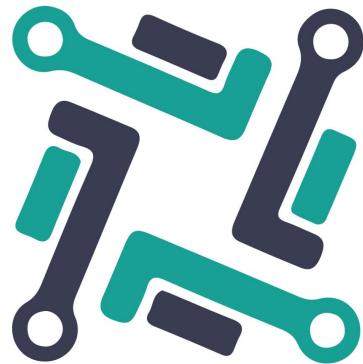




Labyrinth-Löse-Roboter

15.08.2020 - 07.01.2021



Labirinto

Author: Adrian ROSSER - Audric STRÜMLER
Promotion Descartes 2018-2022

Professoren: Silvan WIRTH - Patrick GRUBERT



1 Vorwort

Wir bedanken uns bei Herrn Wirth und Herrn Grubert für ihre kompetente Beratung und Unterstützung bei unserem Projekt bedanken. Ihre Verfügbarkeit, Ihren Rat und Ihre Hilfe während der verschiedenen Teile unseres Projekts haben wir sehr geschätzt. Weiter bedanken wir uns auch speziell für das gute Krisenmanagement während der zweiten Coronavirus-Welle bei ihnen und allen Studiengangleitern. Zum Schluss danken wir allen Klassenkameraden der Klasse Descartes für den kompetitiven und regen Austausch zwischen den Projektteams und die gegenseitige Hilfe während der Coronavirus Pandemie.

Durch die anhaltende Coronavirus Pandemie und die damit verbundene Umstellung des Schulbetriebes auf Fernunterricht wurde die Durchführung der Projektarbeit erschwert. Das binationale Projektteam (F/CH) musste seine Arbeit ebenfalls komplett auf Homeoffice umstellen und konnte nicht mehr physisch gemeinsam am Projekt arbeiten, da zwischen den Wohnorten der beiden Teammitglieder 480km Luftlinie liegen.

Im vergangenen Semester musste ebenfalls eine Projektarbeit unter vergleichbaren Bedingungen erarbeitet werden. Die dabei gewonnenen Erkenntnisse wurden diesmal von Beginn an die Vorgehensweise implementiert, sodass die Arbeit ohne grössere organisatorische Probleme weitergeführt werden konnte. Die Kommunikation erfolgte dabei grösstenteils über Microsoft Teams und Whatsapp, das Dokumentenmanagement wurde über Microsoft Onedrive organisiert und für die Softwareentwicklung wurde Github verwendet.

Durch die geringen Materialkosten konnten die Komponenten zudem doppelt beschafft werden, sodass die Teammitglieder unabhängiger am Projekt arbeiten konnten. Durch die Pandemie wurde ebenfalls der Termin der Projektpräsentation vom 2. Dezember 2020 auf den 7. Januar 2020 verschoben.



Inhaltsverzeichnis

1 Vorwort	1
2 Einführung	4
2.1 Aufgabenstellung	4
2.2 Github Repository	4
3 Situationsanalyse	5
3.1 Zielkatalog	5
3.2 Use-Case Diagramm	6
3.3 Zeitplan	7
4 Grobkonzept	8
4.1 Systemaufbau	8
4.2 Algorithmus	8
4.3 Namensgebung	8
5 Detailkonzept	9
5.1 Topologieschema	9
5.2 Sensorik	10
5.2.1 Ultraschallsensor HC-SR04	10
5.2.2 Beschleunigungssensor MPU6050	10
5.2.3 Hall-Encoder	11
5.3 Aktorik	11
5.3.1 Antriebsmotor DG01D-E	11
5.3.2 Motortreiber TB6612FNG	11
5.4 Boardcomputer Raspberry Pi 4B	12
5.5 Stromversorgung	12
5.5.1 Akku-Modul Diymore.cc	12
5.5.2 Spannungswandler XL6009	12
5.6 Mechanischer Aufbau	13
5.7 Kostenberechnung	14
6 Labirinto Uno	15
7 Labirinto Due	16
7.1 Mechanischer Aufbau	16
7.1.1 CAD-Modell	16
7.1.2 Fertigung & Montage	17
7.2 Elektrische Schaltung	18
7.2.1 Schaltplan	18
7.2.2 Verteil-Platine	19
8 Software	20
8.1 Einführung in ROS	20
8.2 Schematischer Überblick Labirinto	20
8.2.1 Sensoren package	20
8.2.2 Actions package	21
8.2.3 Main package	22
8.3 Launch File	22
8.4 PID-Regler	23
8.5 Labirinth-Löse-Algorithums	23
9 Systemtests & Debugging	25
9.1 Leistungsarme Motoren	25



9.2 Ausgelassene Encoder-Counts	25
9.3 Zuverlässigkeit Ultraschallsensoren	26
10 Verbesserungen	27
10.1 Systemfunktion	27
10.2 Motoren	27
10.3 Navigation	27
11 Fazit	28
Literatur	29
Tabellenverzeichnis	29
Abbildungsverzeichnis	30
12 Anhang	31
12.1 Zeitplan	31
12.2 Use-Cases	32
12.2.1 Use-Case 1	32
12.2.2 Use-Case 2	32



2 Einführung

Im 5. Semester des Studiengangs Mechatronik Trinational wurde im Rahmen des Fachs „Mechatronisches Labor“ der Labyrinth-Löse-Roboter „Labirinto Due“ mit dem Robot Operating System (ROS) erstellt. Der gesamte Roboter wurde durch das Projektteam konzipiert, gefertigt und programmiert. Zudem musste der Umgang mit ROS und das Programmieren mit Python erlernt werden.

2.1 Aufgabenstellung

Im Rahmen der Projektarbeit im fünften Semester des Studiengangs Mechatronik Trinational soll ein autonomer Labyrinth-Löse-Roboter entwickelt werden. Dieser soll eigenständig ein unbekanntes Labyrinth vom Start bis zum Ziel durchfahren und anschliessend auf dem schnellsten Weg zurückfahren. Der Startbefehl soll dabei über einen eingebauten Taster erfolgen.

Die Seitenlängen des Labyrinths für die Projektpräsentation betragen 3m und die Gangbreiten betragen 30cm. Im Labyrinth gibt es Kurven und Abzweigungen, Kreuzungen und Inseln werden jedoch nicht eingebaut.

Die Materialkosten dürfen den Betrag von CHF 200.- nicht überschreiten. Komponenten welche im Elektroniklabor der FHNW bereits verfügbar sind müssen nicht im Budget berücksichtigt werden.

Die Studierenden erhalten einen Nachmittag pro Woche Zeit um am Projekt in der Fachhochschule Nordwestschweiz zu arbeiten. Das fertige Projekt wird soll am 7. Januar 2021 den Betreuern und der Klasse präsentiert werden.

2.2 Github Repository

Wie bereits im Vorwort erwähnt wurde, erfolgte die gesamte Entwicklung des Codes im Home-Office über das Versionsverwaltungs-System „Github“. Das CAD-Modell, die Dokumentation und das Pflichtenheft sind ebenfalls im Labirinto Repository auf Github ersichtlich und öffentlich zugänglich.

<https://github.com/ADIMADE/Labirinto>



3 Situationsanalyse

3.1 Zielkatalog

Bei Projektbeginn wurde ein Zielkatalog nach der Logik von „System-Engineering“ erstellt. Darin sind sämtliche Anforderungen an das Projekt in Form von vier Zielarten definiert.

Zielklasse	Zieleigenschaften	Ausmass	Zeitpunkt	Zielart	Priorität
Projektziele					
Fertigungsunterlagen	Erstellung CAD-Modell		02.12.2020	M	-
Fertigungsunterlagen	Erstellen Schalplan		02.12.2020	M	-
Dokumentation	Erstellen einer vollständigen Projekt-Dokumentation		02.12.2020	M	-
Termin	Projektabslusstermin einhalten	02.12.2020		R	100
Summe					100
Systemfunktion					
Fortbewegung	Labyrinth vom Start zum Ziel durchfahren	Autonom	02.12.2020	M	-
Fortbewegung	Auf dem schnellsten Weg vom Ziel durchs Labyrinth an den Start fahren	Autonom	02.12.2020	M	-
Fortbewegung	Manuelles Fahren, durch Entwicklungsrechner ferngesteuert	SSH Befehle	02.12.2020	W	20
User Interface	Darstellung von Zuständen mit RGB-LED		02.12.2020	W	4
User Interface	Darstellung des zurückgelegten Wegs auf einem externen Gerät		02.12.2020	W	1
Systemeigenschaften					
Geometrie	Maximale Baubreite	> 300mm	02.12.2020	R	30
Energie	Stromversorgung mit wiederaufladbaren Akkus	5VDC	02.12.2020	M	-
Design	Ansprechendes und modernes Aussehen		02.12.2020	W	5
Finanzen					
Budget	CHF	< 200 CHF	02.12.2020	R	40
Summe					100
Zielarten					
M	Muss-Ziel				
W	Wunsch-Ziel				
R	Restriktionsziel				
O	Optimierungsziel				

Abbildung 3.1: Zielkatalog für die Entwicklung von „Labirinto“
[Eigene Darstellung]

Die erfolgreiche Fertigstellung des Projekts bedingt das Erreichen sämtlicher Muss- und Restriktions-Ziele. Die Wunsch-Ziele werden nach Möglichkeit umgesetzt, diese sind jedoch nicht entscheidend für die Grundfunktionalität des Roboters.

3.2 Use-Case Diagramm

Während der Situationsanalyse wurden verschiedene Anwendungsfälle (Use-Cases) für den Labirinto entwickelt und in einem Use-Case Diagramm verdeutlicht. Die Szenarien wurde auf Grundlage des Zielkatalogs definiert und veranschaulichen das System „Labirinto“ aus Anwendersicht.

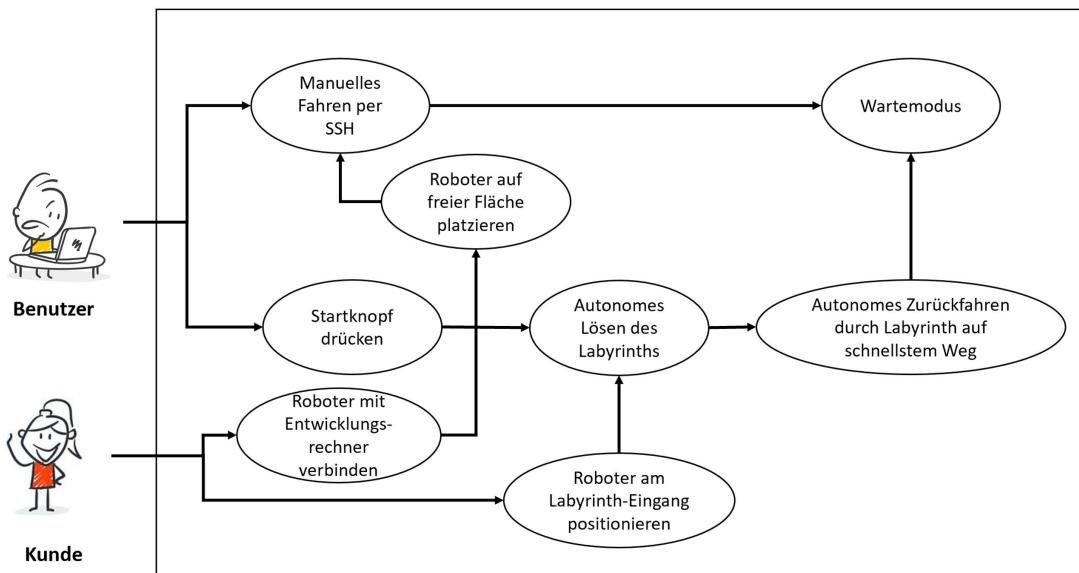


Abbildung 3.2: Use-Case Diagramm
[Eigene Darstellung]

Eine solche Darstellung eines Systems ist speziell in der Software-Entwicklung ein sehr hilfreiches Mittel um sämtliche Eventualitäten aufzuzeigen und eine anwenderfreundliche Funktionalität zu gewährleisten. Die detaillierte Beschreibung der einzelnen Use-Cases sind im Pflichtenheft im Anhang Register 12.2 ersichtlich.



3.3 Zeitplan

Nach Begutachtung der Aufgabenstellung wurde ein detaillierter Soll-Ist-Zeitplan erstellt. Dabei wurde die Arbeit auf die 12 zur Verfügung stehenden Wochen bis zur ursprünglichen Projektpräsentation verteilt. Des Weiteren wurden 5 Meilensteine definiert, die den Projektstart oder den Abschluss gewisser Arbeitsschritte markieren. Der detaillierte Projektzeitplan ist im Anhang Register 12.1 ersichtlich.

Aktivität	Deadline
1. Meilenstein: Beginn Projektarbeit	16.9.2020
Informieren + Recherche	23.09.2020
Erstellung Pflichtenheft	29.09.2020
2. Meilenstein: Präsentation Pflichtenheft	30.09.2020
Bestellung Material	01.10.2020
Erstellung CAD-Modell	06.10.2020
Erstellung Elektroschema	06.10.2020
Mechanische Fertigung Fahrgestell	14.10.2020
Montage und Verdrahtung	20.10.2020
3. Meilenstein: Präsentation Zwischenstand "Hardware"	21.10.2020
Programmierung Funktionen für Sensordaten und Fahrmodus	28.10.2020
Programmierung Labyrinth-Löse-Algorithmus	11.11.2020
Programmierung Spurhalte-Algorithmus	18.11.2020
Fertigung & Montage Karosserie und Beleuchtung	22.11.2020
Programmierung Beleuchtungssteuerung	24.11.2020
4. Meilenstein: Präsentation Zwischenstand "Software"	25.11.2020
Problemlösung, Implementierung von Verbesserungen	01.12.2020
5. Meilenstein: Abschlusspräsentation & Abgabe	07.01.2021 (aktualisiert)

Abbildung 3.3: Übersicht Meilensteine
[Eigene Darstellung]

4 Grobkonzept

4.1 Systemaufbau

Um eine systematische Entwicklung des „Labirintos“ zu gewährleisten wurde mit der Erstellung eines Blockschaltbild begonnen. Dies ist eine schematische Darstellung des mechatronischen Systems und zeigt die Beziehungen zwischen sämtlichen Sensoren und Aktoren, den Steuerungselementen und der Systemumgebung.

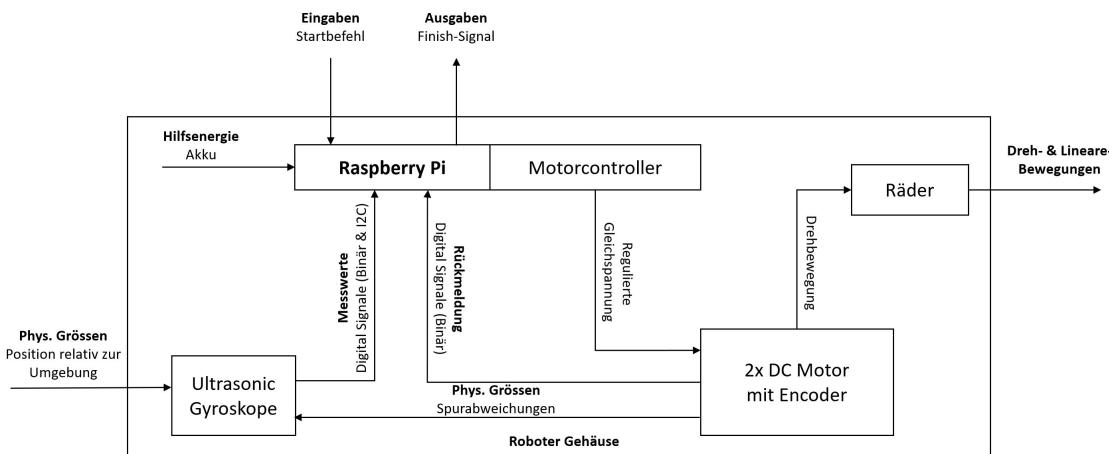


Abbildung 4.1: Mechatronisches Blockschaltbild des „Labirintos“
[Eigene Darstellung]

4.2 Algorithmus

Während der Fahrt des Roboters durch das Labyrinth muss eine Vielzahl von Messwerten erfasst und verarbeitet werden. Damit der Roboter in einem angemessenen Tempo fahren kann müssen sämtliche Messwerte sehr schnell aufbereitet werden, um das Programm nicht zu verlangsamen. Dies bedeutet, dass die Messwerte parallel verarbeitet werden müssen, was die Verwendung von „Nebenläufigkeiten“ (Threads) zur Folge gehabt hätte.

Das Projektteam hat sich daher für die Verwendung des Robot Operating System (ROS) entschieden, da dieses Framework die Verarbeitung von gleichzeitig ablaufenden Programmen erleichtert.

4.3 Namensgebung

Die Bezeichnung „Labirintos“ ist ein Begriff aus dem Vokabular der weit verbreiteten Plansprache Esperanto und bedeutet Labyrinth. Esperanto wurde 1887 vom polnischen Augenarzt Ludwik Lejzer Zamenhof veröffentlicht. Das Ziel dieser Sprache ist eine möglichst einfache und einheitliche Kommunikation zwischen Menschen mit unterschiedlichen Nationalitäten und Sprachen [1].

Das Projektteam hat sich für einen Namen in Esperanto entschieden, da der zu entwickelnde Roboter nach den selben Grundsätzen entwickelt werden soll.

Die Bezeichnungen „Uno“ und „Due“ kennzeichnen die Version des Roboters.

5 Detailkonzept

Im Rahmen des Detailkonzepts wurde das Blockdiagramm aus dem Grobkonzept detailliert und sämtliche Komponenten ausgelegt. Zudem wurde eine Konzeptskizze der Roboter-Hardware, sowie eine Kostenberechnung erstellt.

5.1 Topologieschema

Bei der Entwicklung des elektrischen Schaltkreises wurde ein Topologieschema erstellt. Dieses Schema gibt eine allgemeine Übersicht über sämtliche benötigte Komponenten und die benötigten Spannungsversorgungen und Datenleitungen.

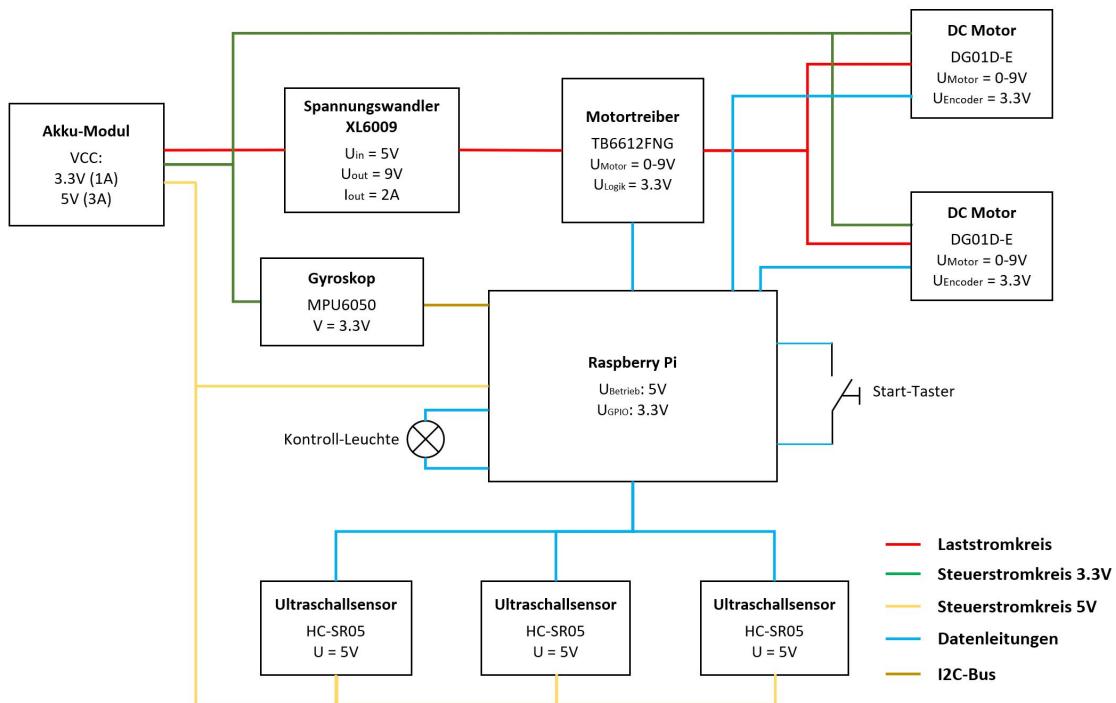


Abbildung 5.1: Topologieschema des „Labirintos“
[Eigene Darstellung]

Anhand des Topologieschemas konnten sämtliche Komponenten für die Sensorik, Aktorik, Steuerung und die Spannungsversorgung ausgelegt werden. Der Raspberry Pi bildet das Herzstück des „Labirintos“, dort wo sämtliche Datenleitungen zusammenkommen. Dieser ist für die Messwerterfassung, Verarbeitung und Ausgabe verantwortlich. Das Labyrinth-Löse-Programm kann mit Hilfe des Start-Tasters gestartet werden. Durch die integrierte Kontroll-Leuchte erhält der Benutzer ein Feedback über den aktuellen Zustand des „Labirintos“.

Im folgenden wird die Auswahl und Dimensionierung sämtlicher Komponenten erläutert.



5.2 Sensorik

5.2.1 Ultraschallsensor HC-SR04

Der HC-SR04 ist ein häufig verwendet Ultraschallsensor, mit welchem Distanzen zwischen 2-400cm erfasst werden können.

Folgende Messwerte werden mit dem HC-SR04 ermittelt:

- Beidseitige Position zu den Wänden [m]
- Vorkommen einer oder mehrerer Abzweigungen [True/False]



Abbildung 5.2:
Ultraschallsensor
HC-SR04
[2]

Die Distanzmessung erfolgt durch den Zeitunterschied zwischen dem Senden und Empfangen eines Ultraschallsignals, nachdem dieses an einem Objekt reflektiert und zurückgeworfen wurde. Um die effektive Distanz zwischen Sensor und Objekt zu erhalten, muss die gemessene Zeitdifferenz mit der Schallgeschwindigkeit multipliziert und anschliessend mit 2 dividiert werden.

Da der Sensor im Elektroniklabor an der FHNW bereits verfügbar war fielen keine Beschaffungskosten dafür an.

5.2.2 Beschleunigungssensor MPU6050

Der MPU6050 ist ein 3-Achsen Beschleunigungssensor, welcher zusätzlich über drei Gyroskopie verfügt. Mit dem Sensor kann die Gravitations-Beschleunigung und die Winkelgeschwindigkeit gemessen werden.

Folgende Messwerte werden mit dem MPU6050 ermittelt:



Abbildung 5.3:
Beschleunigungssensor
MPU6050
[3]

Um den Drehwinkel zu berechnen, muss die Winkelgeschwindigkeit [$^{\circ}/s$] mit der verstrichenen Zeit [s] nach Messbeginn integriert werden. Der Sensor kommuniziert über das I2C-Protokoll.

Da der Sensor im Elektroniklabor an der FHNW bereits verfügbar war fielen keine Beschaffungskosten dafür an.

5.2.3 Hall-Encoder

Mit einem Encoder kann die Umdrehungsgeschwindigkeit der Räder gemessen werden. Dieser Sensor ist direkt im Motorengehäuse montiert und misst die Umdrehung der Motorenachse.

Folgende Messwerte werden mit dem Hall-Encoder ermittelt:

- Zurückgelegte Distanz [m]
- Geschwindigkeit beider Räder [m/s]

Der verwendete Encoder basiert auf dem elektromagnetischen Hall-Effekt und sendet an den Boardcomputer ein Rechtecksignal. Für die Auswertung der Messwerte müssen anschliessend sämtliche positiven Impulse addiert werden. Während einer Umdrehung des Antriebrades sendet der verwendete Hall-Sensor 275 Impulse.

5.3 Aktorik

5.3.1 Antriebsmotor DG01D-E

Als Antriebsmotor wird der DG01D-E verwendet. Dieser DC-Motor verfügt über ein integriertes metallgetriebe Getriebe mit einer Übersetzung von 1:48, sowie einem integrierten Encoder (Siehe Kapitel 5.1.3).

Der Motor kann über eine Spannung von 3 bis 9V angesteuert werden, bei 4.5V erreicht er eine Drehzahl von 90 min^{-1} .

Der Motor wurde auf Grund seiner kompakten Bauweise und des bereits integrierten Encoder und Getriebe ausgewählt.



Abbildung 5.4:
Antriebsmotor DG01D-E
[4]

5.3.2 Motortreiber TB6612FNG

Als Schnittstelle zwischen dem Boardcomputer und dem Motor muss ein Motortreiber mit integrierter H-Brücke verwendet werden. Dieser dient zur Verstärkung der Steuersignale und bildet die Schnittstelle zwischen Steuerlogik und Motor. Der Treiber wandelt die PWM-Steuersignale in den Motorstrom.

Mit dem TB6612FNG können zwei Gleichstrommotoren mit einem Strom von 1.2A (3.2A Peak) gesteuert werden. Die Geschwindigkeit und die Drehrichtung der beiden Motoren können dabei individuell angesteuert werden.

Die Verwendung des TB6612FNG wurde vom Lieferanten in Kombination mit den DG01D-E Motoren empfohlen.



Abbildung 5.5: Motortreiber
TB6612FNG
[5]

5.4 Boardcomputer Raspberry Pi 4B

Der Boardcomputer musste gemäss den Systemanforderungen des Robot Operating System ausgelegt werden, dieses wir vorwiegend auf einem Ubuntu-Server installiert. Durch den Verkaufsstart des Raspberry Pi 4B im Juni 2020 kann ROS nun erstmals barrierefrei in Kombination mit dem Raspberry Pi eingesetzt werden. Der Raspberry Pi 4B ist erstmals in einer 4GB und 8GB RAM Version erhältlich und erfüllt somit die minimalen Systemanforderungen von 4GB RAM des Robot Operating Systems.

Für die Entwicklung des „Labirintos“ wurde die Version mit 8GB RAM gewählt, da der Umfang des späteren Programms in dieser Phase unklar ist. Der Raspberry Pi bietet neben der kompakten Baugröße ebenfalls den Vorteil, dass er bereits über integrierte GPIO-Pins verfügt. Die GPIO-Pins werden als Schnittstelle zwischen dem Controller, den Sensoren und dem Motor-Treiber genutzt, dadurch muss kein zusätzlicher Mikrocontroller verwendet werden.

Der Raspberry Pi wird durch das Elektroniklabor der FHNW zur Verfügung gestellt.



Abbildung 5.6: Raspberry Pi 4B 8GB [6]

5.5 Stromversorgung

5.5.1 Akku-Modul Diymore.cc

Damit der „Labirinto“ autonom ein Labyrinth durchqueren kann entschied sich das Projektteam für den Einbau einer internen Spannungsversorgung in Form eines Akkus. Der Akku muss dabei das gesamte System mit 3.3V, 5V und 9V versorgen. Dazu wird ein Akku-Modul der Firma Diymore.cc verwendet, welches mit vier 18650 Akkuzellen betrieben wird. Die Ladung der einzelnen Zellen beträgt 3200mAh.

Das Akku-Modul verfügt über zwei Spannungswandler, welche die Spannung der Akkus von 3.7V auf 3.3V (1A) und 5V (3A) konvertieren. Das Modul schützt die Akku-Zellen gegen Über- und Tiefentladung. Im Akku-Modul ist zudem eine integrierte Ladelektronik eingebaut, mit der die Lithium Akkus über ein USB-C Anschluss durch ein 5V Netzteil geladen werden können.



Abbildung 5.7:
Akku-Module Diymore.cc
[7]



Abbildung 5.8:
Spannungswandler XL6009
[7]

5.6 Mechanischer Aufbau

Der „Labirintos“ soll ein visionäres und zeitgemäßes Design aufweisen. Dafür wurde ein schlichtes, weißes Gehäuse mit kontrastreichen Ecken und Seitenrändern gewählt. Auf der Oberseite des Gehäuse wird das Logo gut sichtbar aufgetragen. Die Ultraschallsensoren können durch sechs Aussparungen im Gehäuse ihre Signale senden und empfangen.

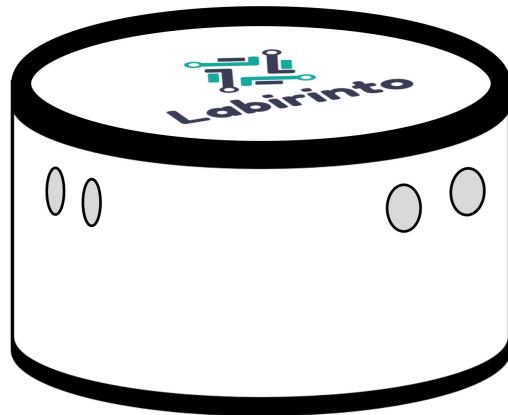


Abbildung 5.9: Dreidimensionale Skizze des „Labirintos“
[Eigene Darstellung]

Unter dem Gehäuse befindet sich das Fahrgestell des „Labirintos“ wo sämtliche Komponenten verbaut sind. Das Fahrgestell besteht aus zwei Plattformen die durch Distanzsäulen miteinander verbunden sind. Auf der unteren Plattform A sind sämtliche Antriebs- und Distributionskomponenten verbaut. Auf der oberen Plattform B befinden sich die Ultraschallsensoren und der Raspberry Pi.

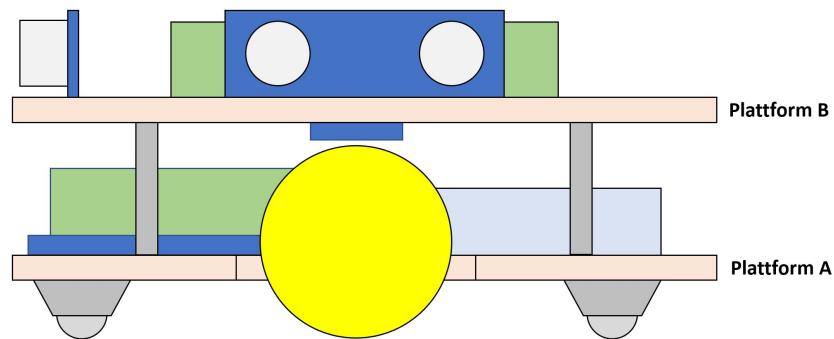


Abbildung 5.10: Fahrgestell in der Ansicht von Rechts
[Eigene Darstellung]

Als Radkonfiguration wurde eine „Two-wheel differential drive with tow additional points of contact“ verwendet. Dies bedeutet, dass sich die beiden Antriebsräder auf der Mitte des Roboters auf einer Achse befinden. Vorne und hinten wird der Roboter durch zwei Kugellräder auf der Mittelachse abgestützt. Diese Radkonfiguration wird vielfach bei kleineren Robotern verwendet. Der Vorteil des „differential drive“ besteht darin, dass sich der Roboter um die eigene Achse drehen kann und dadurch ein minimaler Platz für das Manövrieren benötigt.[8]

Die drei Ultraschallsensoren sind nach vorne, rechts und links ausgerichtet, um durchgehend den Labyrinth-Gang auf Abzweigungen abzusuchen. Der Gyroskop-Sensor ist auf der z-Achse des Roboters montiert.

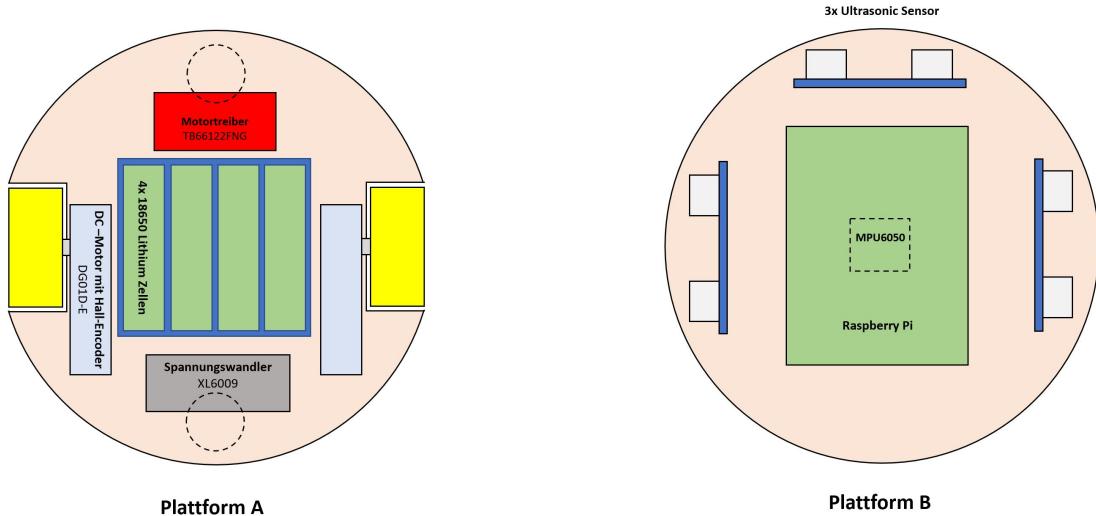


Abbildung 5.11: Plattform A und B in der Ansicht von Oben
[Eigene Darstellung]

5.7 Kostenberechnung

In der Kostenberechnung wurden sämtliche Komponenten erfasst, welche zusätzlich für den Roboter beschafft werden mussten. Sämtliche 3D gedruckten Komponenten und Bauteile die vom Elektroniklabor zur Verfügung gestellt wurden, sind in der Kostenberechnung nicht beurteilt worden.

Artikel	Artikelnummer	Hersteller	Lieferant	Einheit	CHF/Einheit	Anzahl Einheiten	Kosten Soll	Kosten Ist
Getriebemotor DG01D-E 1:48 mit Encoder	421286	Sparkfun	Bastelgarage	Stk	CHF 9.90	2	CHF 19.80	CHF 19.80
Motor Treiber TB6612FNG	420520		Bastelgarage	Stk	CHF 4.90	1	CHF 4.90	CHF 4.90
4x18650 Lithium Batterie Shield	421084	Diymore.cc	Bastelgarage	Stk	CHF 16.90	1	CHF 16.90	CHF 16.90
Lithium Akku 18650	420731		Bastelgarage	Stk	CHF 9.90	4	CHF 39.60	CHF 39.60
XL6009 Spannungswandler	420162		Bastelgarage	Stk	CHF 5.90	1	CHF 5.90	CHF 5.90
Material Total							CHF 87.10	CHF 87.10
Student; Soll				h	0		CHF 0.00	
Student; Ist				h	0			CHF 0.00
Herstellkosten Total							CHF 0.00	CHF 0.00
Kosten Total							CHF 87.10	CHF 87.10

Abbildung 5.12: Kostenberechnung
[Eigene Darstellung]

Unter der Berücksichtigung der Kosten für sämtliches Kleinmaterial (Kabel, Schrauben, Filament, etc.), des Raspberry Pi und sämtlicher Sensoren betragen die effektiven Kosten eines „Labirintos“ schätzungsweise bei 190.- CHF. Ein Raspberry Pi 4B 8GB RAM ist ab 70.- CHF erhältlich.

6 Labirinto Uno

Im Anschluss an das Detailkonzept wurde der Prototyp „Labirintos Uno“ gebaut. Das Ziel war die schnellst mögliche Verfügbarkeit eines funktionellen Prototyps für Systemtest. Der mechanische und elektrische Aufbau des „Labirinto Uno“ wurden in einem iterativen Entwicklungsverfahren erstellt. Das Fahrgestell wurde auf Grund der Skizzen auf dem Detailkonzept aus Sperrholz und Print-Distanzbolzen gefertigt. Die Montage der Komponenten und deren Verdrahtung erfolgte in mehreren Etappen.

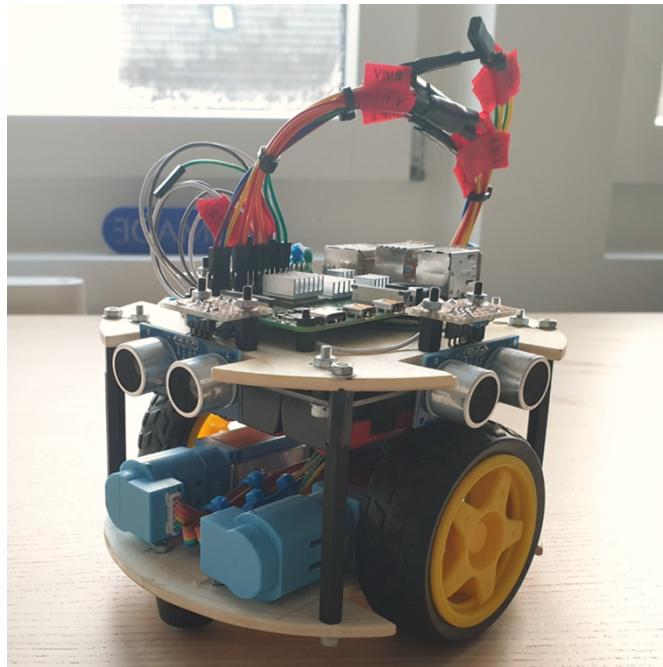


Abbildung 6.1: Plattform A und B in der Ansicht von oben
[Eigene Darstellung]

Als erstes wurde die Stromversorgung und Ansteuerung der Motoren realisiert und erste Test-Programme geschrieben. Anschliessend wurde dieser Vorgang wiederholt und um die Bereiche Ultraschallsensoren, Encoder und Gyroskop schrittweise erweitert. Der „Labirintos Uno“ diente ebenfalls zur Erprobung von unterschiedlichen Algorithmen und im speziellen zur Einarbeitung in das Robot Operating Systems.



Abbildung 6.2: Test-Strecke für PID-Regeltests
[Eigene Darstellung]

7 Labirinto Due

Der „Labirinto Due“ ist die direkte Weiterentwicklung des „Labirinto Uno“. Im neuen Roboter wurden sämtliche Erkenntnisse aus dem Prototyp implementiert und viele Verbesserungen im Bereich Mechanik und Elektronik implementiert. Dadurch ist nun eine einwandfreie Funktionalität der Hardware und eine bessere Wartbarkeit gewährleistet.

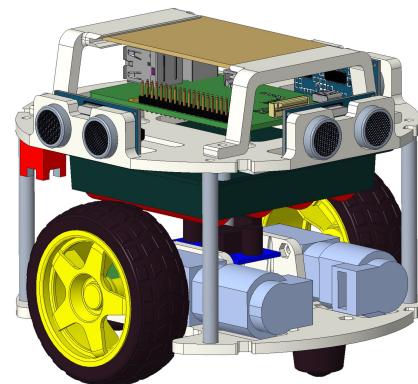
7.1 Mechanischer Aufbau

7.1.1 CAD-Modell

Bei der Entwicklung des „Labirinto Due“ wurde zu Beginn ein CAD-Modell des Roboters erstellt. Dafür wurde die relativ einfache Grundform des „Labirinto Uno“ übernommen und anschliessend für die Fertigung im 3D-Drucker optimiert. Es wurden verschiedene Halterungen für die Elektronik-Komponenten konstruiert, um eine einfachere und schnellere Endmontage zu gewährleisten.



(a) Roboter mit Gehäuse



(b) Roboter ohne Gehäuse

Abbildung 7.1: CAD-Modell des „Labirintos Due“
[Eigene Darstellung]

Das 3D-Modell wurde im CAD-Programm Creo Parametric 6.0 erstellt. Es besteht aus den drei Baugruppen „Plattform A“, „Plattform B“ und „Gehäuse“. Damit das Gehäuse einfach entfernt werden kann wurden sämtliche Elektro-Komponenten auf die übrigen zwei Baugruppen verteilt.

Im Gehäuse Deckel wurde das Negativ des Labirinto-Logo auf der Rückseite ausgespart. Die obere Printplatte über dem Raspberry Pi ist mit vier LEDs bestückt. Im Betrieb beleuchten die LEDs die Rückseite des Deckels, dadurch schimmert das Labirinto-Logo durch den weißen Deckel durch.

Auf der oberen Printplatte ist ebenfalls ein Taster angebracht, mit welchem der Labyrinth-Löse Algorithmus gestartet werden kann. Auf der Oberseite des Gehäuse ist in der Mitte ein Stössel angebracht, durch den der Taster auf der Print-Platte betätigt werden kann.

7.1.2 Fertigung & Montage

Die Grundplatten für die Plattform A und B mittels eines 3D-Drucker im FDM-Verfahren aus Polylactide (PLA) hergestellt. Beide Grundplatten wurden für die Montage der Komponenten mit Gewinde-Einsätzen versehen.

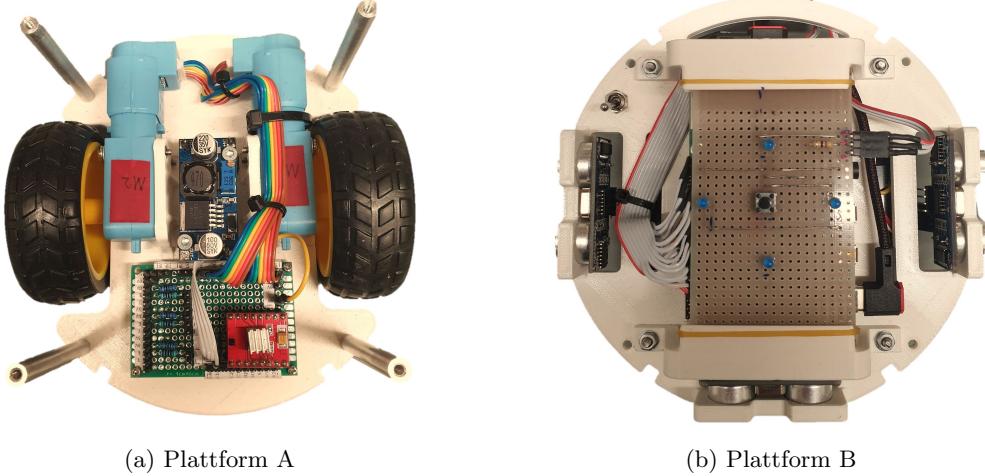


Abbildung 7.2: Montage der Plattformen
[Eigene Darstellung]

Für die Verbindung der beiden Plattformen wurden Distanzsäulen aus Aluminium mit beidseitigem M3 Gewinde gedreht. Zum Schluss wurden die Grundplatten und sämtliche Komponenten wurden mit Zylinderkopf-Schrauben verbunden und das 3D gedruckte Gehäuse auf den Roboter gesteckt.



Abbildung 7.3: Hinterleuchtetes Labirinto-Logo
[Eigene Darstellung]

7.2 Elektrische Schaltung

7.2.1 Schaltplan

Die iterativ erstellte Verdrahtung des Prototyps wurde für die Entwicklung von „Labirinto Due“ dokumentiert und optimiert. Überflüssige Ground-Verbindungen wurden weggelassen und das Schema wurde um die Bereiche Taster und Beleuchtung ergänzt.

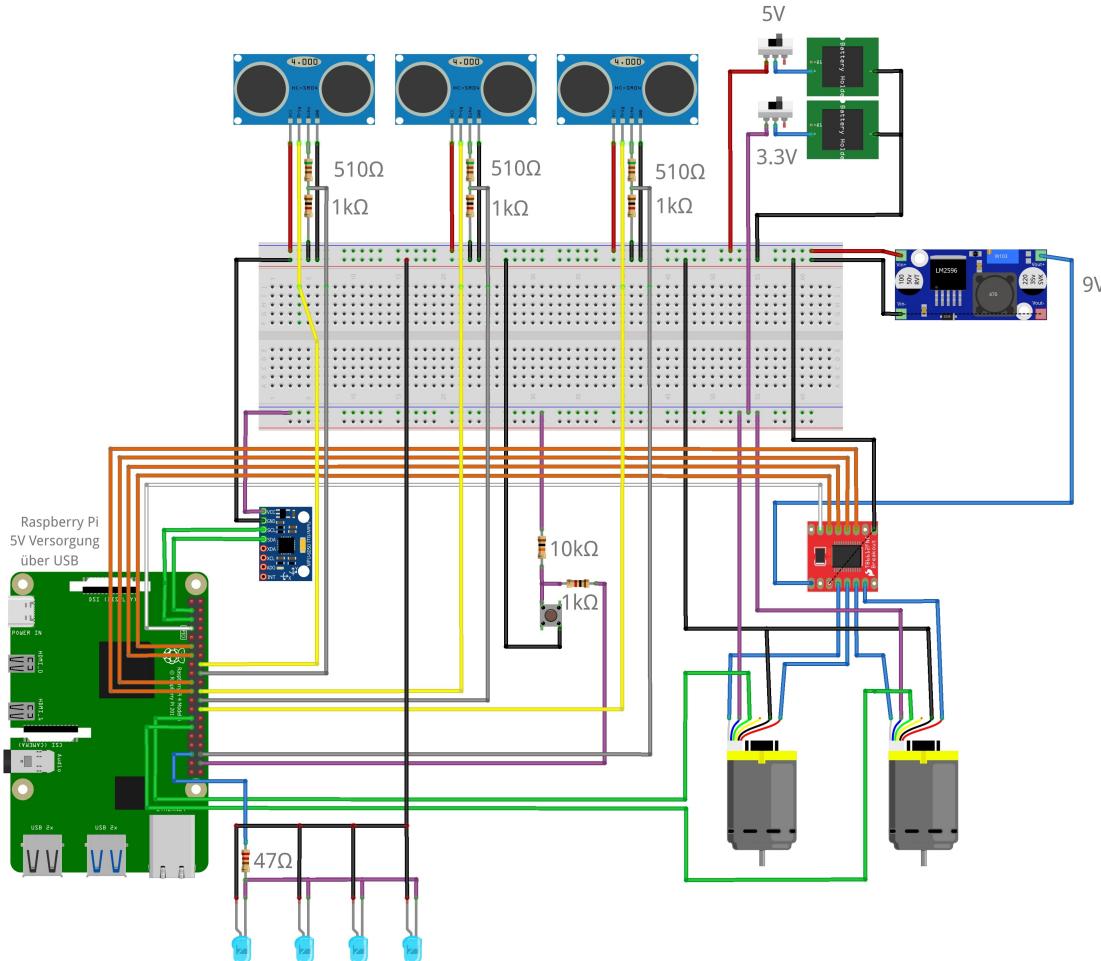


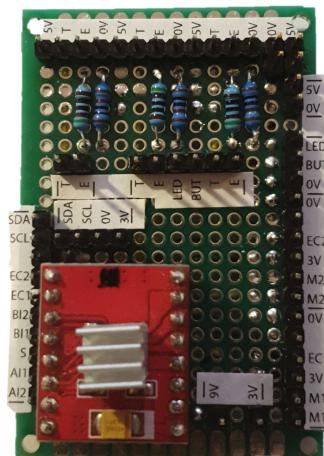
Abbildung 7.4: Schaltplan des „Labirinto Due“
[Eigene Darstellung]

Da die Echo-Ausgänge der Ultraschall-Sensoren ein 5V Signal ausgeben mussten drei Spannungssteiler integriert werden, damit nur 3.3V Signale bei den GPIO-Eingängen des Raspberry Pi ankommen. Bei Eingangsspannungen von über 3.3V können die GPIOs des Raspberry Pi schaden nehmen. Da die Ultraschall-Sensoren auf TTL-Logik basieren benötigen diese eine Versorgungsspannung von 5V.

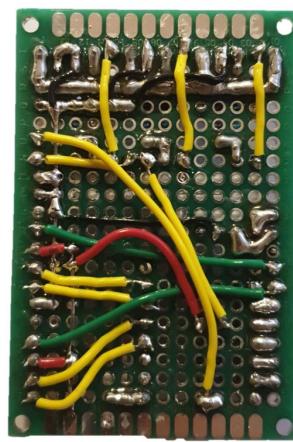
Um eine korrekte Funktion des Tasters zu gewährleisten wurde eine Pull-Up Schaltung angewendet. Durch diese Schaltung wird der GPIO-Eingang des Raspberry Pi im Ausgangszustand definiert auf 3.3V (High) gezogen. Wird nur der Taster gedrückt liegt am GPIO ein definiertes 0V (Low) Signal an. Ohne diese Schaltung ist der Zustand am GPIO nicht eindeutig definiert.[9]

7.2.2 Verteil-Platine

Beim Bau des „Labirinto Due“ wurde ebenfalls die Art der Verdrahtung optimiert. Die Leitungen sämtlicher Aktoren, Sensoren und Steuerungselemente wurden dabei auf einer Verteilplatine zusammengeführt. Die Spannungsteiler und Pull-Up Schaltungen sowie der Motor-Treiber wurden zudem direkt in die Platine integriert. Für die Verbindung der Bauteile mit der Printplatte wurden individuell konfektionierte Flachbandkabel genommen. Die Flachbandkabel werden mit Dupont-Steckern mit der Verteilplatine und den jeweiligen Komponenten verbunden. Die gesamte Stromversorgung der Komponenten erfolgt ebenfalls ausschliesslich über die Printplatte.



(a) Vorderseite



(b) Rückseite

Abbildung 7.5: Verteilplatine
[Eigene Darstellung]

Durch die Verteilplatine konnte eine übersichtliche und fehlerfreie Verdrahtung realisiert werden. Mit der übersichtlichen Beschriftung sämtlicher Pins wurde eine einfache und schnelle Montage gewährleistet.



8 Software

8.1 Einführung in ROS

Das Robot Operating System (abgekürzt ROS) ist eine Middleware-Software für Roboter, welche üblicherweise auf einem Linux-Rechner installiert wird. Die Grundidee von ROS ist eine einfachere Einbindung sämtlicher Komponenten, sodass mehr Zeit für die eigentliche Applikationsentwicklung des Roboters zur Verfügung steht. Die ROS-Infrastruktur erlaubt es, sämtliche Bauteile eines Systems softwaretechnisch zu verbinden und eine standardisierte Kommunikation zu ermöglichen. Dadurch können Roboter-Funktionen, ohne die Entwicklung von Gerätetreiber, schnell umgesetzt werden. Programme in ROS können mit Python, C++ und Java programmiert werden. Der Labirinto wurde hauptsächlich in Python programmiert.

Das Robot Operating System muss ein Vielfalt von Programmen, Treibern und Algorithmen parallel ausführen können, damit ein Roboter als dynamisches Gesamtsystem funktionieren kann. Diese Basis-Struktur ist durch ein Node-Netzwerk realisiert, die alle miteinander verbunden sind. Die Nodes müssen sich beim ROS-Master registrieren, um ins Gesamt-System eingebunden werden zu können.

Die „Nodes“ werden durch Kanäle „Topics“ verbunden. Jeder Node kann auf diesen Topics Daten senden oder empfangen. Das Robot Operating System basiert auf dem „Publish-Subscribe-Prinzip“. ROS führt sämtliche „Nodes“ und „Topics“ parallel, mit Hilfe verschiedener „Threads“ aus. Das Ziel ist zu jeder Zeit eine maximale Verfügbarkeit sämtlicher Daten im System. Über die Topics werden Messages mit einer standardisierten Struktur ausgetauscht. In ROS gibt es eine Vielzahl von verschiedenen „Nodes“. Im Labirinto wurden speziell die „Action-Nodes“ verwendet. Diese „Nodes“ erhalten durch das „Server-Client-Prinzip“ von den „Client-Nodes“ eine Aktions-Anfrage mit einem „Goal“ (Ziel).

Da ROS ein Opensource-Projekt ist, gibt es eine Vielzahl von verfügbarem Code und Packages. Für häufig verwendete Bauteile sind sogar komplette Nodes verfügbar, die barrierefrei in ROS integriert werden können und die Entwicklungszeit beschleunigen.[10]

8.2 Schematischer Überblick Labirinto

Der gesamte Code des „Labirintos“ wurde auf Github organisiert. Der Code wurde in ein „Sensoren-Package“, ein „Action-Package“ und ein „Main-Package“ gegliedert. In den jeweiligen „Packages“ sind die „Nodes“ implementiert (Siehe Abbildung 8.1).

8.2.1 Sensoren package

Für die drei Sensortypen wurde individuelle „Nodes“ erstellt. Für den MPU wurde einen Node programmiert um den Drehwinkel des Roboters um die z-Achse auszulesen. Dieser Node stellt eine I2C -Verbindung mit dem Sensor her, ermittelt die Winkeländerung der Z-Achse mit einer definierten Rate von 10Hz und publiziert diese Information auf dem Topic /mpu/gyroZ..

Ein zweiter Node lässt die Encoder-Takte aus. Durch die Implementierung einer „List“ konnte ein Node für beide Encoders programmiert werden. Beide Encoder verfügen über eine individuelle Counter-Variabel. Wenn der Raspberry Pi ein Encoder-Impuls empfängt wird ein Count zu einer der beiden Counter-Variablen addiert. Der Wert der beiden Variablen wird auf die beiden „Topics“ publiziert. Die beiden Variablen können durch den „Main-Node“ zurückgesetzt werden.

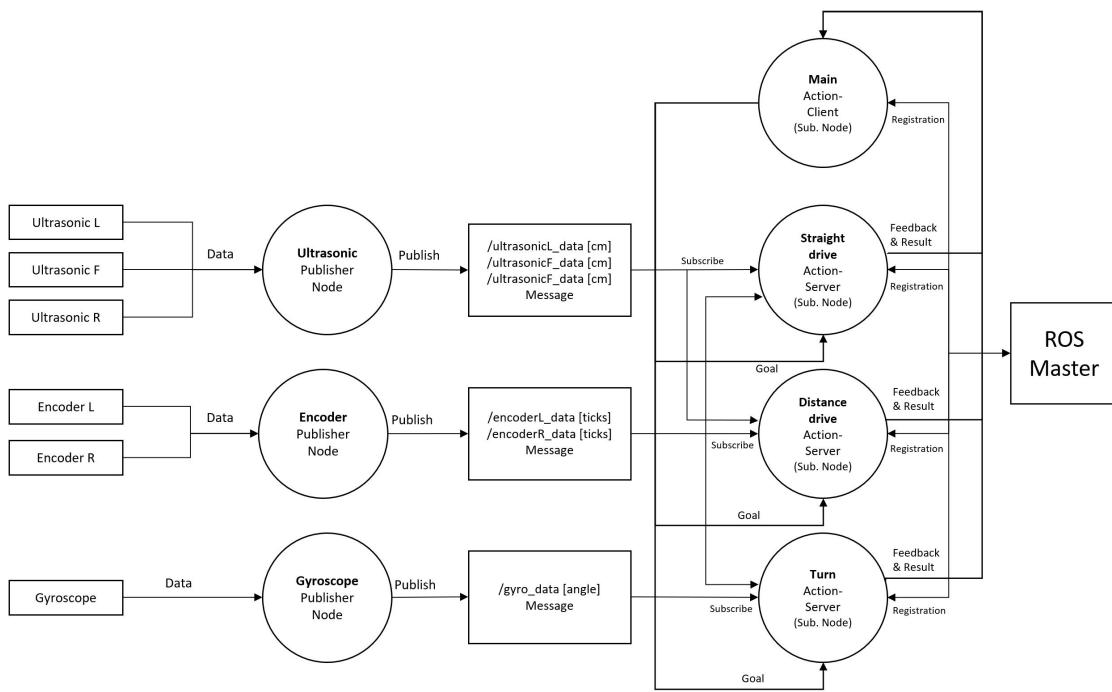


Abbildung 8.1: ROS-Struktur im „Labirinto“
[Eigene Darstellung]

Die drei Ultraschallsensoren werden auch in einem Node, identisch zu den Encodern, verwaltet. Die Distanzen der drei Sensoren werden nach einander ermittelt und auf die Topics „/ultrasonic/Left“, „/ultrasonic/Right“, „/ultrasonic/Front“ publiziert.

8.2.2 Actions package

Der „Turn-Node“ erhält ein „Goal“ vom „Main-Node“ mit einen bestimmten Winkel. Die Aktion kümmert sich um die Drehung des Roboters, während dem sich der „Main-Node“ wieder um den Labyrinth-Löse-Algorithmus kümmert. Der „Turn-Node“ ist ein Subscriber des „Gyroskop-Node“. Dadurch erhält der „Turn-Node“ ein kontinuierliches Feedbacks zur bereits zurückgelegten Drehung.

Die „straightDriveDist-Action“ bekommt den Befehl um den „Labirinto“ eine bestimmte Distanz vorwärts fahren zu lassen. Der „straightDriveDist-Node“ ist ein Subscriber des „Ultraschall-Nodes“ und des „Encoder-Nodes“. Mit einem PID Regler werden die Motor-Geschwindigkeiten geregelt, damit diese sich gleich schnell drehen und der Roboter auf einer geraden Linie.

Die „straightDrive-Action“ erhält keinen quantifizierbaren Zielwert, sondern nur einen Start-Befehl. Um den Roboter in der Gang-Mitte zu halten wurde ein PID-Regler implementiert, welcher die Geschwindigkeit auf Grundlage der beidseitigen Distanz zur Wand regelt. Wenn durch die Ultraschall-Sensoren eine Abzweigung detektiert wurde, hat der „straightDrive-Action“ sein Ziel erreicht und wird durch den „Main-Node“ gestoppt. Der „Main-Node“ entscheidet dann über die Weiterfahrt.



8.2.3 Main package

Im „Main-Package“ ist der Labyrinth-Löse-Algorithmus, die LED-Steuerung und das „Launch-File“ organisiert.

Das Main Programm bildet das Herzstück der gesamten Labirinto-Software. Das Programm ist für sämtliche Richtungsentscheidungen und den Aufruf der „Action-Nodes“ verantwortlich. Der „Main-Node“ ist ein Subscriber des „Ultrasonic-Node“ um Abzweigungen zu erkennen. Abhängig von der Anzahl an verfügbaren Richtungsmöglichkeiten entscheidend der „Main-Node“ welcher Abzweigung gewählt wird.

Der „LED-Node“ ist Abonnent vom „LED-Topic“. Diese bekommt Befehle wie „Blinken-Schnell“ oder „Blinken-Langsam“ und behält diesen Zustand bis ein neuer Befehl kommt.

8.3 Launch File

Anstatt sämtliche „Nodes“ einzeln zu starten, wurde ein „Launch File“ als XML-Datei geschrieben, mit dem sämtliche „Nodes“ auf einmal gestartet werden können. Dieses File erlaubt ebenfalls die Verwaltung sämtlicher GPIO-Pins, damit diese automatisch in den ROS-Parameter-Server aufgenommen werden. Die GPIO-Parameter können von sämtlichen „Nodes“ verwendet werden. Das „Launch File“ startet ebenfalls den ROS-Master („ROS-Core“) und wird direkt nach dem Einschalten des Raspberry Pi gestartet.

Im folgenden ist das komplette „Launch File“ ersichtlich:

```
<launch>

    <!--ENCODER PINS-->
    <param name="HSA1" value="11" />
    <param name="HSB1" value="13" />

    <!--MPU AXIS REGISTER ADRESSES-->
    <param name="gyroZ" value="0x47" />
    <param name="gyroY" value="0x45" />
    <param name="gyroX" value="0x43" />
    <param name="accZ" value="0x3F" />
    <param name="accY" value="0x3D" />
    <param name="accX" value="0x3B" />

    <!--ULTRASONIC PINS-->
    <param name="ultrasonicFront_echo" type="int" value="18" />
    <param name="ultrasonicFront_trig" type="int" value="16" />
    <param name="ultrasonicRight_echo" type="int" value="40" />
    <param name="ultrasonicRight_trig" type="int" value="38" />
    <param name="ultrasonicLeft_echo" type="int" value="26" />
    <param name="ultrasonicLeft_trig" type="int" value="24" />
```

```

<!--STARTING NODES-->
<node name="encoderNode" pkg="sensors" type="encoder.py" />
<node name="mpuNode" pkg="sensors" type="mpu.py" />
<node name="ultrasonicNode" pkg="sensors" type="ultrasonic.py" />

<node name="TurnActionNode" pkg="actions" type="turn.py" />
<node name="StraightDriveActionNode" pkg="actions" type="straightDrive.py" />
<node name="StraightDriveDistActionNode" pkg="actions" type="straightDriveDist.py" />

<node name="mainNode" pkg="main" type="main.py" />

</launch>

```

8.4 PID-Regler

Damit der Roboter möglichst geradeaus fährt, wurden PID-Regler in die „Action-Nodes“ implementiert. Diese PID-Regelung sollte dazu beitragen, dass der „Labirinto“ gerade fährt. Im Rahmen der Abschlussarbeit des Fachs „Statistik“ wurde mittels statistischer Optimierungsverfahren versucht die optimalen Regel-Faktoren zu finden. Die genannte Abschlussarbeit ist im Github-Repository einsehbar.

Folgendes Code-Beispiel zeigt den verwendeten PID-Regler im „straightDriveDist-Node“:

```

# calculating effective distance and expected distance
error = self.encoderRight - self.encoderLeft
# pid controller for speed regulation
self.aSpeed += (error * kp) + (sum_error * ki) + (prev_error * kd)

#adding error for next cycle
prev_error = error
sum_error += error

```

8.5 Labirinth-Löse-Algorithums

Der Labirinth-Löse Algorithmus ist Bestandteil des „Main-Nodes“. Dieser trifft die Entscheidung, wo der Roboter als nächstes hinfahren muss und wie er auf dem schnellsten Weg zurückfahren kann. Als erste Priorität nimmt der „Labirinto“ immer die rechte Abzweigung. Gibt es rechts keinen Gang, so fährt der Roboter weiter gerade aus. Ist keine der genannten Richtungen verfügbar dreht sich der „Labirinto“ nach links. Nach dem Entscheidungsprozess werden die jeweiligen „Action-Nodes“ aufgerufen. Die zurückgelegte Route wird gespeichert, um anschliessend den schnellsten Rückweg zu finden.

Left	Front	Right	Decision
False	True	False	1
True	True	False	1
True	False	True	2
True	False	False	0

Tabelle 8.1: TrajectoryList (Decision 0: Left, 1:Front, 2:Right)

Wenn sich der Algorithmus für einen bestimmten Weg entschieden hat, wird die benötigte Action mit der Funktion „actionCalling“ aufgerufen. Im normalen Fahrmodus „normalDrive“ fährt der Roboter mit der „straightDrive-Action“ vorwärts bis eine Abzweigung detektiert wurde. Anschliessend bewegt die „straightDriveDist-Action“ den Roboter bis in die Mitte des Gangs und der „turn-Node“ führt die Drehung durch. Wenn der Roboter keine Wände mehr um sich herum detektiert ist er am Ziel angelangt und wechselt in den „invertedDrive“ Modus. Jede Richtungsentscheidung wird mit einer Nummer versehen (Siehe Tabelle 8.1). Vor der Rückfahrt analysiert der Roboter den gefahrenen Hinweg und löscht sämtliche gefahrenen Umwege. Dies erfolgt durch das Subtrahieren der Werte beim zurückfahren. Die gesamte Routenspeicherung erfolgt durch die „TrajectoryList“. Mit dem Befehl „closeIntersection“ wird dieser Analyseprozess aufgerufen.

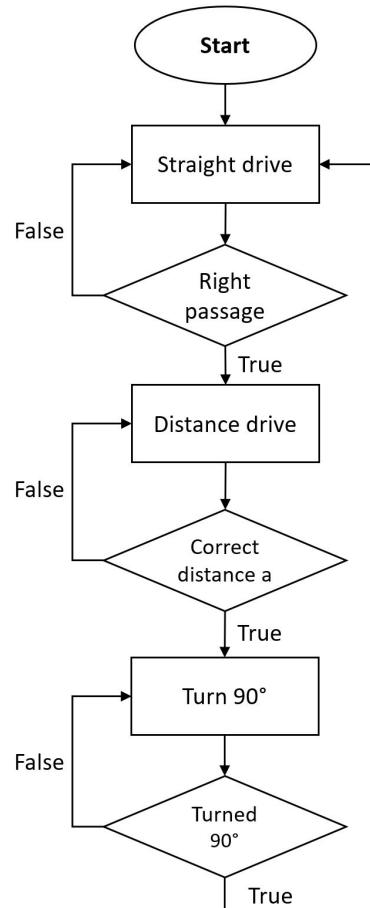


Abbildung 8.2: Ablauf der Abzweigungs-Wahl
[Eigene Darstellung]



9 Systemstests & Debugging

Eine grosse Arbeit bei der Software Entwicklung ist das Debuggen von allen Programme. Als alle „Sensor-Nodes“ und „Action-Nodes“ fertig waren konnte der „Main-Node“ getestet werden. Es gab einige Probleme bei der Kommunikation zwischen den Nodes, da teilweise unkorrekte Bezeichnungen oder falsche Datentypen auf einer „Message“ gesendet wurden. Da sich das Projektteam neu in ROS einarbeiten musste, gestaltete sich die Fehlersuche nicht immer einfach. Dafür musste zuerst die Struktur von ROS möglichst gut verstanden werden, da sich gewisse Protokolle und Message-Konfigurationen in nicht angezeigten Ordnern befanden. Durch die detaillierten Fehlermeldungen konnten aber alle Programmfehler behoben werden.

9.1 Leistungsarme Motoren

Bei der Einstellung der PID-Regelfaktoren wurde bemerkt, dass die Motoren sehr schlecht regeln lassen. Unter einer Geschwindigkeit von 50% haben die Motoren nicht genügend Drehmoment um den Roboter bewegen zu können. Unglücklicherweise ist diese Minimal-Geschwindigkeit bereits sehr schnell. Da nur die Hälfte des Geschwindigkeits-Spektrums zur Verfügung steht, lassen sich die Motoren schwer Regeln. Da der PID-Regler gerne mit Geschwindigkeiten von unter 50% geregelt hätte, mussten Geschwindigkeitsbegrenzungen programmiert werden, damit sich der Roboter nicht in den Stillstand regelt.

Folgendes Code-Beispiel zeigt die schlussendlich gewählten Geschwindigkeits-Limitierungen:

```
# speed limitations
if self.aSpeed > 100:
    self.aSpeed = 100
if self.aSpeed < 60:
    self.aSpeed = 60
if self.bSpeed > 100:
    self.bSpeed = 100
if self.bSpeed < 60:
    self.bSpeed = 60
```

9.2 Ausgelassene Encoder-Counts

Bei den Tests wurde ebenfalls festgestellt, dass die erwartete Encoderwert-Rückgabe nicht mit der effektiven Rückgabe übereinstimmt. Dadurch konnte das gerade fahren nicht gewährleistet werden, da die PID-Regler mit unvollständigen Werten abreiten mussten. Wie bereits erläutert wurde, werden beide Encoders im selben „Node“ verarbeitet und in einer individuellen „Liste“ gespeichert.



Der folgende Code zeigt das Zählen der Counts im Node:

```
# Method : loop for counter
def run(encoderArray):
    bufferArray = [0] * len(encoderArray)
    # loop until node is shutting down

    while not rospy.is_shutdown():
        for i in range(len(encoderArray)):
            # verify if encoders change state
            if GPIO.input(encoderArray[i].encoderPin) != bufferArray[i]:
                encoderArray[i].counter += 1
                bufferArray[i] = GPIO.input(encoderArray[i].encoderPin)
                encoderArray[i].pub.publish(encoderArray[i].counter)
                rospy.loginfo(encoderArray[i].counter)
```

Wenn das Programm in einer *for*-Schleife ist um ein Encoder-Impuls zählen, ist es möglich das zur gleichen Zeit ein Impuls vom anderen Encoder kommt, welcher nicht gezählt wird. Dies wurde als mögliche Fehlerquelle in Betracht gezogen und es wurden zu Testzwecken individuelle „Nodes“ für beide Encoder programmiert. Weitere Systemtests haben jedoch ergeben, dass durch diese Anpassung keine signifikant besseren Messungen erzielt wurden.

Nach intensiven Recherchearbeiten ist das Projektteam der Überzeugung, dass durch die umfangreiche Parallelisierung und Abarbeitung sämtlicher Programme eine Verzögerung entsteht, wobei einige Encoder-Signale nicht rechtzeitig detektiert werden. Bei der Ausführung sämtlicher ROS-Programme liegt die CPU-Auslastung des Raspberry Pi zudem bei über 90%. Diese Hypothese wird ebenfalls dadurch gestützt, dass ROS1 nicht echtzeitfähig ist und dies erst bei ROS2 implementiert wurde. Der Raspberry Pi verfügt ebenfalls über keine speziellen, hochfrequenten Interrupt-Pins, womit eine derart schnelle Signalabfolge eventuell besser erfasst werden können hätte.

9.3 Zuverlässigkeit Ultraschallsensoren

Die Systemtests zeigten zudem, dass die Qualität der Messwerte der Ultraschall-Sensoren nicht optimal ist. Es gab immer wieder fehlerhafte Messwert, die den „Labirinto“ zu Fehlentscheidungen animierten. Ein weiteres Problem ist die geringe Spektrumsabdeckung des Sensors, da nur eine Distanz pro Richtung gemessen wird. Steht der Roboter zudem nicht perfekt rechtwinklig zur Wand, wird der Messwert ebenfalls verfälscht und eine zuverlässige Orientierung ist nicht mehr gewährleistet.



10 Verbesserungen

10.1 Systemfunktion

Die grösste Schwachstelle des „Labirintos“ ist, dass er die Fähigkeit des „Geradeausfahrens“ nicht einwandfrei beherrscht. Die Gründe dafür wurden bereits im Kapitel 9.2 erklärt. Bei der Entwicklung eines weiteren Roboter müssten die Regelalgorithmen auf dezentralen Mikrocontrollern ausgeführt werden. Das bedeutet, dass die komplette Motoransteuerung inklusive PID-Regler durch einen Arduino übernommen werden könnte, welcher nur beispielsweise die Geschwindigkeit der beiden Räder ans ROS weiterleitet. Die Regelung für das geradefahren des Roboters würden direkt auf dem Arduino und dadurch ohne die Einbindung von ROS erfolgen.

Die Distanzmessung der Ultraschallsensoren würde im Optimalfall ebenfalls durch einen Raspberry Pi übernommen, welcher die aufbereiteten Messwerte an ROS sendet. Dadurch müsste ROS nur, auf Grund der empfangenen Daten, den Labyrinth-Löse Algorithmus ausführen und keine Messwertverarbeitung übernehmen.

10.2 Motoren

Da die Motoren bei kleinen Geschwindigkeiten ein zu tiefes Drehmoment aufwiesen um den Roboter noch bewegen zu können, müssen bei einem zukünftigen Projekt zwingend bessere Motoren verwendet werden. Diese Motoren sollten den Roboter auch bei geringer Geschwindigkeit bewegen können, sodass die Geschwindigkeits-Regelung im gesamten Spektrum von 0-100% gemacht werden kann. Aktuell war dies nur im halben Spektrum möglich.

10.3 Navigation

Zukünftig wäre eine flächendeckende Distanzmessung ebenfalls notwendig, damit der Roboter seine Umgebung nicht nur aus drei unzuverlässigen Distanzmessungen interpretieren muss. Dazu wäre die Integration eines kostengünstigen LIDAR-Sensors denkbar. Dieser Sensor scannt die Umgebung mit einem drehbaren, laser-basierten Distanzmessgerät. Der Roboter erhält dadurch kontinuierlich ein präzises zweidimensionales 360°Lagebild. Derartige Sensoren sind heutzutage bereits ab 100.- CHF erhältlich und hätten dadurch trotz des geringen Beschaffungsbudget verwendet werden können.



11 Fazit

Im Rahmen der Semesterarbeit konnte das Projektteam ein funktionierendes ROS-System und zwei Roboter entwickeln. Die einwandfreie Funktionalität des Labyrinth-Löse-Algorithmus konnte während den Systemtests ebenfalls nachgewiesen werden. Das Projektteam konnte jedoch keine fehlerfreie Spurhaltung des Roboters erreichen. Dadurch konnte der „Labirinto“ nie das gesamte Labyrinth autonom durchfahren. Als Rekord konnten während den Systemtests zwei Abzweigungen und eine Sackgasse erfolgreich autonom erkundet werden.

Das Projektteam konnte sich während dem Projekt sehr viel neues Wissen rund um das Robot Operating System und die Entwicklung von Robotern im Allgemeinen aneignen. Aus Sicht der beiden Studierenden ist das erreichte Resultat trotz allem ein grosser Erfolg mit vielen bleibenden Erinnerungen. Eines der Highlights war der Besuch an der Online-Fachtagung ROS-World 2020, wo sich das Projektteam mit erfahrenen Robotik-Ingenieuren austauschen konnte.

Die durch COVID-19 bedingte Situation war optimal für die Erarbeitung eines solchen Projekts. Das Projektteam musste dadurch seine erstklassige Teamfähigkeit speziell unter Beweis stellen. Es wurde immer sehr kollegial und auf einer hohen Vertrauensbasis gearbeitet. Aufgetretene Probleme wurden trotz der grossen räumlichen Entfernung stets gemeinsam gelöst.

Im Anschluss an das Projekt wird trotzdem noch versucht eine funktionierende Spurhalte-Regelung zu implementieren, damit „Labirinto“ schlussendlich doch sein Ziel am anderen Ende des Labyrinths erreicht. Das gewonnene Wissen rund um das Thema ROS und Robotik-Entwicklung wird beim Trinatronic Wettbewerb im 6. Semester erneut angewendet.



Literatur

- [1] Dirk Bindmann. Esperanto heute. http://www.esperanto.net/info/detala/de_eo-detala.html. Accessed: 02.02.2021.
 - [2] Bastelgarage. HC-SR04 Ultrasonic Distance Sensor. <https://www.distrelec.ch/en/hc-sr04-ultrasonic-distance-sensor-sparkfun-electronics-sen-15569/p/30160395>. Accessed: 02.02.2021.
 - [3] Digitec. MPU-6050. https://www.digitec.ch/de/s1/product/mpu-6050-sensor-elektronikmodul-8193998?gclid=Cj0KCQiA88X_BRDUARIgACVMYD_cKQ3zPYXTOMOUj3Y4y95QnXTAxpw3Q9Eb7E_thuRw0a0SP3yKFbEaAhTyEALw_wcB&gclsrc=aw.ds. Accessed: 02.02.2021.
 - [4] Bastelgarage. Getriebemotor DG01D-E 1:48 mit Encoder. <https://www.bastelgarage.ch/bauteile/stepper-motoren/getriebemotor-dg01d-e-1-48-mit-encoder>. Accessed: 02.02.2021.
 - [5] Bastelgarage. Motor Triebler - Dual 1.2A TB6612FNG. <https://www.bastelgarage.ch/motor-treibler-dual-1-2a-tb6612fng>. Accessed: 02.02.2021.
 - [6] Distrelec. Raspberry Pi 4 1.5GHz Quad-Core. http://www.esperanto.net/info/detala/de_eo-detala.html. Accessed: 02.02.2021.
 - [7] Bastelgarage. M4x18650 Lithium Batterie Shield. <https://www.bastelgarage.ch/solar-lipo/4x18650-lithium-batterie-shield-5v-3a-3v-1a>. Accessed: 02.02.2021.
 - [8] Roland Siegward. *Introduction to Autonomous Mobile Robots*. Massachusetts Institute of Technology, 2011.
 - [9] Ekbert Hering. *Elektronik für Ingenieure und Naturwissenschaftler*. Springer-Verlag Berlin Heidelberg, 2005.
 - [10] Murat Calis. *Roboter mit ROS*. dpunkt.verlag, 2020.

Tabellenverzeichnis

8.1 TrajectoryList (Decision 0: Left, 1:Front, 2:Right) 23



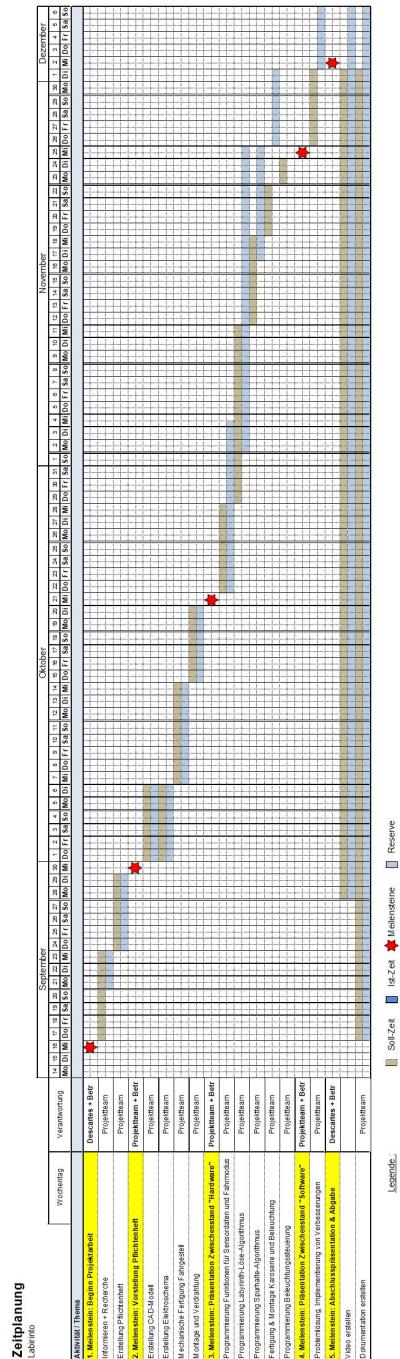
Abbildungsverzeichnis

3.1	Zielkatalog für die Entwicklung von „Labirinto“	5
3.2	Use-Case Diagramm	6
3.3	Übersicht Meilensteine	7
4.1	Mechatronisches Blockschaltbild des „Labirintos“	8
5.1	Topologieschema des „Labirintos“	9
5.2	Ultraschallsensor HC-SR04	10
5.3	Beschleunigungssensor MPU6050	10
5.4	Antriebsmotor DG01D-E	11
5.5	Motortreiber TB6612FNG	11
5.6	Raspberry Pi 4B 8GB [6]	12
5.7	Akku-Module Diymore.cc	12
5.8	Spannungswandler XL6009	12
5.9	Dreidimensionale Skizze des „Labirintos“	13
5.10	Fahrgestell in der Ansicht von Rechts	13
5.11	Plattform A und B in der Ansicht von Oben	14
5.12	Kostenberechnung	14
6.1	Plattform A und B in der Ansicht von oben	15
6.2	Test-Strecke für PID-Regeltests	15
7.1	CAD-Modell des „Labirinto Due“	16
7.2	Montage der Plattformen	17
7.3	Hinterleuchtetes Labirinto-Logo	17
7.4	Schaltplan des „Labirinto Due“	18
7.5	Verteilplatine	19
8.1	ROS-Struktur im „Labirinto“	21
8.2	Ablauf der Abzweigungs-Wahl	24



12 Anhang

12.1 Zeitplan





12.2 Use-Cases

12.2.1 Use-Case 1

USE CASE Nummer	Labyrinth-Lösen 1
Ziel	Labyrinth lösen und Rückweg optimieren
Kategorie	primär
Vorbereitung	Roboter ist über SSH mit Entwicklungsrechner verbunden. Akku von Roboter ist vollständig geladen
Nachbedingung falls erfolgreich	Roboter fährt autonom vom Start zum Ziel eines Labyrinths und anschliessend auf dem schnellsten Weg zurück
Nachbedingung falls Fehlschlag	Wechsel in Manuellen Fahrmodus
Hauptakteure	Bediener, Kunde
Nebenakteure	Zuschauer
Auslöser	Startbefehl per Knopfdruck
Hauptszenario	Roboter am Labyrinth-Eingang positionieren Startbefehl per Knopfdruck Autonomes Lösen des Labyrinths Autonomes Zurückfahren durch Labyrinth auf schnellstem Weg Wartemodus
Alternative	Roboter mit Entwicklungsrechner verbinden Roboter am Labyrinth-Eingang positionieren Startbefehl an Roboter per SSH übermitteln Autonomes Lösen des Labyrinths Autonomes Zurückfahren durch Labyrinth auf schnellstem Weg Wartemodus

12.2.2 Use-Case 2

USE CASE Nummer	Ferngesteuertes Fahren 2
Ziel	Manuelles Fahren durch SSH-Befehle
Kategorie	optional
Vorbereitung	Roboter ist über SSH mit Entwicklungsrechner verbunden. Akku von Roboter ist vollständig geladen
Nachbedingung falls erfolgreich	Roboter fährt manuell durch definierte Richtungsbefehle, welche per SSH an Roboter übermittelt werden.
Nachbedingung falls Fehlschlag	Ausgabe von Fehlermeldung
Hauptakteure	Bediener, Kunde
Nebenakteure	keine
Auslöser	Übermittlung Startbefehl per SSH an Roboter
Hauptszenario	Roboter mit Entwicklungsrechner verbinden Roboter auf freier Fläche platzieren Manuelles Fahren Wartemodus