status: ${result.statusCode}");
    });
  }

  @override
  Widget build(BuildContext context) {
    return Padding(
      padding: EdgeInsets.all(15),
      // Eine Card mit einer ListTile, die einen Button und einen Text enthält
      child: Card(
        color: Colors.grey[70],
        shape: RoundedRectangleBorder(borderRadius: BorderRadius.circular(10)),
        child: ListTile(
          leading: Icon(Icons.restore),
          title: Text("Reset Teaching"),
          onTap: () {
            putJsonData();
          },
        ),
      ),
    );
  }
}
```

1.3.5 syncButton.dart

```dart
import 'package:flutter/material.dart';
import 'package:flutter_phoenix/flutter_phoenix.dart';

class SyncButton extends StatelessWidget {
  @override
  Widget build(BuildContext context) {
    return Padding(
      padding: EdgeInsets.all(15),
      // Eine Card mit einer ListTile, die einen Button und einen Text enthält
      child: Card(
        color: Colors.grey[70],
        shape: RoundedRectangleBorder(borderRadius: BorderRadius.circular(10)),
        child: ListTile(
          leading: Icon(Icons.autorenew),
          title: Text("Sync"),
          onTap: () {
            // Lädt die App neu
            Phoenix.rebirth(context);
          },
        ),
      ),
    );
  }
}
```

1.3.6 infoButton.dart

```dart
import 'package:flutter/material.dart';

class InfoButton extends StatelessWidget {
  @override
  Widget build(BuildContext context) {
    return Padding(
      padding: EdgeInsets.all(15),
      // Eine Card mit einer ListTile, die einen Button und einen Text enthält
      child: Card(
        color: Colors.grey[70],
        shape: RoundedRectangleBorder(borderRadius: BorderRadius.circular(10)),
        child: ListTile(
          leading: Icon(Icons.info),
          title: Text("Developer:  Florian Schmitz"),
        ),
      ),
    );
  }
}
```

1.4 pubspec.yaml

```yaml
name: robo_app
description: A robot control app
publish_to: 'none'
version: 1.0.0+1

environment:
  sdk: ">=2.7.0 <3.0.0"

dependencies:
  flutter:
    sdk: flutter
  cupertino_icons: ^1.0.0
  http: ^0.12.2
  flutter_phoenix: ^0.1.0
  shared_preferences: ^0.5.12+4

dev_dependencies:
  flutter_test:
    sdk: flutter
  flutter_launcher_icons: ^0.8.1

flutter_icons:
  android: true
  ios: false
  image_path: "assets/images/robot_launcher_icon.png"

flutter:
  uses-material-design: true

  assets:
    - assets/images/
```

2 Backend Code

2.1 Prototypen

2.1.1 servotester.py

```
from __future__ import division
from time import sleep
import Adafruit_PCA9685

# Initalisierung der Adafruit Bibliothek
pwm = Adafruit_PCA9685.PCA9685(address=0x41)

# Neutrale Position des Servos
pulse_neutral_µs = 1500

# Hilfsfunktion - erwartet Pulsdauer in Mikrosekunden und bewegt dann den Servo
def set_servo_pulse(channel, pulse):
    pulse_length = 1000000
    pulse_length /= 50
    pulse_length /= 4096
    pulse /= pulse_length
    pulse = round(pulse)
    pulse = int(pulse)
    pwm.set_pwm(channel, 0, pulse)


# Frequenz auf 50Hz setzen
pwm.set_pwm_freq(50)

while True:
    #  Bewege Servos 0-15 in neutral Position
    for i in range(0, 16):
        set_servo_pulse(i, pulse_neutral_µs)
        sleep(0.2)

    print("Bewege Servo an Kanal 0-15 in neutral Position")

    answer = input("Wähle Servokanal aus oder Q für Beenden: ")
    if answer.lower() == "q":
        exit()

    else:
        try:
            channel = int(answer)
```

```
            while True:
                # Erwate Pulsweite (500 bis -2500)
                answer = input(
                    "Pulslänge eingeben in µs oder Q eingeben, um zur Servoauswahl
zurück zu gelangen: ")

                # Kehre zur Servoauswahl zurück
                if answer.lower() == "q":
                    break

                else:
                    try:
                        # Setzte Servoposition
                        pulse = int(answer)
                        set_servo_pulse(channel, pulse)

                    except ValueError:
                        print(
                            "Ungültige Eingabe! Bitte eine Zahl zwischen 500-2500
eingeben!")

        except ValueError:
            print("Ungültige Eingabe! Bitte eine Zahl zwischen 0-15 eingeben!")
```

2.1.2 servoKlassePrototyp

```
import Adafruit_PCA9685
from time import sleep


class Servo():
    # Standartwerte, da noch keine Konfigurationsdatei eingelesen wird
    ID_DEFAULT = 0
    PWM_MIN_LIMIT = 500
    PWM_MIN_DEFAULT = 1000
    PWM_NEUTRAL_DEFAULT = 1500
    PWM_MAX_LIMIT = 2500
    PWM_MAX_DEFAULT = 2000
    POS_MIN_DEFAULT = -1000
    POS_MAX_DEFAULT = 1000
    POS_NEUTRAL_DEFAULT = 0

    def __init__(self, servo_id=ID_DEFAULT, pwm_min=PWM_MIN_DEFAULT,
pwm_max=PWM_MAX_DEFAULT, pwm_neutral=PWM_NEUTRAL_DEFAULT, pos_min=POS_MIN_DEFAULT,
pos_max=POS_MAX_DEFAULT, pos_neutral=POS_NEUTRAL_DEFAULT):
        # Initalisierung der Adafruit Bibliothek
        self._pwm_controller = Adafruit_PCA9685.PCA9685(address=0x41)

        # Frequenz auf 50Hz setzen
        self._pwm_controller.set_pwm_freq(50)

        # Werte in versteckte Variablen schreiben
        # Diese sollen im späteren Verlauf nur durch Funktionen
        # ansprechbar sein
        self._id = servo_id
        self._pwm_min = pwm_min
        self._pwm_max = pwm_max
        self._pwm_neutral = pwm_neutral
        self._pos_min = pos_min
        self._pos_max = pos_max
        self._pos_neutral = pos_neutral

    # Hilfsfunktion - erwartet Pulsdauer in Mikrosekunden und bewegt dann den Servo
    def set_servo_pulse(self, channel, pulse):
        pulse_length = 1000000
        pulse_length /= 50
        pulse_length /= 4096
        pulse /= pulse_length
        pulse = round(pulse)
        pulse = int(pulse)
        self._pwm_controller.set_pwm(channel, 0, pulse)
```

```
    # Setzte alle Servos auf ihre Standartposition zurück
    def reset(self):
        for i in range(0, 15):
            self.set_servo_pulse(i, 1500)
            # Warte etwas, da nicht alle Servos gleichzeitig sich bewegen können
            # Da zu wenig Strom da ist um alle Servos zu bewegen
            # würde sonst der Servo in einander Brechen
            sleep(0.5)
```

2.1.3 servoKlasseStart.py

```
import Adafruit_PCA9685
import servoKlassePrototyp

# Initalisierung der Adafruit Bibliothek
pwm_controller = Adafruit_PCA9685.PCA9685(address=0x41)


# Frequenz auf 50Hz setzen
pwm_controller.set_pwm_freq(50)

# Servo Prototyp Klassen Instanz erzeugen
servo = servoKlassePrototyp.Servo(pwm_controller)

# Servo auf Position 500 seztzen
servo.set_servo_pulse(500)
```

2.2 apiPrototyp.py

```
from flask import Flask, request, Response
from flask_restful import Api, Resource
import logging
from servoKlassePrototyp import Servo

# Erstelle ein JSON Dictionary
servo_data = {
    "servos": []
}


class PrototypApiServer(Resource):

    def __init__(self):
        logging.basicConfig(filename='apiPrototyp.log',
                            level=logging.DEBUG)

        # Erzeuge Servo Instanz
        self.servo_instanz = Servo()

        # Dem JSON Dictionary 6 Servos hinzufügen mit den jeweiligen Werten
        for i in range(0, 6):
            servo_data["servos"].append(
                {
                    "id": str(i),
                    "pos_min": -1000,
                    "pos_max": 1000,
                    "pos_neutral": 0,
                    "pos": 0
                }
            )

    # Gibt ein JSON Dictionary zurück
    def get(self):
        logging.debug("get Methode aufgerufen")
        return servo_data, 200

    def put(self):
        logging.debug("put Method aufgerufen")
        # Holt das JSON Objekt was durch den PUT Befehl geschickt wurde
        resource = request.json

        # Holt sich das Abteil servos
        data_array = resource["servos"]
```

```
        # Geht alle Servos durch
        for data in data_array:

            # Holt sich die ID
            servo_id = data["id"]
            found = False

            # Holt sich das Lokale JSON Dictionary und geht durch das Abteil servos
            for servo in servo_data["servos"]:

                # Holt sich die ID
                servo_id_data = servo["id"]

                # Wenn die Lokale ID mit der gesendeten ID übereinstimmt,
                # wird die Variable found auf True gesetzt
                if servo_id_data == servo_id:
                    found = True
                    break

            if found == True:
                # Holt sich die gesendete Position
                new_pos = data["pos"]

                if new_pos is not None:
                    # Holt sich die Lokale Position
                    old_pos = servo["pos"]

                    if new_pos != old_pos:

                        # Ersetzt die Lokale Position durch die gesendete Position
                        servo["pos"] = new_pos

                        # Setzt Servo mit der richtigen ID auf die neue Position
                        self.servo_instanz.set_servo_pulse(
                            int(servo_id_data), new_pos)

        # Gibt das JSON Dictionary zurück
        resource = servo_data
        return resource, 200

# erzeuge Flask und Api Objekt
app = Flask(__name__)
api = Api(app)

# Erstelle Endpoint
api.add_resource(PrototypApiServer, '/servo')

# Startet Flask Server. Nicht auf 127.0.0.1 sonst ist der Server nicht von anderen
Geräten erreichbar
app.run(host='0.0.0.0', port=5000)
```

2.3 Testfälle

2.3.1 tests.py

```
import unittest
from test_servo import TestServo
from test_api import TestApi

# Die Tests werden automatisch gefunden und ausgeführt

if __name__ == '__main__':
    unittest.main(verbosity=2)
```

2.3.2 test_servo.py

```
import unittest
import Adafruit_PCA9685
import RPi.GPIO as GPIO
from time import sleep
from backend.servo import Servo

# Testet die Servo Klasse. Es Testet alle get und alle set Methoden

class TestServo(unittest.TestCase):
    SLEEP_S = 1

    def setUp(self):
        self.pwm = Adafruit_PCA9685.PCA9685(address=0x41)
        self.pwm.set_pwm_freq(50)

    def test_init(self):
        servo = Servo(self.pwm)
        self.assertIsInstance(servo, Servo, "Bei der Initialisierung ist was
schiefgelaufen")

    def test_id(self):
        servo = Servo(self.pwm, 1)
        servo_id = servo.get_id()
        self.assertEqual(servo_id, 1, "Die ID stimmt nicht")

    def test_pwm_min(self):
        servo = Servo(self.pwm)
        pwm_min = servo.get_pwm_min()
        self.assertEqual(pwm_min, 1000, "pwm_min stimmt nicht")

        servo.set_pwm_min(600)
        pwm_min = servo.get_pwm_min()
        self.assertEqual(pwm_min, 600, "pwm_min stimmt nicht")

    def test_pwm_max(self):
        servo = Servo(self.pwm)
        pwm_max = servo.get_pwm_max()
        self.assertEqual(pwm_max, 2000, "pwm_max stimmt nicht")

        servo.set_pwm_max(2400)
        pwm_max = servo.get_pwm_max()
        self.assertEqual(pwm_max, 2400, "pwm_max stimmt nicht")
```

```
    def test_pwm_neutral(self):
        servo = Servo(self.pwm)
        pwm_neutral = servo.get_pwm_neutral()
        self.assertEqual(pwm_neutral, 1500, "pwm_neutral stimmt nicht")

        servo.set_pwm_neutral(1000)
        pwm_neutral = servo.get_pwm_neutral()
        self.assertEqual(pwm_neutral, 1000, "pwm_neutral stimmt nicht")

    def test_pwm(self):
        servo = Servo(self.pwm)
        pwm = servo.get_pwm()
        self.assertEqual(pwm, 1500, "pwm stimmt nicht")

        servo.set_pwm(2000)
        pwm = servo.get_pwm()
        sleep(TestServo.SLEEP_S)
        self.assertEqual(pwm, 2000, "pwm stimmt nicht")

        servo.set_pwm(1500)
        pwm = servo.get_pwm()
        sleep(TestServo.SLEEP_S)
        self.assertEqual(pwm, 1500, "pwm stimmt nicht")

    def test_reset(self):
        servo = Servo(self.pwm)
        servo.set_pwm(1000)
        sleep(TestServo.SLEEP_S)
        pwm = servo.get_pwm()
        self.assertEqual(pwm, 1000, "pwm stimmt nicht")

        servo.reset()
        sleep(TestServo.SLEEP_S)
        pwm = servo.get_pwm()
        self.assertEqual(pwm, 1500, "pwm stimmt nicht")

    def test_pos_min(self):
        servo = Servo(self.pwm)
        pos_min = servo.get_pos_min()
        self.assertEqual(pos_min, -1000, "pos_min stimmt nicht")

        servo.set_pos_min(-900)
        pos_min = servo.get_pos_min()
        self.assertEqual(pos_min, -900, "pos_min stimmt nicht")
```

```
    def test_pos_max(self):
        servo = Servo(self.pwm)
        pos_max = servo.get_pos_max()
        self.assertEqual(pos_max, 1000, "pos_max stimmt nicht")

        servo.set_pos_max(900)
        pos_max = servo.get_pos_max()
        self.assertEqual(pos_max, 900, "pos_max stimmt nicht")

    def test_pos_neutral(self):
        servo = Servo(self.pwm)
        pos_neutral = servo.get_pos_neutral()
        self.assertEqual(pos_neutral, 0, "pos_neutral stimmt nicht")

        servo.set_pos_neutral(100)
        pos_neutral = servo.get_pos_neutral()
        self.assertEqual(pos_neutral, 100, "pos_neutral stimmt nicht")

    def test_pos(self):
        servo = Servo(self.pwm)
        pos = servo.get_pos()
        self.assertEqual(pos, 0, "pos stimmt nicht")

        servo.set_pos(-1000)
        sleep(TestServo.SLEEP_S)
        pos = servo.get_pos()
        self.assertEqual(pos, -1000, "pos stimmt nicht")

        servo.set_pos(0)
        sleep(TestServo.SLEEP_S)
        pos = servo.get_pos()
        self.assertEqual(pos, 0, "pos stimmt nicht")
```

2.3.3 test_api.py

```
import unittest
import requests
from backend.roboter import Roboter
from time import sleep
import multiprocessing
import json

# Es Testet das Bewegen der Servos mit der PWM und der Umrechnung
# Aus Zeitgründen wurden die Teach Features nicht getestet

class TestApi(unittest.TestCase):
    SLEEP_S = 1
    URL = "http://robopi:5000/servo"

    def setUp(self):
        self.roboter = Roboter
        self.process = multiprocessing.Process(target=self.roboter)
        self.process.start()
        sleep(TestApi.SLEEP_S)

    def tearDown(self):
        self.process.terminate()

    def test_get(self):
        response = requests.get(TestApi.URL)
        self.assertEqual(response.status_code, 200,
                         "Beim get Befehl ist was schiefgelaufen")

        response_body = response.json()
        for i in range(0, 16):
            self.assertEqual(
                response_body["servos"][i]["id"], i, "Die ID stimmt nicht")
            self.assertEqual(
                response_body["servos"][i]["pos"], 0, "Die Position stimmt nicht")
```

```
    def test_put(self):
        servo_data = {"servos": [{"id": 0, "pos": 500}], "teach": []}
        response = requests.put(
            TestApi.URL, json=servo_data)
        self.assertEqual(response.status_code, 200,
                         "Beim put Befehl ist was schiefgelaufen")

        sleep(TestApi.SLEEP_S)

        response = requests.get(TestApi.URL)
        self.assertEqual(response.status_code, 200,
                         "Beim get Befehl ist was schiefgelaufen")

        response_body = response.json()
        self.assertEqual(
            response_body["servos"][0]["pos"], 500, "Die Position stimmt nicht")
```

2.4 Backend

2.4.1 main.py

```
#!/usr/bin/python3

from backend.roboter import Roboter

# Startet die Roboter Klasse
Roboter()
```

2.4.2 servo.py

```
class Servo():
    # Standartwerte der Servos
    SERVO_ID = 0
    PWM_MIN_LIMIT = 500
    PWM_MIN_DEFAULT = 1000
    PWM_NEUTRAL_DEFAULT = 1500
    PWM_MAX_LIMIT = 2500
    PWM_MAX_DEFAULT = 2000
    POS_MIN_DEFAULT = -1000
    POS_MAX_DEFAULT = 1000
    POS_NEUTRAL_DEFAULT = 0

    def __init__(self, pwm_controller, servo_id=SERVO_ID, pwm_min=PWM_MIN_DEFAULT,
pwm_max=PWM_MAX_DEFAULT, pwm_neutral=PWM_NEUTRAL_DEFAULT, pos_min=POS_MIN_DEFAULT,
pos_max=POS_MAX_DEFAULT, pos_neutral=POS_NEUTRAL_DEFAULT):

        # Werte in versteckte Variablen schreiben
        # Diese sollen nur durch Funktionen ansprechbar sein
        self._pwm_controller = pwm_controller
        self._servo_id = servo_id
        self._pwm_min = pwm_min
        self._pwm_max = pwm_max
        self._pwm_neutral = pwm_neutral
        self._pos_min = pos_min
        self._pos_max = pos_max
        self._pos_neutral = pos_neutral
        self._pos_current = self._pos_neutral
        self._pwm_current = self._pwm_neutral

        # Setzte Servo auf Standartposition
        self.set_pwm(self._pwm_current)

    # Die get_... Methoden geben die Variable zurück
    # Die set_... setzt die Varibale

    def get_id(self):
        return self._servo_id

    def get_pwm_min(self):
        return self._pwm_min

    def set_pwm_min(self, pwm_min):
        if pwm_min >= Servo.PWM_MIN_LIMIT and pwm_min < self._pwm_max:
            self._pwm_min = pwm_min
        else:
            raise ValueError("Ungültiger pwm_min Parameterwert")

    def get_pwm_max(self):
        return self._pwm_max
```

```
    def set_pwm_max(self, pwm_max):
        if pwm_max > self._pwm_min and pwm_max <= Servo.PWM_MAX_LIMIT:
            self._pwm_max = pwm_max
        else:
            raise ValueError("Ungültiger pwm_max Parameterwert")

    def get_pwm_neutral(self):
        return self._pwm_neutral

    def set_pwm_neutral(self, pwm_neutral):
        if pwm_neutral >= self._pwm_min and pwm_neutral <= self._pwm_max:
            self._pwm_neutral = pwm_neutral
        else:
            raise ValueError("Ungültiger pwm_neutral Parameterwert")

    def get_pwm(self):
        return self._pwm_current

    def set_pwm(self, pwm):
        if pwm >= self._pwm_min and pwm <= self._pwm_max:
            # Berechne Pulslänge pro Bit | von dem Beispielcode der Adafruit
Bibliothek
            pulse_length_per_bit = 1000000 / 50 / 4096
            pulse = int(round(pwm / pulse_length_per_bit))
            self._pwm_controller.set_pwm(self._servo_id, 0, pulse)
            self._pwm_current = pwm
        else:
            raise ValueError("Ungültiger pwm Parameterwert")

    def reset(self):
        # Setzt Servo auf die Standartposition zurück
        self.set_pwm(self._pwm_neutral)

    def get_pos_min(self):
        return self._pos_min

    def set_pos_min(self, pos_min):
        if pos_min < self._pos_max:
            self._pos_min = pos_min
        else:
            raise ValueError("Ungültiger pos_min Parameterwert")

    def get_pos_max(self):
        return self._pos_max

    def set_pos_max(self, pos_max):
        if pos_max > self._pos_min:
            self._pos_max = pos_max
        else:
            raise ValueError("Ungültiger pos_max Parameterwert")

    def get_pos_neutral(self):
```

```
        return self._pos_neutral

    def set_pos_neutral(self, pos_neutral):
        if pos_neutral >= self._pos_min and pos_neutral <= self._pos_max:
            self._pos_neutral = pos_neutral
        else:
            raise ValueError("Ungültiger pos_neutral Parameterwert")

    def _calculate_pwm(self, x):
        # Fallunterscheidung erforderlich zwischen Werten kleiner pos_neutral und
größer pos_neutral, da die Steigung unterschiedlich sein kann
        if x < self._pos_neutral:
            # Lineare Gleichung: y = m * x + t
            # Bestimmte die Steigung m:
            #   m = y2 - y1 / x2 - x1
            m = (self._pwm_neutral - self._pwm_min) / \
                (self._pos_neutral - self._pos_min)
            # Bestimme t:
            #   t = y - m * x
            t = self._pwm_neutral - m * self._pos_neutral
            # Berechne das Ergebnis
            y = m * x + t
        elif x == self._pos_neutral:
            y = self._pwm_neutral
        else:
            # Lineare Gleichung: y = m * x + t
            # Bestimmte die Steigung m:
            #   m = y2 - y1 / x2 - x1
            m = (self._pwm_max - self._pwm_neutral) / \
                (self._pos_max - self._pos_neutral)
            # Bestimme t:
            #   t = y - m * x
            t = self._pwm_neutral - m * self._pos_neutral
            # Berechne das Ergebnis
            y = m * x + t
        y = round(y)
        y = int(y)
        return y

    def get_pos(self):
        return self._pos_current
```

```
    def set_pos(self, pos):
        # Rechne den Wert um und setzte den Servo auf diese Positon
        if pos >= self._pos_min and pos <= self._pos_max:
            self._pos_current = pos
            new_pwm = self._calculate_pwm(pos)
            self.set_pwm(new_pwm)
        else:
            raise ValueError("ValueError: Ungültiger pos Parameterwert:
pos_requested: {pos_requested}, pos_current: {pos_current}, pos_min: {pos_min},
pos_max: {pos_max}".format(
                pos_requested=pos,
                pos_current=self._pos_current,
                pos_min=self._pos_min,
                pos_max=self._pos_max))
```

2.4.3 roboter.py

```
from backend.roboterResource import RoboterResource
from flask import Flask
from flask_restful import Api
from backend.servo import Servo


class Roboter():

    DEFAULT_API_BIND_ADDRESS = "0.0.0.0"
    DEFAULT_API_PORT = 5000

    def __init__(self):

        # Erzeugt Flask und Api Objekte
        app = Flask(__name__)
        api = Api(app)

        # Erstellt Endpoint
        api.add_resource(RoboterResource, '/servo')

        # Startet Flask Server
        app.run(host=Roboter.DEFAULT_API_BIND_ADDRESS,
                port=Roboter.DEFAULT_API_PORT, debug=False)
```

2.4.4 roboterResource.py

```
from flask_restful import Resource
from flask import request
# Muss mit backend. importiert werden, da sonst die Unittests die Dateien nicht
findet
from backend.servo import Servo
from backend.sequence import Sequence
import Adafruit_PCA9685
import json
from time import sleep


class RoboterResource(Resource):
    _instance = None

    # Diese Methode von der Singelton Klasse wird benötigt, da in dieser Klasse bei
jedem
    # Request, die __init__ aufruft. Dies ist nicht gewollt, da viele Sachen nur
einmal initialisiert werden dürfen.
    def __new__(cls):
        if cls._instance is None:
            cls._instance = super().__new__(cls)
            cls.servo_liste = []
            cls.teaching_list = []
            cls._pwm = Adafruit_PCA9685.PCA9685(address=0x41)
            cls._pwm.set_pwm_freq(50)
            cls.sequence = Sequence()

            # Konfigurationsdatei in config_file speichern
            config_file = open("backend/roboconfig.json")

            # Konfig einlesen
            cls._config = json.load(config_file)

            if config_file is not None:
                # Datei wird wieder geschlossen
                config_file.close()

                # In der Konfig den Abschnitt servos lesen
                servos_config = cls._config["servos"]

                # Alle Servos durchgehen
                for i in servos_config:

                    # Config vom ausgewählten Servo durchgehen
                    config = servos_config[i]
```

```
                    # Der Servo der Servoliste hinzufügen
                    cls.servo_liste.append(
                        Servo(cls._pwm,
                              # Dem Servo die Konfig Daten übergeben
                              servo_id=config["id"],
                              pwm_min=config["pwm_min"],
                              pwm_max=config["pwm_max"],
                              pwm_neutral=config["pwm_neutral"],
                              pos_min=config["pos_min"],
                              pos_max=config["pos_max"],
                              pos_neutral=config["pos_neutral"],)
                    )
                # Alle Servos zurücksetzten
                for servo in cls.servo_liste:
                    servo.reset()

        # Falls Klasse erstellt wurde, wird keine neue erzeugt sondern die erzeugte
zurückgegeben
        return cls._instance

    def _get_servo_data(self):

        # Ertsellt JSON Dictionary
        resource = {
            "servos": [],
            "teach": []
        }

        # Fügt dem JSON Dictionary genau so viele Servos hinzu, wie in der Liste
vorhanden sind
        for servo in self.servo_liste:
            resource["servos"].append(
                {
                    "id": servo.get_id(),
                    "pos": servo.get_pos()
                }
            )

        # Fügt dem JSON Dictionary die Variablen hinzu
        resource["teach"].append(
            {
                "teaching": False,
                "run": False,
                "reset": False,
                "example": False
            }
        )

        return resource
```

```
    def get(self):
        # Gibt das JSON Dictionary zurück
        data = self._get_servo_data()
        return data, 200

    def put(self):
        # Holt sich das gesendete JSON Objekt und geht in den Abteil servos
        resource = request.json
        servo_data = resource["servos"]

        for data in servo_data:
            # Holt sich die ID und Position
            servo_id = int(data["id"])
            servo_position = data["pos"]

            # Holt sich die Servos aus der Servoliste
            for servo in self.servo_liste:

                # Gleicht ID ab
                if servo.get_id() == servo_id:

                    if servo_position != servo.get_pos():
                        # Setzte Position des Servos
                        servo.set_pos(servo_position)

        # geht in den Abteil teach
        teach = resource["teach"]

        # Holt sich den Inhalt von teach
        for teach_answer in teach:

            if teach_answer["teaching"] == True:
                teach_answer["teaching"] = False
                liste = []
```

```
                for servo in self.servo_liste:
                    # Holt von jedem Servo die ID und Position
                    servo_get_id = servo.get_id()
                    servo_get_pos = servo.get_pos()

                    # Fügt dies der liste hinzu
                    liste.append(servo_get_pos)

                # Fügt die liste der Positionen der teaching_list hinzu.
                # Dies wird getan, da der Roboter so Schritte bekommt, die der
ausführen kann.
                # Jeder Schritt hat dann eine Liste mit allen Positionen der Servos
                self.teaching_list.append(liste)

            if teach_answer["run"] == True:
                teach_answer["run"] = False

                # Holt sich jeden Schritt der Sequenz
                for sequence in self.teaching_list:

                    # Pro Schritt in der Sequenz einen eigenen Counter
                    counter = 0

                    # Wartet nach jedem Schritt 0.7 Sekunden
                    sleep(0.7)

                    # Holt sich die Servopositionen des aktuellen Schrittes
                    for position in sequence:

                        # Der Counter zählt als Servo ID, da man keine weitere for
i in liste schleife machen konnte.
                        # Sonst wäre eine Position bei jedem Servo eingetragen
                        if counter <= 15:
                            # Holt sich den aktuellen Servo und setzt die Position
                            servo = self.servo_liste[counter]
                            servo.set_pos(position)
                            counter += 1

            if teach_answer["reset"] == True:
                teach_answer["reset"] = False
                # Überschreibt die teaching_list und somit sind alle Schritte, die
dem Roboter beigebracht wurden gelöscht
                self.teaching_list = []
```

```
            if teach_answer["example"] == True:
                teach_answer["example"] = False

                # Holt sich die Sequenz
                example_data = self.sequence.sequence_1()

                # Wandelt das JSON Onjekt in eine Liste, um ein gewisses Element
aus der Liste zu erhalten.
                # Ein Dictionary hat nämlich kein Index
                keys = list(example_data["sequence_1"].keys())

                # Holt sich die verschiedenen Schritte
                for i in keys:
                    position = example_data["sequence_1"][i]

                    for i in position:
                        # Holt ID und Position um den richtigen Servo zu
Positionieren
                        servo_position_id = i["id"]
                        servo_sequence_position = i["pos"]

                        servo_object = self.servo_liste[servo_position_id]
                        servo_object.set_pos(servo_sequence_position)

                        # Wartet nach jeder bewegung 0.7 Sekunden, da nicht alle
Servos gleichzeitig bewegt werden dürfen
                        sleep(0.7)

        # Gibt JSON Dictionary zurück
        resource = self._get_servo_data()
        return resource, 200
```

2.4.5 sequence.py

```
class Sequence():

    def sequence_1(self):
        # Eine Abfolge von Positionen, die der Roboter ausführen kann
        json_dict = {"sequence_1": {"step_1": [{"id": 0,
                                                "pos": 0},
                                               {"id": 1,
                                                "pos": 0},
                                               {"id": 2,
                                                "pos": 0},
                                               {"id": 3,
                                                "pos": 0},
                                               {"id": 4,
                                                "pos": 0},
                                               {"id": 5,
                                                "pos": 0}
                                               ],
                                           "step_2": [{"id": 0,
                                                       "pos": 1000},
                                                      {"id": 1,
                                                       "pos": 350},
                                                      {"id": 5,
                                                       "pos": 620},
                                                      {"id": 2,
                                                       "pos": -590},
                                                      {"id": 1,
                                                       "pos": 530},
                                                      {"id": 5,
                                                       "pos": 23}
                                                      ],

                                           "step_3": [{"id": 1,
                                                       "pos": 0},
                                                      {"id": 2,
                                                       "pos": -310},
                                                      {"id": 3,
                                                       "pos": 230},
                                                      {"id": 4,
                                                       "pos": 1000},
                                                      {"id": 0,
                                                       "pos": -570},
                                                      {"id": 2,
                                                       "pos": -590},
                                                      {"id": 3,
                                                       "pos": 0},
                                                      {"id": 1,
                                                       "pos": 500},
                                                      {"id": 5,
                                                       "pos": 620}
                                                      ],
```

```
                                    "step_4": [{"id": 1,
                                                "pos": 0},
                                               {"id": 2,
                                                "pos": 0},
                                               {"id": 0,
                                                "pos": 0},
                                               {"id": 3,
                                                "pos": 0},
                                               {"id": 4,
                                                "pos": 0},
                                               {"id": 5,
                                                "pos": 0}
                                               ]}}

        return json_dict
```

2.4.6 roboconfig.json


```
{
    "servos": {
        "0": {
            "id": 0,
            "pwm_min": 600,
            "pwm_max": 2400,
            "pwm_neutral": 1500,
            "pos_min": -1000,
            "pos_max": 1000,
            "pos_neutral": 0
        },
        "1": {
            "id": 1,
            "pwm_min": 1000,
            "pwm_max": 2300,
            "pwm_neutral": 1550,
            "pos_min": -1000,
            "pos_max": 1000,
            "pos_neutral": 0
        },
        "2": {
            "id": 2,
            "pwm_min": 600,
            "pwm_max": 2400,
            "pwm_neutral": 1450,
            "pos_min": -1000,
            "pos_max": 1000,
            "pos_neutral": 0
        },
        "3": {
            "id": 3,
            "pwm_min": 1100,
            "pwm_max": 2400,
            "pwm_neutral": 1500,
            "pos_min": -1000,
            "pos_max": 1000,
            "pos_neutral": 0
        },
        "4": {
            "id": 4,
            "pwm_min": 600,
            "pwm_max": 2350,
            "pwm_neutral": 1500,
            "pos_min": -1000,
            "pos_max": 1000,
            "pos_neutral": 0
        },
```

```
"5": {
    "id": 5,
    "pwm_min": 1400,
    "pwm_max": 2400,
    "pwm_neutral": 1500,
    "pos_min": -1000,
    "pos_max": 1000,
    "pos_neutral": 0
},
"6": {
    "id": 6,
    "pwm_min": 1000,
    "pwm_max": 2000,
    "pwm_neutral": 1500,
    "pos_min": -1000,
    "pos_max": 1000,
    "pos_neutral": 0
},
"7": {
    "id": 7,
    "pwm_min": 1000,
    "pwm_max": 2000,
    "pwm_neutral": 1500,
    "pos_min": -1000,
    "pos_max": 1000,
    "pos_neutral": 0
},
"8": {
    "id": 8,
    "pwm_min": 1000,
    "pwm_max": 2000,
    "pwm_neutral": 1500,
    "pos_min": -1000,
    "pos_max": 1000,
    "pos_neutral": 0
},
"9": {
    "id": 9,
    "pwm_min": 1000,
    "pwm_max": 2000,
    "pwm_neutral": 1500,
    "pos_min": -1000,
    "pos_max": 1000,
    "pos_neutral": 0
},
```

```
"10": {
    "id": 10,
    "pwm_min": 1000,
    "pwm_max": 2000,
    "pwm_neutral": 1500,
    "pos_min": -1000,
    "pos_max": 1000,
    "pos_neutral": 0
},
"11": {
    "id": 11,
    "pwm_min": 1000,
    "pwm_max": 2000,
    "pwm_neutral": 1500,
    "pos_min": -1000,
    "pos_max": 1000,
    "pos_neutral": 0
},
"12": {
    "id": 12,
    "pwm_min": 1000,
    "pwm_max": 2000,
    "pwm_neutral": 1500,
    "pos_min": -1000,
    "pos_max": 1000,
    "pos_neutral": 0
},
"13": {
    "id": 13,
    "pwm_min": 1000,
    "pwm_max": 2000,
    "pwm_neutral": 1500,
    "pos_min": -1000,
    "pos_max": 1000,
    "pos_neutral": 0
},
```

```
        "14": {
            "id": 14,
            "pwm_min": 1000,
            "pwm_max": 2000,
            "pwm_neutral": 1500,
            "pos_min": -1000,
            "pos_max": 1000,
            "pos_neutral": 0
        },
        "15": {
            "id": 15,
            "pwm_min": 1000,
            "pwm_max": 2000,
            "pwm_neutral": 1500,
            "pos_min": -1000,
            "pos_max": 1000,
            "pos_neutral": 0
        }
    }
}
```