

Entwurf und Implementierung eines Corona Robots mit Raspberry Pi

Hausarbeit

vorgelegt von

Ghassen Jamoussi Wadi Touil

Mtr.Nr.: 910628 910562

dem Fachbereich VI – Informatik und Medien –
der Beuth Hochschule für Technik Berlin

Studiengang
Technische Informatik

Tag der Abgabe 28. August 2021

Betreuer

Prof. Markus Schubert

Inhaltsverzeichnis

1 Einleitung	1
1.1 Motivation	1
1.2 Ziel	1
1.3 Gliederung der Arbeit	2
1.4 Arbeitsmethode	2
2 Theoretische Grundlagen	3
2.1 Raspberry Pi	3
2.1.1 GPIO Pins	4
2.2 Internet of Things (IOT)	4
2.3 Linux und Robotik	5
3 Analyse und Entwurf	6
3.1 Systembeschreibung und Anforderungen	6
3.1.1 Textuelle Beschreibung des Systems	6
3.1.2 Diagramme zum Verständnis des Aufbaus	9
3.1.3 Liste von Anforderungen an das System	9
3.1.4 Generelle Rahmenbedingungen	11
3.2 Anwendungsfalldiagramm und -beschreibungen	12
3.2.1 UML Anwendungsfalldiagramm	12
3.2.2 Beschreibung des Anwendungsfalldiagramms	12
3.3 Flowchartdiagramm	13
3.3.1 Erklärung des Flowchartdiagramms	13
3.3.2 Erklärung der Funktionalitäten des Flowchartdiagramms	14
4 Implementierung	15
4.1 Setup und Komponententest	15
4.1.1 Raspberry Pi	15
4.1.2 Motor Driver Module	16
4.1.3 Line follower Sensor	19
4.1.4 Abstandssensor	21
4.1.5 Audiogerät	24
4.1.6 Wärmebildkamera	26
4.1.7 Kamera	30
4.2 Integration	35
4.2.1 Graphical User Interface (GUI)	35
5 Zusammenfassung und Ausblick	39
5.1 Zusammenfassung	39
5.2 Ausblick	39
Literatur- und Quellenverzeichnis	40
Abbildungsverzeichnis	41
Tabellenverzeichnis	43

1 Einleitung

Im Jahr 2020 waren fast alle Länder und mehr als 50 Millionen Menschen auf der ganzen Welt von COVID-19 betroffen. Die Regierungen agieren in einem Kontext radikaler Unsicherheit und müssen angesichts der damit verbundenen gesundheitlichen, wirtschaftlichen und sozialen Herausforderungen schwierige Kompromisse eingehen. Im Frühjahr 2020 war mehr als die Hälfte der Weltbevölkerung von einem Lockdown mit starken Eindämmungsmaßnahmen betroffen. Abgesehen von der gesundheitlichen und menschlichen Tragödie des Coronavirus ist inzwischen allgemein anerkannt, dass die Pandemie die schwerste Wirtschaftskrise seit dem Zweiten Weltkrieg ausgelöst hat.

Die Menschen werden seit Ende Dezember 2020 geimpft, aber Studien haben gezeigt, dass auch nach einer vollständigen Impfung noch die Möglichkeit besteht, sich zu infizieren.

Eines der ersten und häufigsten Symptome von Covid ist Fieber. Außerdem gehen die Menschen während der Ausgangssperre aus und unterhalten sich mit ihren Freunden, ohne dass es ihnen etwas ausmacht, wenn die Stadt von der Polizei abgesperrt wird.

Aufgrund der mangelnden Kontrolle auf der Straße und der häufigen Symptome von Covid, kamen wir auf die Idee, dieses Projekt zu realisieren.

1.1 Motivation

Seit Beginn der Corona-Krise macht man sich Gedanken darüber, wie man die Zahl der Infizierten reduzieren kann. Als Technische Informatiker, haben wir uns überlegt, wie man sowohl Hardware als auch Software einsetzt, um diese kritischen Zahlen reduzieren zu können.

1.2 Ziel

Das Ziel, ist es ein Roboter zu entwickeln, der autonom und ferngesteuert fährt und eine Thermo-kamera hat, um die Temperatur der Menschen zu erkennen. Wenn diese höher als 39 Grad ist, wird er ein Signal erzeugen und dem betreffenden Menschen mitteilen, dass er so schnell wie möglich sich testen lässt.

Während der Ausgangssperre kann dieser Roboter die Straße mit einer Kamera überwachen und ein Signal erzeugen, sobald das System Menschen auf der Straße erkennt.

Dieser Roboter wird sehr effektiv sein, um die Zahl der Corona-Fälle noch weiter zu senken und die Sicherheit für alle Bürger zu gewährleisten.

1.3 Gliederung der Arbeit

Die Arbeit besteht aus sechs Teilen, von denen der erste die Einleitung ist, die einen kurzen Überblick über das Problem und das Thema gibt und das Ziel der Arbeit definiert.

Der zweite Teil der Aufgabe besteht aus dem theoretischen Grundlagen. Darin werden der Raspber Pi und das Internet der Dinge erklärt, die für das spätere Verständnis der Arbeit wichtig sind.

Die Analyse und der Entwurf des zuvor definierten Ziels sind die Bausteine des dritten Teils. Zu diesem Zweck werden die Beschreibung und die Anforderungen an das System anhand einer textlichen Beschreibung des Systems, Diagrammen zum Verständnis des Aufbaus, einer Liste von Anforderungen an das System und allgemeinen Rahmenbedingungen erläutert. Anschließend wird das Anwendungsfalldiagramm anhand eines Diagramms und einer UML-Beschreibung erläutert. Zum Schluss dieses Abschnitts wird die Funktionsweise des Corona-Roboters anhand eines Flowchartdiagramms und einer Beschreibung verdeutlicht.

Auf der Grundlage der Ergebnisse des dritten Teils wird im vierten Abschnitt die Entwurfsentscheidung anhand der Basisfunktionalität umgesetzt. Die Komponenten werden zunächst mit Python-Code getestet und dann in ein System integriert, das mit einer grafischen Benutzeroberfläche realisiert wird.

Der letzte Teil der Arbeit enthält die Reflexion über die zu Beginn der Arbeit definierten Ziele. Hier werden eine Zusammenfassung des Projekts erläutert und ein Ausblick auf mögliche zukünftige Erweiterungen gegeben.

1.4 Arbeitsmethode

Die Methodologie der Arbeit ist sehr wichtig, um die Prozesskette des Projekts zu beschleunigen und die Produktqualität zu erhöhen. In diesem Abschnitt stellen wir die angewandte Methodik, den Entwicklungszyklus und den formalisierten Entwurf vor, die wir verwendet haben.

Wir haben uns für Kanban entschieden, da es sich zu einem Rahmen entwickelt, in dem Aufgaben visuell dargestellt werden, während sie durch Spalten auf einer Kanban-Tafel fortschreiten. die Aufgaben des Projekts werden aus dem vordefinierten Auftragsbestand entnommen und durch die Spalten auf der Tafel verschoben, wobei jede Spalte eine Phase des Prozesses darstellt.

Kanban eignet sich hervorragend, um jedem einen sofortigen visuellen Überblick darüber zu verschaffen, wo sich die einzelnen Arbeiten zu einem bestimmten Zeitpunkt befinden.

Diese Projektmanagement-Methode wird verwendet, wenn die Prozesse eines Projekts einfach sind und nicht viele Stufen haben, was bei uns der Fall ist. Kanban ist eine einfache Methode innerhalb des agilen Projektmanagements und wir denken, dass die beste Auswahl für unser Projekt ist.[1]

Corona Robot

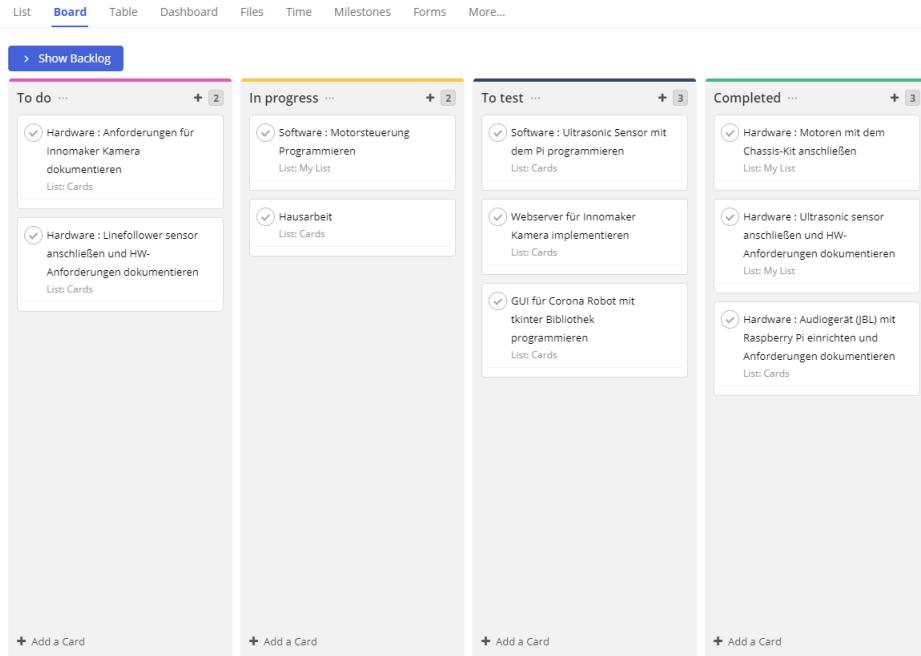


Abbildung 1: Beispiel von unserem Kanban Board

2 Theoretische Grundlagen

2.1 Raspberry Pi

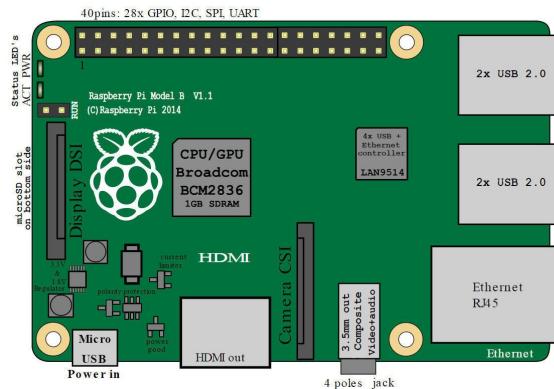


Abbildung 2: Raspberry Pi

Der Raspberry Pi ist ein preiswerter, kreditkartengroßer Computer, der an einen Computermonitor oder Fernseher angeschlossen wird und eine Standardtastatur und -maus verwendet. Es ist ein leistungsfähiges kleines Gerät, mit dem Menschen aller Altersgruppen die Computerwelt erkunden und das Programmieren in Sprachen wie Scratch und Python erlernen können. Er kann alles, was man von einem Desktop-Computer erwartet, vom Surfen im Internet und der Wiedergabe von hochauflösenden Videos bis hin zur Erstellung von Tabellenkalkulationen, Textverarbeitung und Spielen.

Darüber hinaus ist der Raspberry Pi in der Lage, mit der Außenwelt zu interagieren, und wurde in

einer Vielzahl von digitalen Maker-Projekten eingesetzt, von Musikmaschinen und Elterndetektoren bis hin zu Wetterstationen und zwitschernden Vogelhäusern mit Infrarotkameras. Wir möchten, dass der Raspberry Pi von Kindern auf der ganzen Welt genutzt wird, um das Programmieren zu lernen und zu verstehen, wie Computer funktionieren. [2]

2.1.1 GPIO Pins

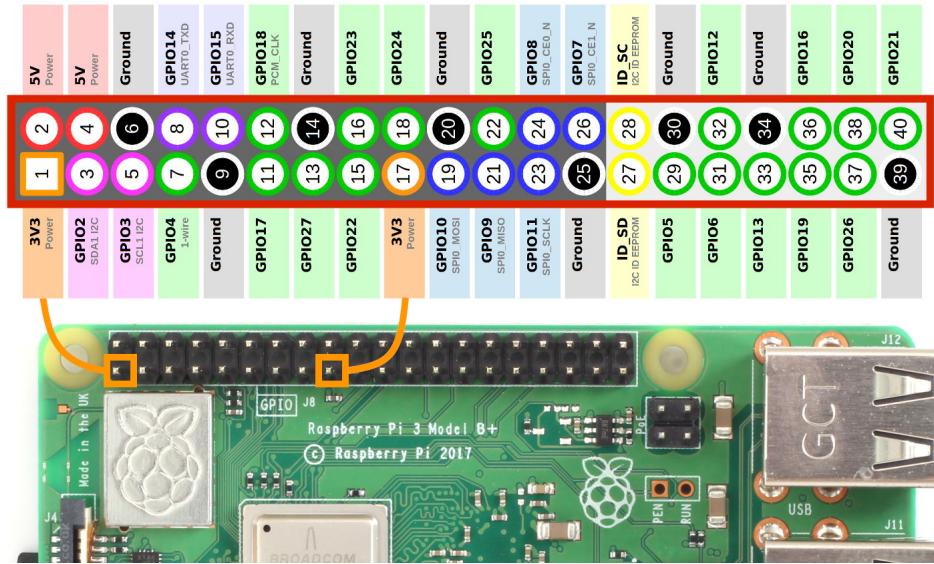


Abbildung 3: GPIO Pins

Eine leistungsstarke Funktion des Raspberry Pi ist die Reihe der GPIO-Pins am oberen Rand der Platine. GPIO steht für General-Purpose Input/Output. Diese Pins sind eine physische Schnittstelle zwischen dem Raspberry Pi und der Außenwelt. Auf der einfachsten Ebene kann man sie sich als Schalter vorstellen, die man ein- oder ausschalten kann (Eingabe) oder die der Pi ein- oder ausschalten kann (Ausgabe).

Die GPIO-Pins ermöglichen es dem Raspberry Pi, die Außenwelt zu steuern und zu überwachen, indem sie mit elektronischen Schaltungen verbunden werden. Der Pi ist in der Lage, LEDs zu steuern, sie ein- oder auszuschalten, Motoren zu betreiben und viele andere Dinge. Er kann auch erkennen, ob ein Schalter gedrückt wurde, die Temperatur und das Licht. Wir bezeichnen dies als Physical Computing.

Der Raspberry Pi verfügt über 40 Pins (26 Pins bei frühen Modellen), die verschiedene Funktionen erfüllen. [3]

2.2 Internet of Things (IOT)

Das Internet der Dinge bezeichnet das Netzwerk und die Verbindung verschiedener Geräte mit datenerfassender Hardware (wie Sensoren), die Informationen austauschen können. Der Sinn des IoT besteht darin, dass die einzelnen Geräte miteinander interagieren und zusammenarbeiten können, um Ihnen, dem Nutzer, die beste Erfahrung zu bieten.

Raspberry Pi-Boards sind äußerst beliebte Single-Board-Computer (SBCs), die sich aufgrund ihrer geringen Größe und ihrer umfassenden Fähigkeiten gut für den Bau von IOT-Geräten eignen. Es gibt viele verschiedene Modelle von Raspberry-Pi-Platinen, mit unterschiedlichen Kombinationen von Anschlüssen und Sensoren.

Laut Wikipedia wird das IOT wie folgt definiert:

"...das Netzwerk physischer Objekte - Geräte, Fahrzeuge, Gebäude und andere Gegenstände -, die mit Elektronik, Software, Sensoren und Netzwerkverbindungen ausgestattet sind, die es diesen Objekten ermöglichen, Daten zu sammeln und auszutauschen".

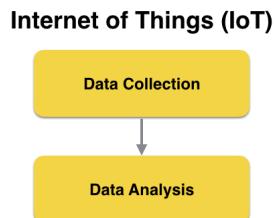


Abbildung 4: IOT Konzept

2.3 Linux und Robotik

Linux ist ein Unix-basiertes Betriebssystem. Es ist sehr beliebt bei Programmierer und Informatiker, weil es einfach, unkompliziert und geradlinig ist. Doch für viele andere, kann Linux sehr herausfordernd sein.

Warum also wähle ich diese Umgebung für die Entwicklung meines Robots ? Die Antwort auf diese Frage ist dreifach. Erstens, wenn man mit Robotik arbeitet, muss man sich irgendwann mit Linux beschäftigen. Das ist einfach eine Tatsache. Man könnte eine Menge tun, ohne jemals einen einzigen sudo Befehl einzugeben, aber man wird nur begrenzte Möglichkeiten haben. Der sudo-Befehl steht für super user do in Linux. Damit wird dem Betriebssystem mitgeteilt, dass wir im Begriff sind eine geschützte Funktion ausführen zu wollen, die mehr als den allgemeinen Benutzerzugriff erfordert.

Zweitens ist Linux eine Herausforderung. Wenn man aber bereits mit Linux gearbeitet hat, dann trifft dieser Grund nicht auf sich zu. Wenn man jedoch neu bei Linux, dem Raspberry Pi oder der Arbeit mit einer Kommandozeile arbeitet, dann werden einige der Dinge, eine Herausforderung sein.

Drittens, und das ist bei weitem das Wichtigste, verwendet der Raspberry Pi Linux. Man könnte andere Betriebssysteme auf dem Pi installieren, aber er wurde für die Verwendung von Linux entwickelt und vorgesehen. Tatsächlich hat der Raspberry Pi seine eigene Linux-Variante namens Raspbian. Dies ist das empfohlene Betriebssystem, also werden wir es auch verwenden. Einer der Vorteile der Verwendung eines vorgefertigten vorinstallierten Betriebssystems ist neben der Benutzerfreundlichkeit, dass viele Tools bereits installiert und einsatzbereit sind. Da wir Linux verwenden, werden wir die Befehlszeilenanweisungen ausgiebig nutzen.

3 Analyse und Entwurf

3.1 Systembeschreibung und Anforderungen

3.1.1 Textuelle Beschreibung des Systems

Der Roboter, den wir bauen werden, ist ein Roboter mit vier Rädern. Er wird in der Lage sein, Hindernisse zu erkennen und einer Linie zu folgen. Wir werden auch die Möglichkeit bieten, den Roboter über SSH fernzusteuern.

Die besonderen Merkmale, die diesen Roboter von anderen unterscheiden, sind, dass er eine normale und eine Wärmebildkamera hat. Die Wärmebildkamera wird in der Lage sein, Temperaturen zu erkennen und uns dann die Daten über den Raspberry Pi zu senden. Die normale (Innomaker) Kamera wird Live-Videos über einen Webserver streamen.

Materialien :

Zum größten Teil habe ich versucht, die Liste der Materialien so allgemein wie möglich zu halten. Es gibt ein paar Artikel, die anbieterspezifisch sind. Ich habe sie ausgewählt, weil sie eine Menge Funktionalität und Bequemlichkeit bieten. Die meisten Teile habe ich bei Amazon gekauft. Das Chassis-Kit ist aber von einem anderen Online-Händler namens banggood, der viele mechanische Teile für die Robotik herstellt.

Hier sind die Materialien, die wir verwendet haben :

- 2 x Auto Chassis (Acryl)
- 4 x DC-Getriebemotor (starker Magnet, Prüfung durch EMV)
- 4 x 20-Zeilen-Geschwindigkeitsgeber
- 4 x Rad
- 8 x Verschluss
- Schraube, Mutter und Kupfersäule



Abbildung 5: Robot Kit

Noch weitere Hardware Komponenten :

- Raspberry Pi : Um unser Roboter zu programmieren
- L298N Motor Driver Module : Um der Roboter zu steuern
- 2 x IR Sensor : Damit der Roboter in der Lage ist, eine Linie zu verfolgen
- Innomaker Kamera : Um die Straße zu überwachen
- AMG88 Wärmebildkamera : Um Temperaturen zu erkennen
- Ultrasonic sensor : Um Hindernisse zu erkennen
- 4 x 1.5 Batterien : Um unsere Motoren anzutreiben
- 3.7-5V Batterie : Um Raspberry Pi anzutreiben
- Kabeln : 40 x 20CM Female-Female, Male-Female, Male-Male

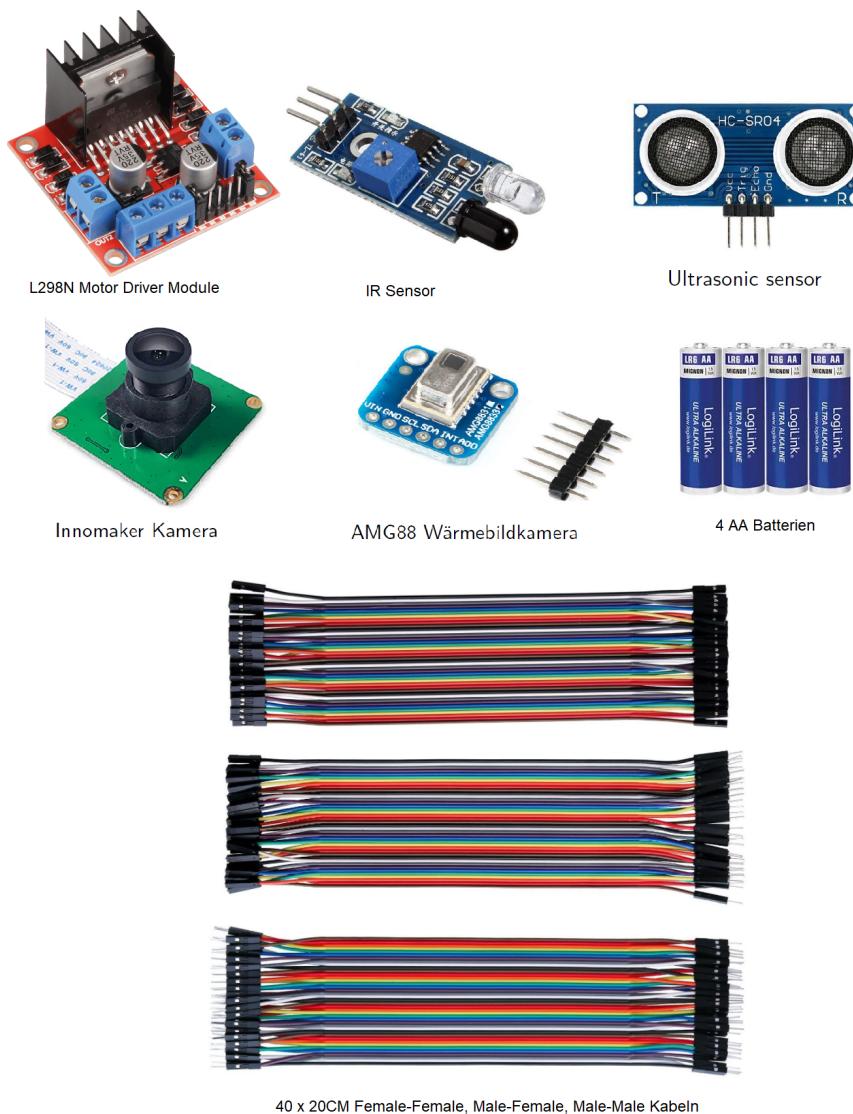


Abbildung 6: Hardware Komponenten

Verwendete Materialien, um Hardware zusammenszuschließen oder zu debuggen :

- 1 x Digitalmultimeter, 1 x Lötkolben.
- 1 x Entlötpumpe, 1 x Lötkolbenhalter,
- 5 x Lötspitzen, 1 x Digitalmultimeter.
- 1 x antistatische Pinzette.
- AMG88 Wärmebildkamera
- 2 x elektronische Drähte.
- 2 x Stifte für das Multimeter.
- 1 x Schraubendreher.
- 1 x Abisolierzange.
- 1 x Leitungsrohr.



Abbildung 7: Elektronische Materialien

3.1.2 Diagramme zum Verständnis des Aufbaus

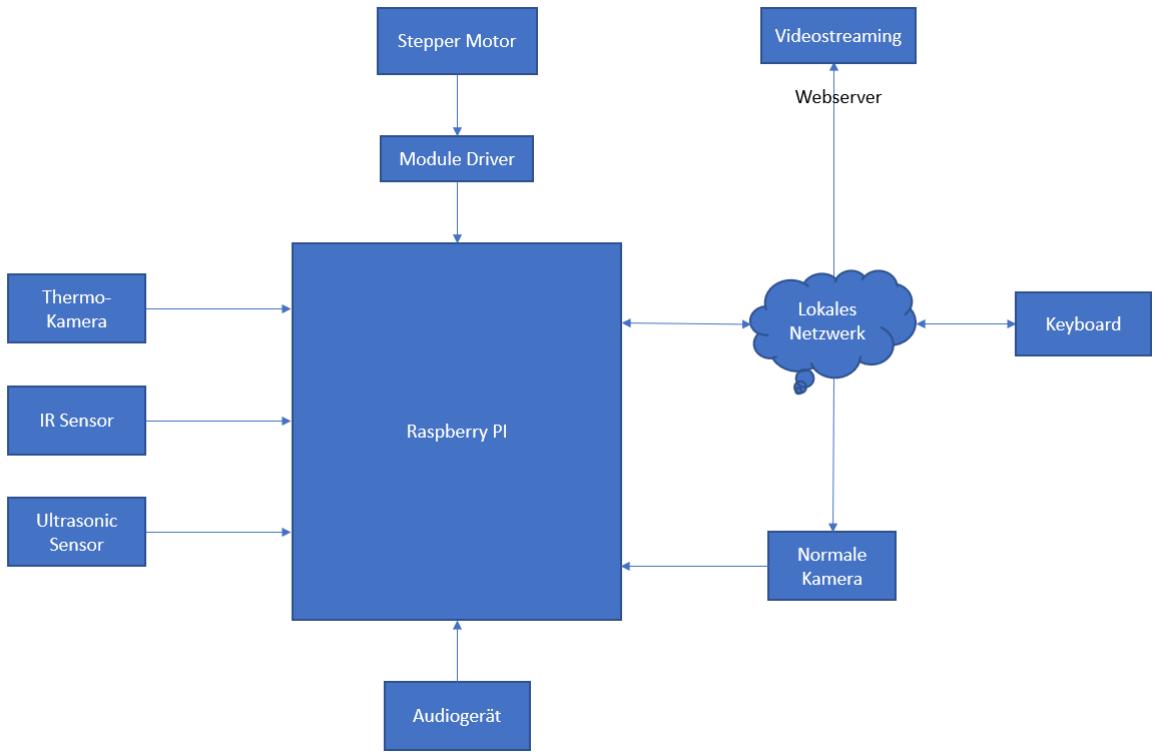


Abbildung 8: Abstrakter Aufbau des Robots

3.1.3 Liste von Anforderungen an das System

- Ein autonomes Auto, das 2 verschiedene Kameras hat; eine normale und eine Wärmekamera.
- Das Auto wird einige Komponenten für bessere Sicherheit und Zuverlässigkeit haben (um Autounfälle zu vermeiden).
- Dieses Auto fährt autonom, indem es eine schwarze Linie erkennt und ihr folgt.
- Wenn das Auto der schwarzen Linie folgt, kann es zwischenzeitig die Temperaturen der Menschen erkennen.
- Das Auto besitzt einen Audioausgang, um die Töne zu erzeugen.
- Wenn jemand eine Temperatur von über 39 Grad hat, hält das Auto an und fragt den Beifahrer nach, ob es ihm gut geht und ob er sofort zum Arzt gehen soll. Der Arzt kann dann entscheiden, ob der Mensch sich auf das Corona-Virus testen lassen muss.

Was wir erwartet haben :

- Kosten : Weniger als 150 Euros insgesamt (RPI , Stromversorgung , SD-Karte , Leds , Widerstände, Ultraschallsensor, L298N Modul, Raspi Cam, AMG88 Wärmebildkamera Low-Budget-Lautsprecher, Komponenten des Autos)
- Nur die GPIOs vom Raspi 3 zu verwenden (Die sind ausreichend für unser Projekt)
- Umsetzungszeit : 1 bis 2 Monate

Echte Ergebnisse :

- Kosten

Hardware Komponente	Preis	Anmerkung
Raspberry Pi	35,89 €	Raspberry Pi 3 Model B ARM-Cortex-A53 4x 1.2GHz, 1GB RAM, WiFi, Bluetooth, LAN, 4x USB
5 x L298N Motor Driver Module	€9.99	Hier haben wir eine Packung für 5 Teile gekauft, da die im Angebot waren
2 x IR Sensor		ausgeliehen
Innomaker Kamera	€13,99	
2 x AMG88 Wärmebildkamera	€29,99	Hier haben wir zwei AMG88 gekauft um Probleme zu vermeiden
Ultrasonic Sensor		ausgeliehen
8 x 1.5 Batterien	€6,00	
Powerbank/Batterie 3.7V	€20,00	Zusammengekauft
Kabeln 40 x 20CM Female-Female, Male-Female, Male-Male	€5,00	
Schwarzer Kleberband	€5,80	
Summe	€121,66	

Tabelle 1: Kosten der Komponenten

Wir hatten bereits andere Komponenten dabei, also wurden diese nicht in die Kosten eingerechnet.

Die erwarteten Kosten betrugen 150 Euro, aber wir haben es geschafft, unter diesem Grenzwert zu bleiben, wodurch wir das Projekt mit einem Budget von 121 Euro realisieren konnten, was eigentlich ziemlich gut ist.

- GPIO Pins

Wir konnten uns auch auf die GPIOs des Raspberry Pi 3 beschränken und keine zusätzlichen GPIOs von anderen Raspberry Pis verwenden.

- Umsetzungszeit

Die Umsetzungszeit lag zwischen 2 und 3 Monaten und das ist fast wie erwartet. Das Problem lag hier bei der Lieferung. Einige Produkte waren in Deutschland nicht verfügbar und sollten aus China kommen. Deshalb hat es ein bisschen gedauert, das Projekt rechtzeitig abzuschließen.

3.1.4 Generelle Rahmenbedingungen

- Gesetzliche Bedingung :

In Deutschland ist es im Grunde genommen illegal, Videos von Fremden auf der Straße aufzunehmen, ohne eine Genehmigung von ihnen zu erhalten. Aber auf unserer Seite werden wir keine Videos aufnehmen, sondern eher Videos streamen. Das bedeutet, dass alle gestreamten Videos dauerhaft gelöscht und nicht gespeichert werden können.

Wir müssen also nur eine Vereinbarung mit der Regierung treffen, in der wir ausführlich erklären, dass wir keine Videos aufzeichnen, sondern streamen werden, was etwas völlig anderes ist, weil sie nicht auf unseren Festplatten gespeichert werden.

- Technische Bedingung :

- Gewährleistung der Sicherheit des Roboters durch Ultrasonic Sensoren zur Vermeidung von Autounfällen.
- Anwesenheit von Menschen im kritischen Bereich der Entwicklung.
- Überprüfung aller Kabel, ob sie sicher miteinander verbunden sind.
- Sicherstellen, dass es keinen Kurzschluss gibt.

3.2 Anwendungsfalldiagramm und -beschreibungen

3.2.1 UML Anwendungsfalldiagramm

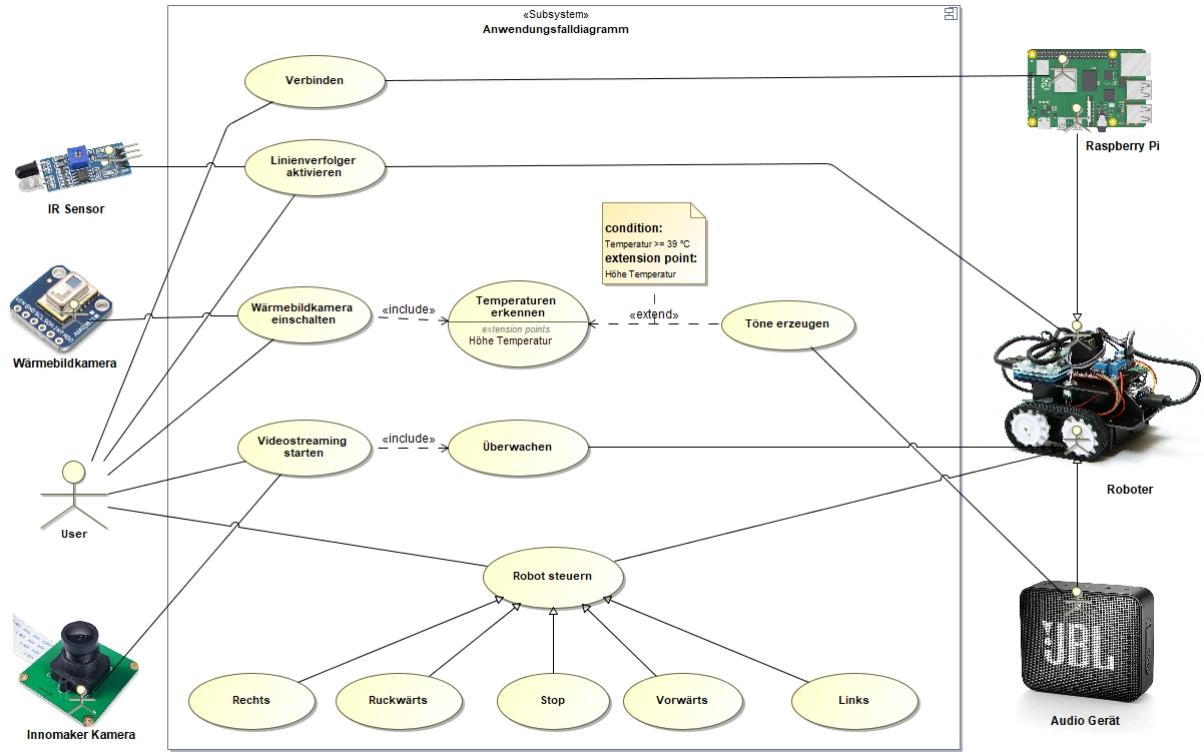


Abbildung 9: Anwendungsfalldiagramm des Corona-Robots

3.2.2 Beschreibung des Anwendungsfalldiagramms

Das Anwendungsfalldiagramm hilft uns zu verstehen, wie der Benutzer mit dem von Ihnen entworfenen System interagieren könnte. Und schließlich sollte es uns helfen, Anforderungen besser zu definieren und zu organisieren.

Der Benutzer hat die Möglichkeit, der Raspberry Pi bzw. der Roboter via VNC Viewer/SSH zu verbinden.

Sobald die Verbindung erfolgreich durchgeführt werden, hat der User verschiedene Features zum Auswahl, wie zum Beispiel :

- Robot steuern : Rechts / Rückwärts / Stop / Vorwärts / Links
- Videostreaming starten
- Wärmebildkamera einschalten
- Linienverfolger aktivieren

3.3 Flowchartdiagramm

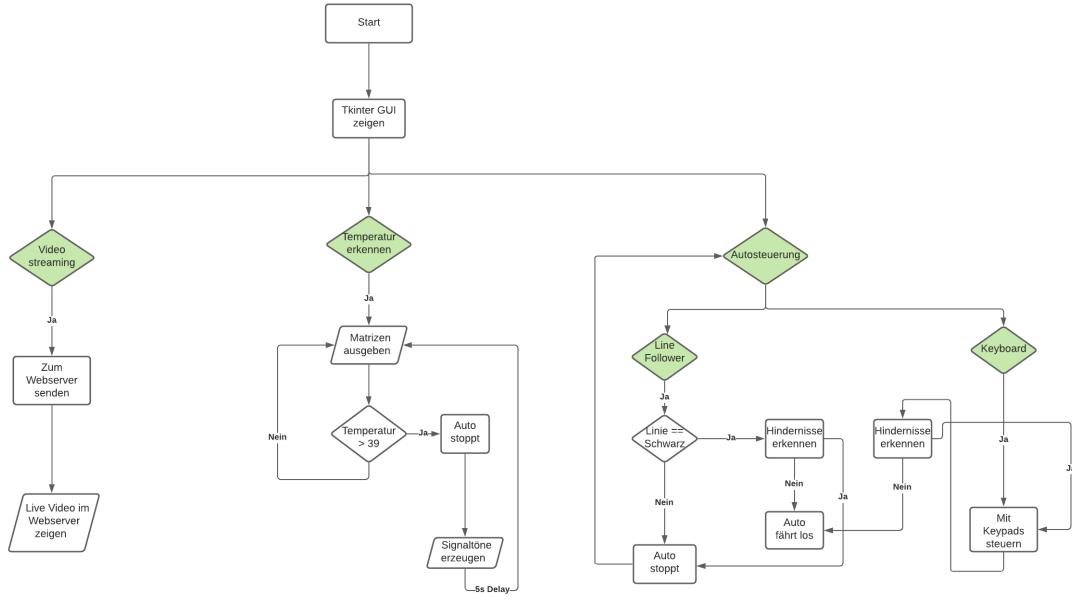


Abbildung 10: Flowchartdiagramm des Corona Robots

3.3.1 Erklärung des Flowchartdiagramms

Ein Flowchart ist ein Diagramm, das einen Prozess, ein System oder einen Computeralgorithmus abbildet. Es wird in vielen Bereichen eingesetzt, um oft komplexe Prozesse in klaren, leicht verständlichen Diagrammen zu dokumentieren, zu untersuchen, zu planen, zu verbessern und zu kommunizieren.[4] Deswegen haben wir das in unserem Projekt implementiert, um die Funktionsweise des Roboters klarer darzustellen.

Unser Roboter verwendet eine einfache grafische Benutzeroberfläche (GUI), um seine Funktionalitäten zu verbinden. Über die Tkinter-GUI können wir das Auto, die Temperaturmessung oder das Videostreaming zu einem Webserver steuern.

Wie Sie aus dem Flowchart Diagramm ersehen können, besteht der Abschnitt, der sich auf die Steuerung des Roboters ("Autosteuerung") bezieht, aus 2 Teilen. Einer durch eine Linie mit einem Line-Follower-Sensor und der andere durch die Tastatur. Der Benutzer hat die Möglichkeit, zwischen beiden zu wählen, aber er kann sie nicht gleichzeitig verwenden.

Die GUI, die wir erstellt haben, unterstützt jedoch einige Multithreading-Prozesse während der Steuerung des Robots. Zum Beispiel können wir einen Live-Stream an einen Webserver senden, die Temperatur messen und die Roboter gleichzeitig steuern, indem wir die gewünschten Funktionen in der Tkinter-GUI zusammen anklicken.

3.3.2 Erklärung der Funktionalitäten des Flowchartdiagramms

- Autosteuerung
 - Keyboard : Hier hat man die Möglichkeit, den Roboter mit Keypads zu steuern :
KEY_UP = Auto fährt vorwärts.
KEY_DOWN = Auto fährt rückwärts.
KEY_RIGHT = Auto fährt nach rechts.
KEY_LEFT = Auto fährt nach links.
KEY_x = Auto stoppt.
KEY_q = Steuerung mit dem Keypads beenden.
Während der Steuerung prüft das Auto, ob sich Hindernisse vor ihm befinden. Wenn dies nicht der Fall ist, fährt das Auto problemlos mit den Keypads los.
 - Line Follower : Hier folgt das Auto automatisch einer schwarzen Linie.
Eine nicht schwarze Linie = Auto stoppt/fährt nicht.
Eine schwarze Linie = Auto fährt los.
Während das Auto durch eine schwarze Linie fährt, prüft es, ob sich Hindernisse vor ihm befinden. Wenn dies nicht der Fall ist, folgt das Gerät der schwarzen Linie ohne Probleme weiter.
- Temperatur erkennen :
Mit Hilfe des Thermokameras können wir durch die Verwendung einer spezifischen API, die mit der Thermokamera in Beziehung steht, Matrizen mit Temperaturwerten erzeugen. Nachdem wir die Matrizen extrahiert haben, können wir ableiten, ob die Temperatur größer als 39 Grad Celsius ist. Wenn das der Fall ist, hält der Roboter an, gibt einen Signalton aus und alarmiert den Beifahrer, um zu überprüfen, ob er keine Corona-Symptome hat. Nach 5 Sekunden wird der Roboter wieder Matrizen mit Temperaturwerten erstellen.
- Videostreaming :
Nach einem Klick auf die Funktion in der Tkinter-GUI werden die Videoinformationen an einen Webserver gesendet und gestreamt. Der Grund, warum wir dies getan haben, ist, dass SSH oder VNC Viewer uns nicht erlauben, die Live-Kamera in Echtzeit zu sehen, wenn wir den Raspberry Pi ferngesteuert verwenden wollen. Es wird nur möglich sein, wenn der Raspberry Pi an einen Monitor (mit HDMI) angeschlossen wird. Da wir das Live-Streaming aus der Ferne ermöglichen wollen, haben wir beschlossen, ein Programm zu erstellen, das die Videoinformationen / das Format an einen Webserver sendet, so dass wir es in Echtzeit sehen können.

4 Implementierung

4.1 Setup und Komponententest

In diesem Abschnitt werden wir die Setup-Anweisungen und Implementierungen für jede Hardware, die wir in unserem Projekt verwenden, erläutern.

4.1.1 Raspberry Pi

Es gibt im Wesentlichen 2 Methoden zur Installation des Betriebssystems auf unserem Pi. Wir haben die Methode verwendet, bei der das neueste Raspbian-Image heruntergeladen und auf eine SD-Karte geschrieben wird. Diese Methode erfordert die Installation eines Softwarepakets eines "third-party", das ein bootfähiges Image auf eine SD-Karte schreibt. Der Vorteil dieser Methode ist, dass sie weniger Platz auf der SD-Karte benötigt. Nachdem wir die Software auf die SD-Karte geschrieben haben, können wir sie in den microSD-Kartenslot auf der Rückseite unserer Raspberry PI einstecken, dann sie an einen Bildschirm, eine Tastatur und eine Maus anschließen, um zu überprüfen, ob alles richtig funktioniert.

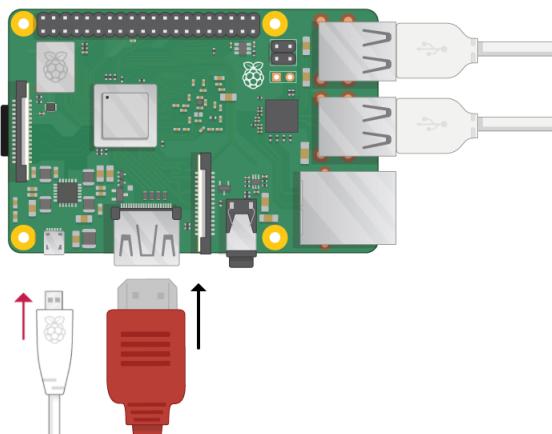


Abbildung 11: Raspberry Pi Verbindungen

Nach ein paar Sekunden erscheint der Desktop des Raspberry Pi OS.

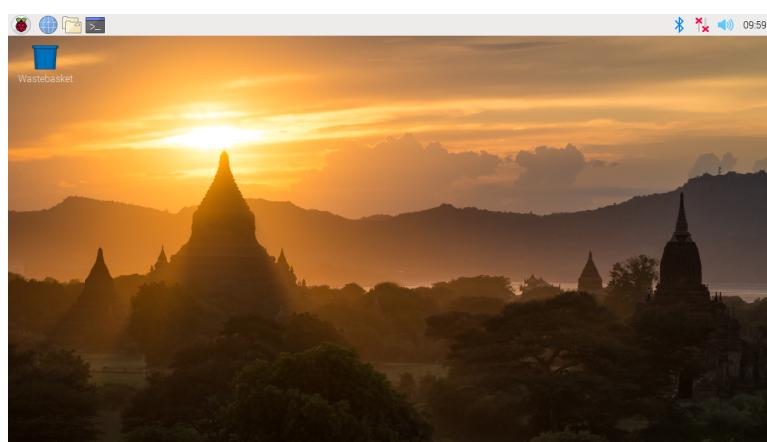


Abbildung 12: Pi Desktop

Dann können wir SSH in der Konfiguration des Raspberry Pis aktivieren und seine IP-Adresse mit dem Befehl ipconfig im Terminal abrufen. Um den Raspberry Pi fernzusteuern, können wir

mit dem Terminal(Unter SSH) oder VNC VIEWER verwenden, aber in unserem Fall benötigen wir VNC Viewer, damit wir den Bildschirm unseres Raspberry Pi beobachten und die grafische Benutzeroberfläche aus der Ferne nutzen können. Allerdings können wir mit dem Terminal keine Bilder extrapolieren und die GUI-Funktionen und Pygame-Fenster nicht verwenden.

4.1.2 Motor Driver Module

Hardwaremäßig :

Der L298N ist ein Zweikanal-H-Brücken-Motortreiber, der ein Paar Gleichstrommotoren ansteuern kann. Das bedeutet, dass er bis zu 2 Motoren einzeln ansteuern kann, was ihn ideal für den Bau von 2-Rad-Roboterplattformen macht. Es gibt jedoch auch die Möglichkeit, 4 Motoren zu steuern, was wir in unserem Fall getan haben. Der Anschluss der Motoren an den L298N war einfach, wir hätten es auch mit 2 Modulen L298N realisieren können, aber für ein besseres Design und Portabilität haben wir es möglich gemacht, 4 Motoren an nur einen Modul-Treiber L298N anzuschließen.

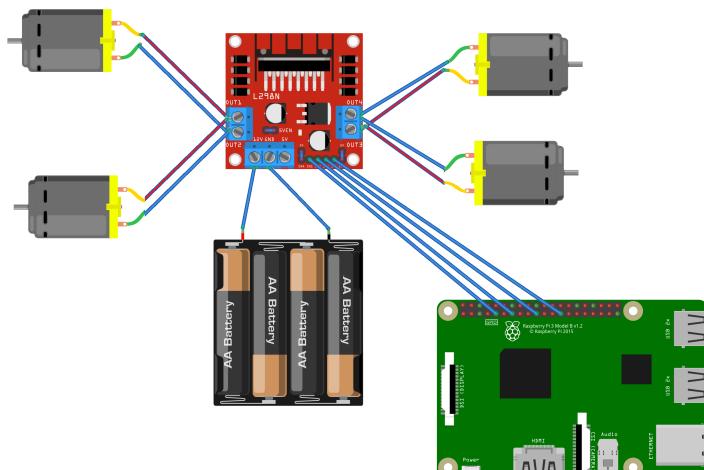


Abbildung 13: Anschluss der L29N Module mit Raspberry Pi

Wie im oberen Bild zu sehen ist, werden 2 Motoren auf der linken Seite an OUT1 /OUT2 und 2 Motoren auf der rechten Seite an OUT3/OUT4 angeschlossen. Wir versorgen die Motoren auch mit 4 AA-Batterien, die eine Spannung zwischen 5 und 35 V haben können. In unserem Fall haben wir 6 V für den Modultreiber L298N zur Verfügung gestellt. Danach haben wir die Schnittstellenpins von dem Modultreiber mit dem Raspberry Pi verbunden, so dass wir die Motoren auf unsere eigene Weise steuern können. Dafür haben wir den Modultreiber mit den folgenden GPIOs verbunden: **27 - 22 - 4 - 17**.

Softwaremäßig :

- Vorwärts : Um vorwärts zu fahren : GPIO.4 und GPIO.27 sollten "True" sein.
GPIO.17 und GPIO.27 sollten "False" sein.
- Rückwärts : Um rückwärts zu fahren : GPIO.4 und GPIO.27 sollten "False" sein.
GPIO.17 und GPIO.27 sollten "True" sein.
- Rechts : Um rechts zu drehen: GPIO.4 und GPIO.22 sollten "True" sein.
GPIO.17 und GPIO.27 sollten "False" sein.

- Links : Um links zu drehen GPIO.17 und GPIO.27 sollten "True" sein.
GPIO.4 und GPIO.22 sollten "False" sein.
- Stop : GPIO.4 GPIO.17 GPIO.27 und GPIO.22 sollten "False" sein.

Hinweis: Wenn GPIO.27 True ist, bewegen sich beide rechten Räder vorwärts. Wenn GPIO.4 True ist, dann bewegen sich die beiden linken Räder vorwärts und umgekehrt.
motorControl.py

```
import RPi.GPIO as GPIO
import time

def init():
    GPIO.setmode(GPIO.BCM)
    GPIO.setup(4,GPIO.OUT)
    GPIO.setup(17,GPIO.OUT)
    GPIO.setup(27,GPIO.OUT)
    GPIO.setup(22,GPIO.OUT)

def forward():
    GPIO.output(4,True)
    GPIO.output(17,False)
    GPIO.output(27,True)
    GPIO.output(22,False)

def reverse():
    GPIO.output(4,False)
    GPIO.output(17,True)
    GPIO.output(27,False)
    GPIO.output(22,True)

def turn_left():
    GPIO.output(4,False)
    GPIO.output(17,True)
    GPIO.output(27,True)
    GPIO.output(22,False)

def turn_right():
    GPIO.output(4,True)
    GPIO.output(17,False)
    GPIO.output(27,False)
    GPIO.output(22,True)

def stop():
    GPIO.output(4,False)
    GPIO.output(17,False)
    GPIO.output(27,False)
    GPIO.output(22,False)
```

Abbildung 14: Python code für Motorsteuerung

Für die Initialisierung haben wir die Option GPIO.BCM verwendet. GPIO.BCM bedeutet, dass wir uns auf die Pins durch die "Broadcom SOC channel" Nummer beziehen, das sind die Nummern nach GPIO in den grünen Rechtecken um die Außenseite der Abbildung 3.

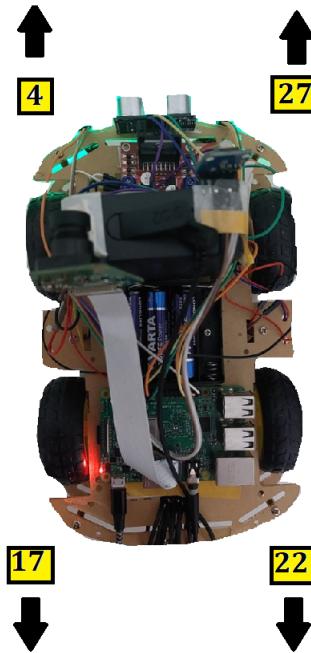


Abbildung 15: Roboter-Richtungen

Anschließend haben wir ein weiteres Python-Programm erstellt, mit dem wir den Roboter über die Tastatur steuern können :

KeypadControlPC.py

```

import pygame
from motorControl import *
from distance_sensor import check
GPIO.setwarnings(False)

def Keyboard():
    GPIO.cleanup()
    init()
    pygame.init()
    screen = pygame.display.set_mode([240, 160])
    while True:
        for event in pygame.event.get():
            if event.type == pygame.KEYDOWN:
                if event.key == pygame.K_RIGHT:
                    print("right")
                    turn_right()
                elif event.key == pygame.K_LEFT:
                    print("left")
                    turn_left()
                elif event.key == pygame.K_UP:
                    print("up")
                    if check():
                        forward()
                elif event.key == pygame.K_DOWN:
                    print("down")
                    reverse()
                elif event.key == pygame.K_x:
                    print("stop")
                    stop()
                elif event.key == pygame.K_q:
                    stop()
                    pygame.quit()
            elif event.type == pygame.KEYUP:
                print('No key pressed')
                stop()

```

Abbildung 16: Python code für Motorsteuerung mit Keypads

Zuerst haben wir pygame importiert, weil es die Möglichkeit bietet, unser Gerät mit der Tastatur zu steuern. Dann haben wir motorControl importiert, um alle Funktionen zu importieren, die wir im Zusammenhang mit den Bewegungen des Roboters erstellt haben, und auch die check()-Funktion des Abstandssensors, damit der Roboter selbst überprüfen kann, ob sich ein Objekt vor ihm befindet oder nicht. Wenn das der Fall ist, hält der Roboter an und kann sich nicht weiterbewegen.

Dann haben wir unsere Keyboard() Funktion definiert, zuerst haben wir alle GPIOs gereinigt, um sicherzustellen, dass alle GPIOs frei sind, dann haben wir unsere Motoren mit der init() Funktion aus motorControl initialisiert. Danach initialisierten wir pygame mit pygame.init(), dies initialisiert alle importierten pygame Module. Wir haben die Höhe und Breite für unser pygame-Fenster auf (240,160) gesetzt, was bedeutet, dass das Verhalten nicht funktioniert, wenn wir auf Tastaturen außerhalb dieses Fensters klicken, und das sorgt für mehr Sicherheit für den Benutzer.

Dann geben wir eine while-Schleife ein und lassen pygame in einer for-Schleife laufen, die prüft, ob ein Keypad innerhalb des pygame-Fensters angeklickt wurde, und wenn das der Fall ist, führen wir eine bestimmte Aktion aus. Wenn wir zum Beispiel auf die rechte Pfeiltastatur klicken, rufen

wir die Funktion `turn_right()` von `motorControl` auf, dasselbe gilt für die linke Pfeiltastatur usw...

Wenn wir jedoch auf die Pfeiltastatur nach oben klicken, muss der Roboter zuerst die `check()` Funktion aufrufen, und wenn alles in Ordnung ist, wird die Funktion `forward()` von `motorControl` aufgerufen. Bei jedem Klick wird ein kleiner Text ausgedruckt, um sicherzustellen, dass unser Programm fehlerfrei funktioniert (z.B. `print("right")` oder `print("left")` und so weiter). Um das Auto anzuhalten, müssen wir `x` drücken, und um die Tastaturfunktion zu verlassen, müssen wir `q` drücken, was das Pygame-Fenster verschwinden lässt. Andernfalls, wenn nichts gedrückt wird, hält das Auto automatisch an. Das Video für die Motorsteuerung mit den Keypads befindet sich im folgenden [Link](#).

Hinweis: Die Verwendung der Tastatur 'x' ist hier ein wenig nutzlos, da das Auto automatisch anhält, wenn wir keine Tastatur anklicken, aber zur besseren Sicherheit wollten wir diese Funktion hinzufügen.

4.1.3 Line follower Sensor

Hardwaremäßig :

Der Sensor erkennt aktiv Infrarotreflexionen, so dass das Reflexionsvermögen und die Form des Ziels der Schlüssel zur Entfernungsbestimmung sind. Darunter, ist die Erkennungsentfernung bei Schwarz am kleinsten und bei Weiß am größten, bei kleinflächigen Objekten ist die Entfernung gering und bei großflächigen groß.

Wir werden also diesen Sensor verwenden, um schwarze Farben zu erkennen und den Roboter einer schwarzen Farbe oder einer schwarzen Linie folgen zu lassen, damit wir den Roboter nicht jedes Mal manuell steuern müssen. Nachdem wir diese IR-Sensoren mit dem Raspberry Pi kompatibel gemacht haben, können wir auswählen, ob der Roboter automatisch einer schwarzen Linie folgen soll oder ob er manuell über eine Tastatur gesteuert werden soll.

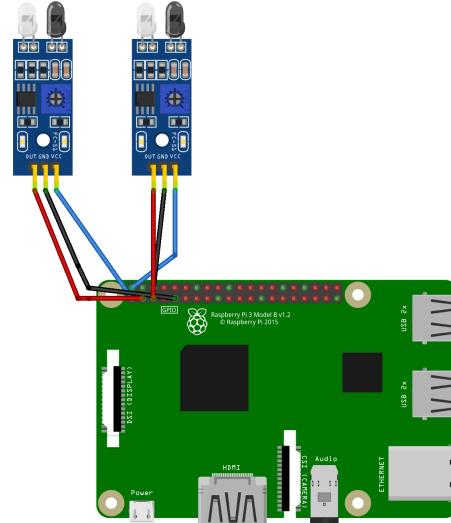


Abbildung 17: Anschluss der IR Sensors mit Raspberry PI

Wie im obigen Bild zu sehen ist, werden wir 2 IR-Sensoren anschließen, einen für die linken Motoren und einen für die rechten Motoren des Robots. Dafür werden wir folgende GPIOs verwenden: GPIO.20 für den ersten IR-Sensor und GPIO.21 für den zweiten IR-Sensor.

Softwaremäßig :

Line_Follower.py

```

from motorControl import init
from distance_sensor import check
import RPi.GPIO as GPIO
import pygame

def follow():
    OK=0
    init()
    GPIO.setup(21,GPIO.IN) #Right
    GPIO.setup(20,GPIO.IN) #Left
    pygame.init()
    screen = pygame.display.set_mode([240, 160])

    try:
        while True:
            if check():
                if GPIO.input(20):
                    GPIO.output(4,True)
                else:
                    GPIO.output(4,False)

                if GPIO.input(21):
                    GPIO.output(27,True)
                else:
                    GPIO.output(27,False)

                for event in pygame.event.get():
                    if event.type == pygame.KEYDOWN:
                        if event.key == pygame.K_q:
                            stop()
                            OK = 1
                            pygame.quit()

                if OK == 1:
                    break

    finally:
        GPIO.cleanup()

```

Abbildung 18: Python code für Linienverfolger

Zuerst importieren wir die init-Funktion aus motorControl.py, weil wir sie für die Initialisierung der Pins benötigen, dann die pygame-Bibliothek, da diese später für unsere Integrationsimplementierung benötigt wird. Außerdem haben wir RPi.GPIO wie üblich importiert, um unsere GPIO-Pins des Raspberry PI zu verwenden.

In der folgenden Funktion haben wir zuerst eine Variable mit dem Namen OK = 0 initialisiert, die init() Funktion von motorControl aufgerufen und GPIO.21 und GPIO.20 als Input initialisiert, da sie ihre Werte von ihren Umgebungen (schwarzes oder weißes Licht) erhalten werden. Dann haben wir pygame mit pygame.init() initialisiert und ein pygame-Fenster mit der Größe von (240,160) erstellt.

Anschließend haben wir eine Schleife in Try-Finally implementiert: Wenn einer der Sensoren den Wert 1 (D.H Schwarz wird anerkannt) hat, werden die Motoren auf der entsprechenden Seite ausgelöst, um sich vorwärts zu bewegen, andernfalls wird er sich nicht bewegen. Dies sollte jedoch unter der Voraussetzung geschehen, dass keine Hindernisse vor ihm liegen, und zwar durch die Funktion check().

Um die Schleife zu verlassen, haben wir ein einfaches Verhalten in Bezug auf das Pygame-Fenster implementiert. Wenn wir auf die Tastatur 'q' klicken, hält das Auto an und der OK-Wert wird auf 1 gesetzt. Sobald OK auf 1 gesetzt ist, bricht die while-Schleife ab und geht direkt zur finally-Zeile und macht eine "cleanup" für alle GPIOs Pins. Das Video für den Linienverfolger befindet sich im folgenden [Link](#).

4.1.4 Abstandssensor

Hardwaremäßig :



Abbildung 19: Ultrasonic Sensor

Model	HC-SR04
Betriebsspannung	5V DC
Entfernung	2 bis 500 cm
Auflösung	0.3 cm
Frequenz	40 kHz
Periode	50 ms

Tabelle 2: Spezifikation der Ultrasonic Sensor

Das Modul hat vier Pins, die mit VCC, Trigger, Echo und Masse bezeichnet sind. VCC wird mit dem 5-Volt Pin des Raspberry Pi verbunden und die Masse mit einem der Masse-Pins. Der "Trigger" des Ultraschalls aktiviert den Sensor und muss mit einem GPIO-Ausgangspin verbunden werden, während das Echo ein Signal liefert, das von einem GPIO-Eingangspin gelesen werden muss.

So funktioniert es: Ein 10 Mikrosekunden langer hoher Impuls wird vom Raspberry Pi an den Sensor gesendet. Dies signalisiert dem Sensor, einen 8-Zyklen-Schallimpuls mit 40 Kihertz zu erzeugen, wenn der Sensor das Echo empfängt. Wenn der Sensor ein Echo empfängt, kann er die Zeit zwischen der Erzeugung des Stoßes und dem Empfang des Echos bestimmen und dies an den PI zurückmelden. Er sendet einen hohen Impuls für eine Länge, die der Dauer des Echos entspricht. Die Länge des Impulses ist also proportional dazu, wie weit das Objekt entfernt ist - wir müssen nur auf den Echo-Pin hören und die Dauer des hohen Impulses messen.

Wir wollen die Entfernung zwischen dem Sensor und einem Objekt berechnen können. Die Impulse legen eine Strecke **D** bis zum Objekt zurück und dann eine weitere Strecke **D** zurück zum Sensor. Wenn wir also die Gleichung für

$$\text{Geschwindigkeit} = \text{Entfernung}/\text{Zeit}$$

verwenden, ist die Distanz **2 D**, weil es hin und her geht. Außerdem beträgt die Schallgeschwindigkeit etwa **340 m/s**, woraus sich folgendes ergibt:

$$340m/s = 2D/\text{Zeit}$$

$$D = 170m/s * \text{Zeit}$$

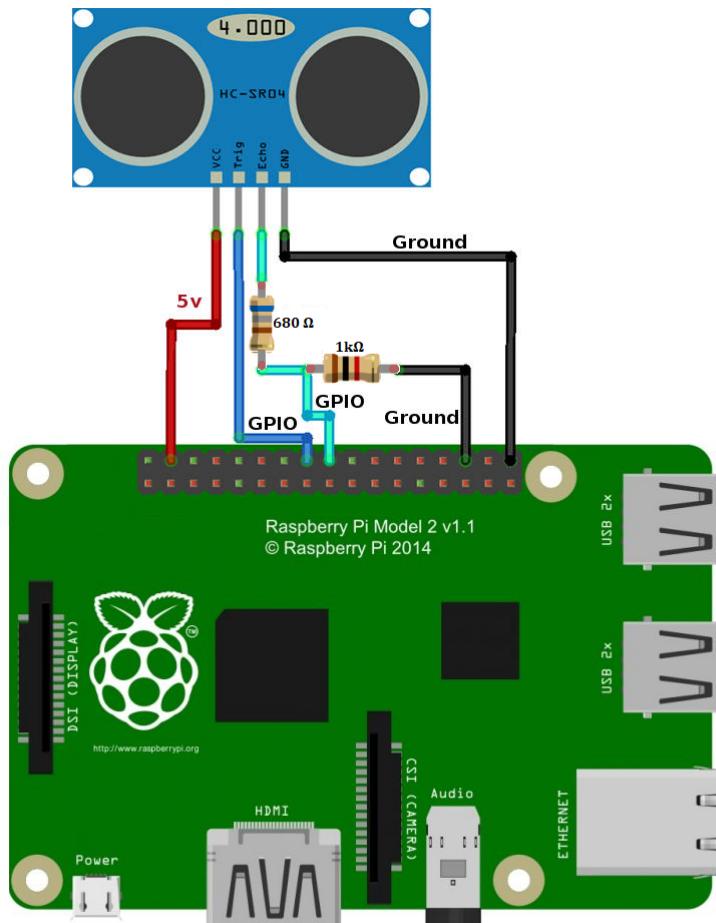


Abbildung 20: Anschluss des Ultrasonic-Sensors mit Raspberry Pi

Die Verdrahtung ist ziemlich einfach, aber wir müssen vorsichtig sein wie wir den Echo-Pin einrichten, da er ein High bei 5 Volt ausgibt ein High bei 5 Volt ausgibt. Um dies zu bewältigen, wird der Echo-Pin mit einem Spannungsteiler verbunden, wir haben einen 1 Kilo-Ohm-Widerstand für R₂ gewählt, um einen gewissen Schutz zwischen dem GPIO-Pin und Raspberry Pi. Wenn ich den GPIO-Pin versehentlich als Ausgang einstelle, würde ein Strom vom Pin zur Masse fließen 1kOhm ist genug Widerstand, um eine Beschädigung des Pins zu verhindern, wenn ein solcher Fehler auftritt. Mit unseren beiden gewählten Werten müssen wir R₁ mit Hilfe der Spannungsteiler Gleichung berechnen.

$$U_{aus} = U_{ein} * R_2 / (R_1 + R_2)$$

Wir wollen 3,3 Volt als Ausgang für eine bessere Sicherheit, also lösen wir die Gleichung :

$$3.3 = 5 * 1000 / (R_1 + 1000)$$

$$R_1 = 515\text{ Ohm}$$

Wir brauchen einen Widerstand, der mindestens so groß ist. Der nächstgrößere Widerstand, den wir hatten, war **680** Ohm, was **2,98** Volt ergibt, was leicht eine ausreichend hohe genug ist, um vom GPIO-Pin als hoch erkannt zu werden.

Softwaremäßig :

distance_sensor.py

```

import RPi.GPIO as GPIO
from motorControl import init
import time

def distance_Ultrasonic():
    GPIO.setmode(GPIO.BCM)
    TRIG = 18
    ECHO = 24

    GPIO.setup(TRIG, GPIO.OUT)
    GPIO.output(TRIG, 0)

    GPIO.setup(ECHO, GPIO.IN)

    time.sleep(0.1)

    print("Starting Measurement...")

    GPIO.output(TRIG, 1)
    time.sleep(0.0001)
    GPIO.output(TRIG, 0)

    while GPIO.input(ECHO) == 0:
        pass
    start = time.time()

    while GPIO.input(ECHO) == 1:
        pass
    stop = time.time()

    distance_cm = (stop - start) * 17000
    GPIO.cleanup()

    return distance_cm

def check():
    init()
    if distance_Ultrasonic() < 1:
        stop()
        return False
    return True

```

Abbildung 21: Python code für den Abstandssensor

Für dieses Programm importieren wir nicht nur das GPIO-Modul und die init-Funktion von motorControl, sondern auch das Zeitmodul, da wir messen müssen, wie lange der Eingangspin hoch ist. Nachdem wir angegeben haben, dass wir die Pin-Nummerierung verwenden werden, sollten wir 2 Konstanten für die Pins erstellen, die mit Trigger und Echo verbunden sind, um den Code ein wenig lesbarer zu machen. In unserem Fall haben wir dem TRIGGER Pin GPIO.18 und dem ECHO Pin GPIO.24 zugewiesen.

Dann werden die Pins so eingestellt, dass der Trigger-Pin auf Low gesetzt wird. Der Eingangs-Pin braucht keinen Pulldown-Widerstand, da der Sensor mit Masse verbunden ist und diesen auf Low setzt, wenn er nicht benutzt wird. Wir benötigen nun eine kurze Verzögerung, damit sich der Sensor einpendeln kann. Die Spezifikation sieht eine Zykluszeit von 50 Millisekunden vor, also sollten wir mit 100 Millisekunden auskommen. Jetzt sind wir bereit, eine Messung vorzunehmen. Um zu wissen, wann die Messung beginnt, haben wir eine Print-Anweisung an den Benutzer eingefügt, um anzugeben, dass die Messung läuft.

Wir müssen den Trigger auf "high" setzen, 10 Mikrosekunden warten, wie zuvor angegeben, und dann den Trigger auf "low" setzen, wobei der Sensor ausgelöst wird. Da dies unglaublich schnell geschieht, müssen wir den Code so effizient wie möglich gestalten. time.time drückt die aktuelle Zeit in Sekunden aus.

Der Code funktioniert folgendermaßen: Die while-Schleife wird so lange fortgesetzt, bis der Eingangspin High wird. Die nächste while-Schleife wird fortgesetzt, bis der Pin auf Low geht.

Sobald er auf Low geht, ist das Signal beendet und die Stoppzeit wird notiert. Innerhalb der while-Schleife sagt der pass-Befehl einfach, dass nichts getan werden soll, was bedeutet, dass das Programm fast die gesamte Zeit damit verbringt, den Status der Pins zu beobachten. Da wir nun die Start- und Stoppzeit kennen, können wir daraus schließen, dass die Differenz zwischen ihnen die Länge des Impulses ist. Diese wird mit 170m multipliziert, wie in der zuvor besprochenen Gleichung. Da wir dies in cm wollen, multiplizieren wir **170 mit 100** und geben dann `distance_cm` zurück.

Wir haben auch eine andere Funktion namens `check()` erstellt, die prüft, ob `distance_cm` kleiner als 1 ist. Wenn das der Fall ist, dann hält das Auto an und gibt `false` zurück, andernfalls gibt es `True` zurück. Um die Funktionalität unseres Abstandssensors testen zu können, haben wir folgendes Programm erstellt. `Ultrasonic_test.py`

```
import sys
sys.path.append('..')
from distance_sensor import *

init()

while True:
    forward()
    if check() == False:
        break

stop()
```

Abbildung 22: Python code für den Test des Abstandssensors

Wir haben also zunächst unsere GPIO-Pins der Motoren initialisiert und eine Schleife laufen lassen. In dieser while-Schleife fährt das Auto vorwärts und prüft gleichzeitig, ob sich ein Objekt vor ihm befindet. Wenn das der Fall ist, wird die Schleife unterbrochen und das Auto hält an. Das Video für den Abstandssensor mit dem Roboter befindet sich im folgenden [Link](#).

4.1.5 Audiogerät

Hardwaremäßig :

Um Töne in unserem Roboter zu erzeugen, brauchen wir ein kleines Audiogerät, das nicht viel wiegt und genug Platz zum Einsetzen hat. Wir haben uns für ein JBL-Gerät entschieden, das nur 0,18 kg wiegt und 8,6 x 7,1 x 3,2 cm misst. Die Verkabelung ist ganz einfach: Wir haben das Gerät mit dem AUX IN-Kabel zu Raspberry Pi angeschlossen.

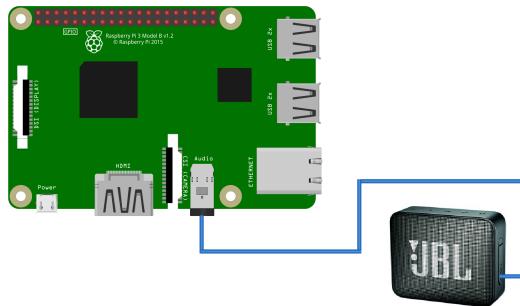


Abbildung 23: Anschluss des Audiogeräts mit Raspberry Pi

Also haben wir einige Audioaufnahmen gemacht und sie in den Raspberry PI hochgeladen. Um diese Audios abzuspielen, haben wir Omxplayer verwendet.

Omxplayer ist eigentlich ein Kommandozeilen-Videoplayer, der speziell für die GPU des Raspberry Pi entwickelt wurde. Er verwendet die OpenMAX-Hardwarebeschleunigungs-API, Broadcoms offiziell unterstützte VideoCore-API für GPU-Video-/Audioverarbeitung. Kurz gesagt, er hat vollen Zugriff auf die GPU des Raspberry Pi, so dass er im Grunde jeden existierenden Audio- und Video-Codec unterstützt.[\[5\]](#)

Das Audiosignal wird standardmäßig an die Kopfhörerbuchse gesendet. Die Befehlssyntax ist so einfach wie unten:

```
omxplayer /pfad/zu/audio/datei.wav
```

Softwaremäßig :

audioPlay.py

```
import subprocess
import pygame
import os

def play():
    os.chdir("/home/pi/Desktop/Corona_Robot/audio")
    subprocess.call('omxplayer stop.wav', shell=True)
    time.sleep(1)
    subprocess.call('omxplayer ausweis.wav', shell=True)
    time.sleep(3)
    subprocess.call('omxplayer temp_hoch.wav', shell=True)
    os.chdir("/home/pi/Desktop/Corona_Robot")
```

Abbildung 24: Python code für das audioPlay Programm

Zuerst haben wir subprocess importiert, um Linux-Befehle in Python zu verwenden, dann time library für einige Verzögerungsimplementierungen und os, um den Verzeichnispfad zu ändern.

Der Grund, warum wir den Verzeichnispfad geändert haben, ist, dass wir unser Integrationsprogramm in einem anderen Pfad ausführen werden und um die Audios aus diesem Ordner abzuspielen, müssen wir den Pfad angeben, in dem sich die Audios befinden.

Zuerst riefen wir "stop.wav" auf, welches ein Audio ist, das wir aufgenommen haben und das dem Fahrgäst sagt, dass er anhalten soll.

Der Roboter wartet 1 Sekunde lang und ruft dann den nächsten Unterprozess auf (ausweis.wav). Dieser wav-Player fragt den Fahrgäst nach seinem Ausweis und teilt ihm dann mit, dass die Temperatur zu hoch ist und er so bald wie möglich einen Test gegen Covid machen muss.

Am Ende haben wir den Verzeichnispfad zurück zu dem Ort geändert, an dem sich das Integrationsprogramm befindet.

4.1.6 Wärmebildkamera

Hardwaremäßig :

Der AMG8833 ist ein 64-Pixel-Temperatursensor, der von Panasonic im Rahmen seiner Grid-EYE®-Produktlinie entwickelt wurde. Der Sensor enthält ein 8x8-Array von Infrarot-Thermosäulen, die die Temperatur durch Messung der von emittierenden Körpern abgegebenen Infrarotstrahlung annähernd bestimmen.

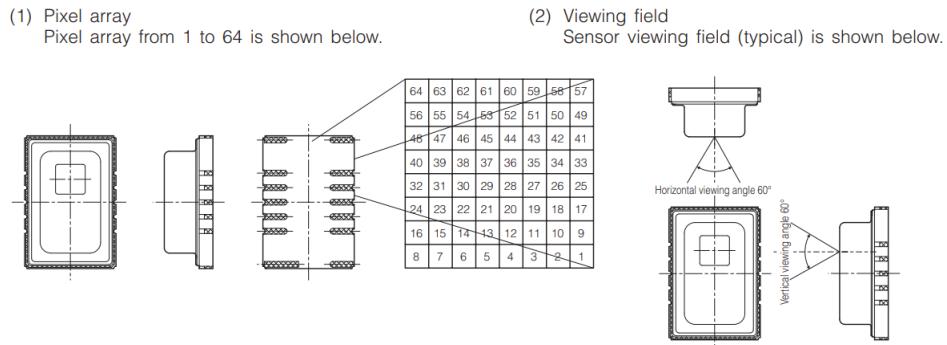


Abbildung 25: Pixelarray und Sichtfeld

Die Grid-EYE kommuniziert über den I2C-Bus, wodurch sie auch sofort mit Raspberry Pi kompatibel ist. Der AMG8833 enthält eine integrierte Linse, die den Betrachtungswinkel des Sensors auf 60 Grad begrenzt, was zu einem Erfassungsbereich führt, der für Objekte in der Mitte des Feldes nützlich ist (im Gegensatz zu Fern- oder Nahfeld). Der Sensor arbeitet mit 3,3 V und 5 V, einer Abtastrate von 1 Hz bis 10 Hz und einer ungefähren Temperaturauflösung von 0,25 °C über einen Bereich von 0 °C bis 80 °C.[7]

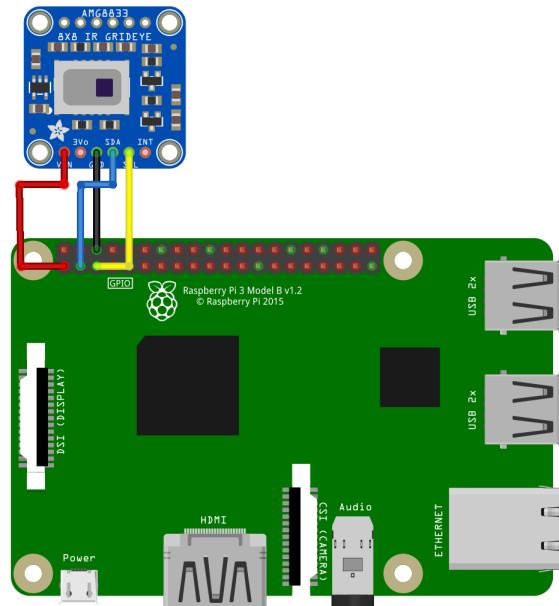


Abbildung 26: Anschluss des AMG88-Kameras zu Raspberry Pi

Der AMG8833 kommuniziert mit dem Raspberry Pi über das I2C-Protokoll. Um Daten über I2C lesen und schreiben zu können, müssen wir zunächst die I2C-Ports auf dem RPi aktivieren. Dies geschieht entweder über die Kommandozeile oder über die Einstellungen → Raspberry Pi Konfiguration. Nachdem wir unsere Wärmebildkamera mit dem Raspberry Pi verbunden haben, müssen wir überprüfen, ob sie richtig angeschlossen ist. Um dies zu überprüfen, können wir den folgenden Befehl verwenden : `sudo i2cdetect -y 1`

Laut [Adafruit](#) ist die Adresse des AMG833s 0x69. Also sollte 0x69 im Terminal angezeigt werden, wenn dieser Befehl ausgeführt wird, sonst wird unsere Wärmebildkamera nicht funktionieren.

```
pi@raspberrypi:~ $ i2cdetect -y 1
  0  1  2  3  4  5  6  7  8  9  a  b  c  d  e  f
00: --- --- --- --- --- --- --- --- --- --- --- -
10: --- --- --- --- --- --- --- --- --- --- --- -
20: --- --- --- --- --- --- --- --- --- --- --- -
30: --- --- --- --- --- --- --- --- --- --- --- -
40: --- --- --- --- --- --- --- --- --- --- --- -
50: --- --- --- --- --- --- --- --- --- --- --- -
60: --- --- --- --- --- --- --- 69 --- --- --- -
70: --- --- --- --- --- --- --- --- --- --- --- -
pi@raspberrypi:~ $
```

Abbildung 27: Pixelarray und Sichtfeld

Softwaremäßig :

```
import time
import busio
import board
import adafruit_amg88xx
import signal
import os
from motorControl import stop,init
from audio.audioPlay import play
```

Abbildung 28: Python code für audio/amg88temp.py 1

Wir importierten time Bibiliothek für Verzögerungen Implementierungen, dann importierten wir adafruit Bibiliothek für amg88xx Thermokamera. Die Art und Weise, ein I2C-Objekt zu erstellen, hängt von der verwendeten Karte ab. Für Boards mit beschrifteten SCL- und SDA-Pins importieren wir board. Dann haben wir die Signal- und Os-Bibliotheke importiert, um benutzerdefinierte Signale in unserem Code zu implementieren. Anschließend importieren wir stop und init aus motorControl und play aus audio.audioplay, um die aufgenommenen Töne zu erzeugen

```
SIG = 0

def receive(signum, stack):
    global SIG
    SIG = 1

signal.signal(signal.SIGUSR1, receive)
```

Abbildung 29: Python code für audio/amg88temp.py 2

Zuerst haben wir eine Variable namens SIG auf 0 initialisiert und eine Empfangsfunktion mit 2 Parametern SIGNUM und stack definiert. In dieser Funktion haben wir SIG global gemacht, so dass es außerhalb der Funktion geändert werden kann, und danach setzen wir SIG auf 1. Diese

Funktion wird nur aufgerufen, wenn das Programm ein Sigkill-Signal von USR1 empfängt.

```

19 def temp_audio():
20     global SIG
21     pid = os.getpid()
22     with open("/tmp/progl.pid", "w") as pidfile:
23         pidfile.write(f"{pid}\n")
24     i2c = busio.I2C(board.SCL, board.SDA)
25     amg = adafruit_amg88xx.AMG88XX(i2c)
26     init()
27     while True:
28         OK = 0
29         print("", end = '')
30         for row in amg.pixels:
31             for temp in row:
32                 if temp >= 39 and SIG == 1:
33                     stop()
34                     print("Face detected and temperature above 39")
35                     print("Playing audio...")
36                     SIG = 0
37                     play()
38                     time.sleep(5)
39                     OK = 1
40                     break
41                 else:
42                     print(temp, ', ', end = '')
43                     continue
44             if OK == 1:
45                 break
46             print("")
47             print("}")
48             print("\n")
49             time.sleep(1)

```

Abbildung 30: Python code für audio/amg88temp.py 3

Anschließend haben wir die Funktion `temp_audio` erstellt. Wir setzten `SIG` wie üblich auf `global`, damit wir den `SIG`-Wert von außen extrahieren können.

Dann haben wir die Prozess-ID des ausgeführten Programms in der Variable `pid` gespeichert. Leider können wir diesen Wert nicht abrufen und ihn mit import x/y an ein anderes Programm weitergeben. Wir sollten also die PID dieses Programms irgendwo im Dateisystem speichern. Wir haben also uns dafür entschieden, sie in der temporären Datei zu speichern. Zeile 24 dient der Initialisierung des I2C-Busses. Sobald wir das I2C-Interface-Objekt erstellt haben, können wir es verwenden, um das AMG88xx-Objekt zu instanziieren (Zeile 25).

Danach haben wir unsere GPIOs mit der Funktion `init()` initialisiert. Dann sind wir in eine while-Schleife eingestiegen: Wir setzen eine Variable namens `OK` auf 0 und geben dann ein Druckformat ein, um unsere Temperaturwerte in einer Matrix zu speichern.

Die Temperaturwerte werden in einer zweidimensionalen Liste gespeichert, wobei der erste Index die Zeile und der zweite die Spalte ist. Die Pixelfunktion des `AMG88_xx` speichert die Temperaturwerte in einer Matrix und gibt sie zurück.

Auf unserer Seite des Codes werden wir also diese Matrix aus `AMG88.pixels` extrahieren und prüfen, ob einer der Werte über 39 liegt, aber unter der Bedingung, dass die Temperaturen gemessen werden, wenn die normale Kamera ein Gesicht erkennt, und da wir die normale Kamera direkt hinter der Wärmebildkamera anschließen, wird das Verhalten korrekt funktionieren.

Kurz gesagt, wenn `SIG` gleich 1 ist, bedeutet das, dass ein Gesicht erkannt wird. Wir werden im nächsten Abschnitt (Kamera) genauer erklären, wie die Gesichtserkennung implementiert wird.

Wenn also ein Gesicht erkannt wird und die Temperatur über oder gleich 39 Grad ist, halten wir

das Auto an und drucken im Terminal einige Texte aus, setzen SIG zurück auf 0 und spielen die aufgenommenen Audios ab, dann verzögern wir um 5 Sekunden, um den Fahrgast gehen zu lassen und weiterzufahren. Der Wert OK wurde hier verwendet, um die zweite for-Schleife zu verlassen und zur besseren Lesbarkeit beim Debuggen zur ersten Schleife zurückzukehren. Alle anderen Zeilen wurden verwendet, um am Ende eine ordentliche Matrix zu erstellen.

So wird unsere Matrix mit den Temperaturwerten aussehen :

```
{19.5 ,19.75 ,20.0 ,19.5 ,19.75 ,20.0 ,20.25 ,20.5 ,
19.25 ,19.75 ,19.5 ,19.75 ,20.0 ,20.0 ,20.0 ,19.75 ,
19.5 ,19.75 ,19.25 ,19.25 ,19.5 ,20.0 ,19.75 ,19.0 ,
19.25 ,19.25 ,19.75 ,19.0 ,19.0 ,19.75 ,19.5 ,20.25 ,
18.75 ,19.5 ,19.25 ,19.0 ,19.5 ,19.75 ,20.0 ,19.75 ,
19.25 ,19.75 ,19.25 ,19.25 ,19.25 ,19.75 ,19.5 ,20.75 ,
18.75 ,19.0 ,18.75 ,19.75 ,19.5 ,19.25 ,20.25 ,20.75 ,
18.75 ,19.5 ,19.5 ,19.5 ,19.5 ,20.0 ,20.75 ,20.25 ,
}
```

Abbildung 31: 8x8 Matrize vom AMG88xx

4.1.7 Kamera

Hardwaremäßig :



Abbildung 32: Innomaker Kamera

Marke	Innomaker
Abmessungen	12,7 x 8,89 x 4,57 cm
Hersteller-Referenz	CAM-MIPIOV9281
Konnektivität Typ	Infrarot
Betriebssystem	Linux
Gewicht	45g

Tabelle 3: Spezifikation der Innomaker Kamera

Die Kamera hat einen 5-MPixel-Sensor und wird über ein Flachbandkabel mit dem CSI-Anschluss des Raspberry Pi verbunden. Die Video- und Standbildqualität ist besser als bei einer USB-Webcam ähnlichen Preises.

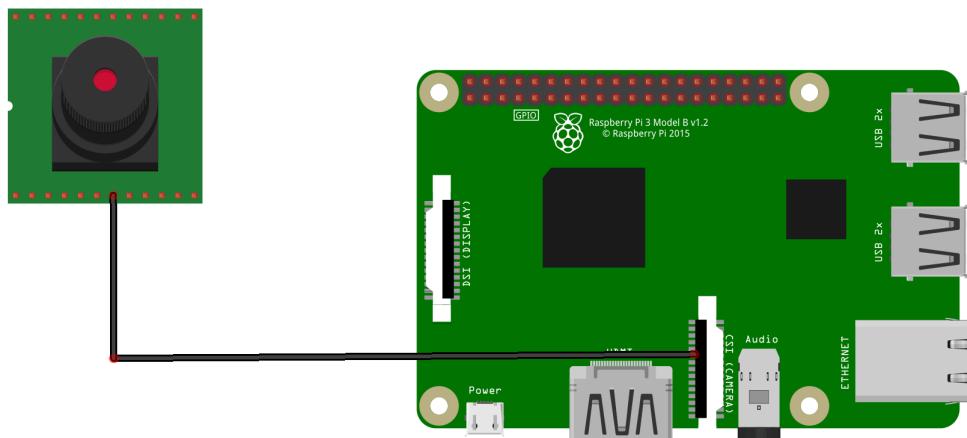


Abbildung 33: Anschluss der Innomaker Kameras mit Raspberry Pi

Um unsere Kamera in Raspbian zu aktivieren, führen wir `sudo raspi-config` im Terminal aus und können dann die Kamera dort freischalten.

Der Grund, warum wir eine Kamera verwenden, ist, dass wir eine bessere Erfahrung für den Benutzer gewährleisten wollen. Wenn er den Roboter fernsteuert, kann er mit der Kamera überprüfen, ob er sich auf dem richtigen Weg befindet oder nicht.

Mit dieser Kamera haben wir auch eine Videostreaming-Funktion implementiert. Da wir das Auto ferngesteuert verwenden werden, müssen wir es live von einer Website beobachten. Warum nicht direkt vom Raspberry Pi VNC Viewer oder SSH ? Weil es nicht möglich ist, das Verhalten der Kamera von SSH oder VNC Viewer zu sehen. Dies funktioniert nur, wenn wir Raspberry PI mit HDMI an einen Monitor anschließen.

Softwaremäßig :

Für die Implementierung des Computer-Vision-Teils werden wir das OpenCV-Modul in Python verwenden, und für die Anzeige des Live-Streams im Webbrowser werden wir das Web-Framework Flask einsetzen. Bevor wir uns mit der Programmierung befassen, erklären wir mal was Flask und OpenCV sind.

Laut Wikipedia ist Flask ein in Python geschriebenes Mikro-Webframework. Es wird als Mikroframework eingestuft, weil es keine besonderen Tools oder Bibliotheken benötigt. Es hat keine Datenbankabstraktionsschicht, Formularvalidierung oder andere Komponenten, bei denen bereits vorhandene Bibliotheken von Drittanbietern gemeinsame Funktionen bereitstellen.

Laut GeeksForGeeks ist OpenCV die große Open-Source-Bibliothek für Computer Vision, maschinelles Lernen und Bildverarbeitung und spielt jetzt eine wichtige Rolle im Echtzeitbetrieb, der in den heutigen Systemen sehr wichtig ist.

Die Struktur der Kamera-Streaming-Implementierung ist etwas komplexer als die anderen Programme, die wir entwickelt haben. Wir haben folgende Struktur für unser Programm verwendet:

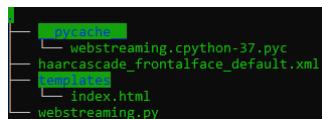


Abbildung 34: Struktur der Videostreaming-Funktion

Wir haben einen Ordner mit dem Namen templates erstellt, in den wir unsere index.html für unsere Flask-App schreiben werden, dann webstreaming.py und haarcascade.xml außerhalb des templates-Ordners.

__pycache__ ist aber ein Verzeichnis, das Bytecode-Cache-Dateien enthält, die automatisch von Python erzeugt werden, nämlich kompilierte Python-Dateien oder . pyc-Dateien.

```

1  from flask import Flask, render_template, Response
2  import cv2
3  from imutils.video import VideoStream
4  import imutils
5  import signal
6  import os

```

Abbildung 35: Python Code für Videostreaming Implementierung 1

Nach der Installation von Opencv und Flask haben wir diese in unser Programm importiert. In Zeile 1 werden die benötigten Flask-Pakete importiert - wir werden diese Pakete verwenden, um unsere index.html-Vorlage zu rendern und sie an die Clients weiterzuleiten. Zeile 2 übernimmt den Import unserer opencv-Bibliothek. Die Klasse VideoStream (Zeile 3) ermöglicht uns den Zugriff auf unser Raspberry Pi-Kameramodul. Zeile 5 und 6 importieren die Signal- und Os-Bibliothek, die wir verwenden werden, um Signale an andere Programme für andere Aktionen zu senden. Wir werden dies ausführlich erklären, wenn wir unsere Funktionen zeigen.

```

10 app = Flask(__name__)
11
12 # Load the cascade
13 face_cascade = cv2.CascadeClassifier(cv2.data.haarcascades + 'haarcascade_frontalface_default.xml')
14
15 vs = VideoStream(usePiCamera=1).start()

```

Abbildung 36: Python Code für Videostreaming Implementierung 2

In Zeile 10 initialisieren wir unsere Flask-App selbst. In Zeile 13 laden wir einen Haar-Kaskaden-Klassifikator in eine Variable namens `face_cascade`; Gesichtserkennung mit Haar-Kaskaden ist ein auf maschinellem Lernen basierender Ansatz, bei dem eine Kaskadenfunktion mit einem Satz von Eingabedaten trainiert wird. OpenCV enthält bereits viele vorgebildete Klassifikatoren für Gesichter, Augen, Lächeln, usw.. In unserem Projekt werden wir also den Gesichtsklassifikator verwenden, den wir von [Github Repository von OpenCV](#) erhalten haben. Mit Zeile 15 können wir auf unseren Videostream zugreifen, da wir das Rpi Modul Camera verwenden.

```

17 def gen_frames(): # generate frame by frame from camera
18     while True:
19         # Capture frame-by-frame
20         global vs
21         global faces
22         OK = 0
23
24         frame = vs.read() # read the camera frame
25         frame = imutils.resize(frame, width=500)
26         gray = cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY)
27
28         if os.path.exists('/tmp/prog1.pid'):
29             OK = 1
30             with open("/tmp/prog1.pid", "r") as pidfile:
31                 temp_pid = int(pidfile.readline())
32
33         # Detect the faces
34         faces = face_cascade.detectMultiScale(gray, 1.5, 4)
35         length = len(faces)
36
37         if length != 0 and OK == 1:
38             print(temp_pid)
39             print("killing")
40             os.kill(temp_pid, signal.SIGUSR1)
41
42         # Draw the rectangle around each face
43         for (x, y, w, h) in faces:
44             cv2.rectangle(frame, (x, y), (x+w, y+h), (255, 0, 0), 2)
45
46         ret, buffer = cv2.imencode('.jpg', frame)
47         frame = buffer.tobytes()
48         # concat frame one by one and show result
49         yield (b'--frame\r\n'b'Content-Type: image/jpeg\r\n\r\n' + frame + b'\r\n')

```

Abbildung 37: Python Code für Videostreaming Implementierung 3

Dann haben wir eine Funktion namens `gen_frames()` definiert. Die Funktion tritt in einer Schleife ein, in der sie kontinuierlich Bilder von der Kamera als Antwortpakete zurückgibt. Die Funktion fordert die Kamera auf, ein Bild zu liefern, und gibt dieses Bild als Antwortpaket mit dem Inhaltstyp `image/jpeg` zurück, wie oben gezeigt.

In Zeile 25 haben wir die Breite des Videostreams, der an den Webserver gesendet werden soll, etwas angepasst. Dann haben wir die Bilder in "Grayscale" umgewandelt. Der Grund, warum wir hier "Grayscale" verwenden, ist, dass die Gesichtserkennung nur bei Grayscalebildern funktioniert, daher ist es wichtig, das Farbbild der Frames in Grayscale zu konvertieren.

Dann haben wir eine if-Anweisung mit der Bedingung geschrieben, dass wir die `prog1.pid` aus dem Wärmekameraprogramm bereits geschrieben haben. Jetzt muss das Kameraprogramm diese

PID extrahieren. Wir prüfen zuerst, ob der Pfad existiert, wenn das der Fall ist, setzen wir Ok auf 1 und öffnen die Datei, lesen den Wert der PID und speichern ihn in einem temp-pid-Wert.

Dann erkennen wir die Gesichter mit detectMultiScale. Es nimmt 3 Argumente entgegen - das Eingabebild, scaleFactor und minNeighbours. scaleFactor gibt an, um wie viel die Bildgröße bei jeder Skalierung reduziert wird. minNeighbours gibt an, wie viele Nachbarn jedes Kandidatenrechteck haben sollte, um es zu behalten.

faces enthält eine Liste von Koordinaten für die rechteckigen Regionen, in denen Flächen gefunden wurden. Wir verwenden diese Koordinaten, um die Rechtecke in unserem Bild zu zeichnen.

Wenn die Länge von faces ungleich 0 und OK = 1 ist (was bedeutet, dass prog1.pid im temporären Dateisystem gefunden wurde), führen wir zur besseren Debugging-Erfahrung einige print Anweisungen durch und beenden dann das SIGUSR1-Signal, das wir in dem anderen laufenden Programm definiert haben (bezogen auf die Wärmebildkamera, die wir zuvor erklärt haben)

Kurz gesagt, wenn ein Gesicht erkannt wird und die PID des anderen Programms gefunden wird, dann versuchen wir das SIGUSR1-Signal zu killen. Das andere Programm empfängt dieses Signal, und wenn die Temperatur über 39 Grad liegt und ein Gesicht erkannt wird (das Signal wird gesendet), dann spielen wir den Ton ab und so weiter.

In den Zeilen 42 und 43 wird nur das Rechteck um jedes Gesicht gezeichnet. Die Funktion cv2.imencode (Zeile 46) wird verwendet, um das Bildformat in Streaming-Daten zu konvertieren bzw. zu kodieren und sie dem Speicher-Cache zuzuweisen. Sie wird hauptsächlich zur Komprimierung des Bilddatenformats verwendet, um die Netzwerkuübertragung zu erleichtern.

Mit dem Schlüsselwort yield wird die Exektion fortgesetzt, und es werden so lange Bilder erzeugt, bis sie nicht mehr benötigt werden.

```
52 @app.route('/video_feed')
53 def video_feed():
54     #Video streaming route.
55     return Response(gen_frames(), mimetype='multipart/x-mixed-replace; boundary=frame')
```

Abbildung 38: Python Code für Videostreaming Implementierung 4

Die Route "/video_feed" gibt die Streaming-Antwort zurück. Da dieser Stream die Bilder liefert, die auf der Webseite angezeigt werden sollen, befindet sich die URL zu dieser Route im srcAttribut des Bild-Tags (siehe index.html unten). Der Browser aktualisiert das Bildelement automatisch, indem er den Strom von JPEG-Bildern darin anzeigt, da mehrteilige Antworten von den meisten/allen Browsern unterstützt werden.

```
1 <html>
2   <head>
3     <title>Pi Video Surveillance</title>
4   </head>
5   <body>
6     <h1>Pi Video Surveillance</h1>
7     
8   </body>
9 </html>
```

Abbildung 39: Html Code für den Flask-Webbrowser

Wie wir in webstreaming.py gesehen haben, rendern wir eine HTML-Vorlage namens index.html. Die Vorlage selbst wird vom Flask-Webframework ausgefüllt und dann an den Webbrowser übermittelt.

Der Webbrowser übernimmt dann das generierte HTML und rendert es auf unserem Bildschirm. Wie wir sehen können, ist dies eine sehr einfache Webseite, allerdings. In Zeile 7 weisen wir Flask an, die URL unserer `video_feed`-Route dynamisch zu rendern.

Da die Funktion `video_feed` für die Bereitstellung von Bildern von unserer Webcam zuständig ist, wird die `src` des Bildes automatisch mit unseren Ausgabebildern gefüllt.

Unser Webbrowser ist dann intelligent genug, um die Webseite richtig zu rendern und den Live-Videostream zu liefern.

```
58     @app.route('/')
59     def index():
60         """Video streaming home page."""
61         return render_template('index.html')
```

Abbildung 40: Python Code für Videostreaming Implementierung 5

Die nächste Funktion, `index`, rendert unsere `index.html`-Vorlage und stellt den ausgegebenen Videostream bereit: Diese Funktion ist recht simpel - alles, was sie tut, ist der Aufruf von Flask `render_template` für unsere HTML-Datei.

```
64     def videostreaming():
65         # start the flask app
66         app.run(host="192.168.0.44", port=8000, debug=True, use_reloader=False)
67 
```

Abbildung 41: Python Code für Videostreaming Implementierung 6

Um unsere Flask-App zu starten, rufen wir die Funktion `app.run()` auf. Als Parameter fügen wir die Host-Ip des Raspberry Pis und den Port 8000 hinzu, so dass wir das Videostreaming in einem Webbrower von jedem Gerät starten können, das mit demselben Netzwerk verbunden ist. Das Video für die Videostreaming mit der Thermokamera und Gesichtserkennung Feature befindet sich im folgenden [Link](#).

4.2 Integration

Nachdem wir alle Anschlüsse miteinander verbunden haben, ergibt sich die folgende Schaltung:

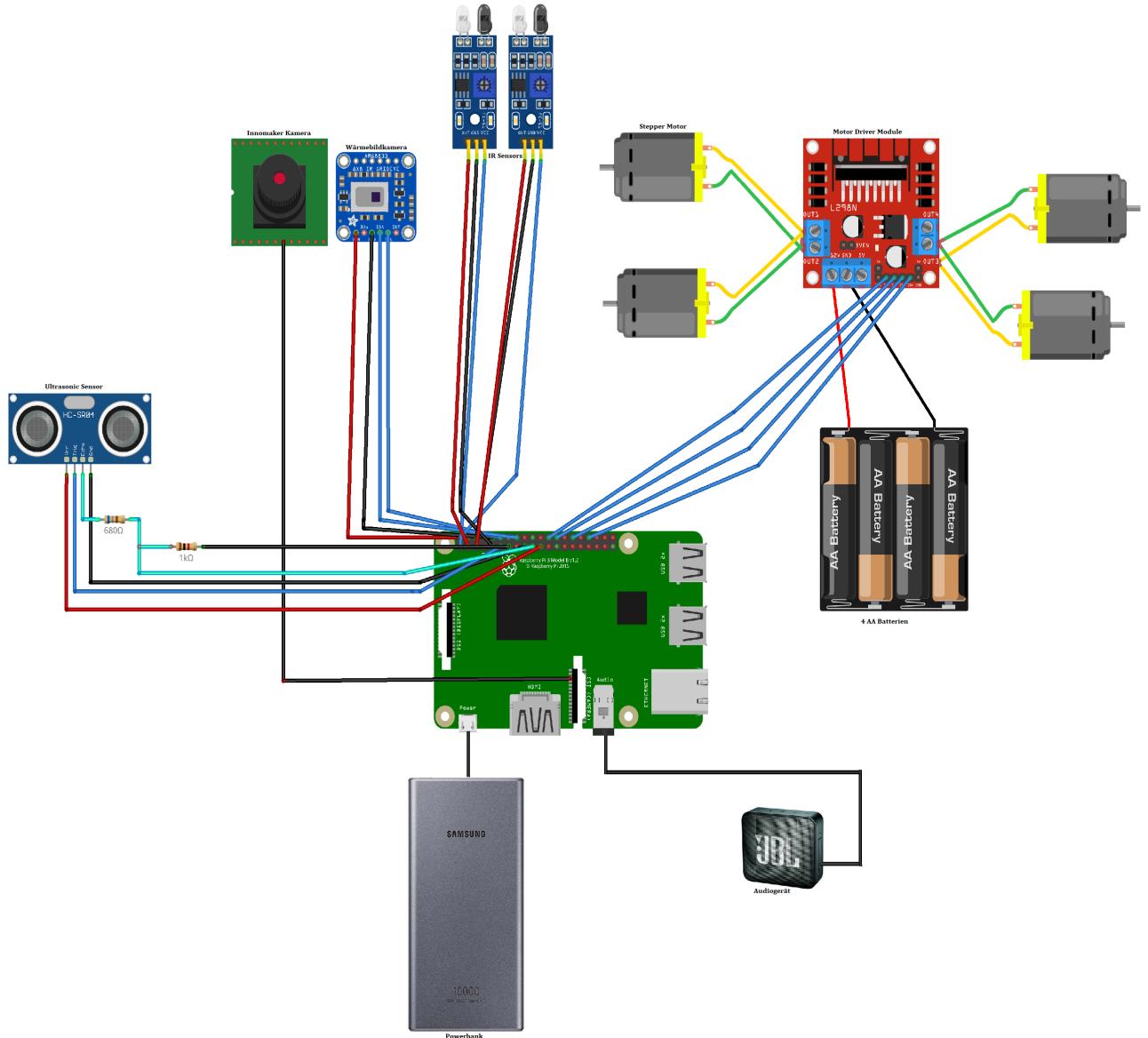


Abbildung 42: Gesamte Schaltung

Um unser Projekt zu integrieren, haben wir beschlossen, eine grafische Benutzeroberfläche für die verschiedenen Funktionen des Roboters zu erstellen. Daher haben wir uns für die Tkinter-Bibliothek entschieden, da sie leichtgewichtig und im Vergleich zu anderen Frameworks relativ einfach zu bedienen ist.

4.2.1 Graphical User Interface (GUI)

Python hat viele GUI-Frameworks, aber Tkinter ist das einzige Framework, das in die Python-Standardbibliothek integriert ist. Tkinter hat mehrere Stärken. Es ist plattformübergreifend, d. h. derselbe Code funktioniert auf Windows, macOS und Linux. Visuelle Elemente werden mit nativen Betriebssystemelementen gerendert, sodass mit Tkinter erstellte Anwendungen so aussehen, als gehörten sie auf die Plattform, auf der sie ausgeführt werden.[6]

```

from tkinter import *
from KeypadControlPC import *
from urllib.request import urlopen
from Line_Follower import follow
import base64
from threading import *
from cam_streaming.webstreaming import videotostreaming
from audio.amg88_temp import *

```

Abbildung 43: Python code für die Integration 1

Zuerst haben wir die Tkinter-Bibliothek importiert, die wir zur Erstellung unserer grafischen Benutzeroberfläche benötigen, dann die benötigten Module, die wir für unseren Corona-Roboter programmiert haben, KeypadControlPC, Line_Follower, Webstreaming, amg88_temp. und urllib.request, um ein Bild in unsere grafische Benutzeroberfläche zu importieren, und das Base64-Modul, um eine Funktion bereitzustellen, die Binärdaten in ein base64-kodiertes Format umwandelt.

```

def threading1():
    t1=Thread(target=videostreaming)
    t1.start()

def threading2():
    t2=Thread(target=temp_audio)
    t2.start()

```

Abbildung 44: Python code für die Integration 2

Dann haben wir 2 Threading-Funktionen definiert, die wir in unserer GUI verwenden werden. Einige Funktionen in unserem Programm erfordern eine Multithreading-Umgebung, z.B. Videostreaming oder Temperaturmessungen. Diese 2 Funktionen müssen gleichzeitig mit anderen Funktionen laufen, wenn wir sie anklicken. Denn wir wollen nicht nur unseren Roboter steuern, sondern ihn auch beim Videostreaming beobachten, bzw. wir wollen, dass unser Roboter einer Linie verfolgt und sowohl Videostreaming als auch Temperaturerfassung durchführt. So kommen die Threads ins Spiel.

```

app = Tk()
app.title("Controlling Interface")
app.geometry("600x1000")

```

Abbildung 45: Python code für die Integration 3

Danach haben wir unser Widget mit Tk() erstellt und es "Controlling Interface" genannt und unserem Fenster eine vernünftige Breite und Höhe (600,1000) gegeben.

```

img_url = "https://i.imgur.com/sdS4zeN.png"
img_byt = urlopen(img_url).read()
img_b64 = base64.encodestring(img_byt)
photo = PhotoImage(data=img_b64)
w = Label(app, image=photo)
w.pack()
ent = Entry(app)

```

Abbildung 46: Python code für die Integration 4

Das erste, was wir in unsere GUI einbauen wollten, war ein Foto unseres Roboters. Wir luden es auf eine Website hoch, damit wir eine eigene URL haben, wiesen wir die URL img_url zu, dann wurde sie mit urlopen gelesen und mit base64.encodestring kodiert. Dann haben wir das Foto oben auf der Seite platziert. Aufgrund von w.pack() bedeutet dies, dass die Position des Bildes auf den

Standardwert gesetzt wird, d. h. auf die oberste Ebene.

```
welcome=Label(app, text="\nRobot Interface\n")
welcome.config(font=('tahoma', 15, 'bold'))
welcome.pack()
```

Abbildung 47: Python code für die Integration 5

Wir haben einen Titel am oberen Rand des Widgets mit der Label-Funktion von Tkinter erstellt, dann sein Format mit welcome.config konfiguriert und ihn wieder an den oberen Rand gepackt (Standardzustand)

```
myButton = Button(app, text="Keyboard", command=Keyboard)
myButton.place(relx=0.5, rely=0.5, anchor=CENTER)
myButton.pack()

space=Label(app, text="\n")
space.config(font=('tahoma', 2, 'bold'))
space.pack()

myButton = Button(app, text="Line Follower", command=follow)
myButton.place(relx=0.5, rely=0.5, anchor=CENTER)
myButton.pack()

space=Label(app, text="\n")
space.config(font=('tahoma', 2, 'bold'))
space.pack()

myButton = Button(app, text="Video streaming", command=threading1)
myButton.place(relx=0.5, rely=0.5, anchor=CENTER)
myButton.pack()

space=Label(app, text="\n")
space.config(font=('tahoma', 2, 'bold'))
space.pack()

myButton = Button(app, text="Detect temperatures", command=threading2)
myButton.place(relx=0.5, rely=0.5, anchor=CENTER)
myButton.pack()

space=Label(app, text="\n\nDeveloped by:\nGhassen Jamoussi\nWadi Touil")
space.config(font=('italic', 12, 'bold'))
space.pack()

space=Label(app, text="\n© All rights are reserved to those 2 developers")
space.pack()
```

Abbildung 48: Python code für die Integration 6

Danach haben wir alle Schaltflächen erstellt, die wir brauchen, um zwischen unseren verschiedenen Optionen zu wählen: Tastatur, Line Follower, Video Streaming, Detect Temperatures. Als erstes wurden die Schaltflächen "Keyboard" der Funktion Keyboard() von KeypadControlPC und "Line Follower" der Funktion follow() von Line_Follower zugewiesen. Für das VideoStreaming und die Temperaturerkennung haben wir die Funktionen threading1 und threading2 der Befehlsoption des Buttons zugewiesen, dann haben wir den Stil und das Format des Buttons angepasst. Am Ende der Seite haben wir unten unser Copyright geschrieben.

```
app.mainloop()
```

Abbildung 49: Python code für die Integration 7

Am Ende des Programms schreiben wir app.mainloop(), um sicherzustellen, dass unsere grafische Benutzeroberfläche für immer geöffnet bleibt, bis ich sie manuell schließe.

So sieht unsere GUI aus :

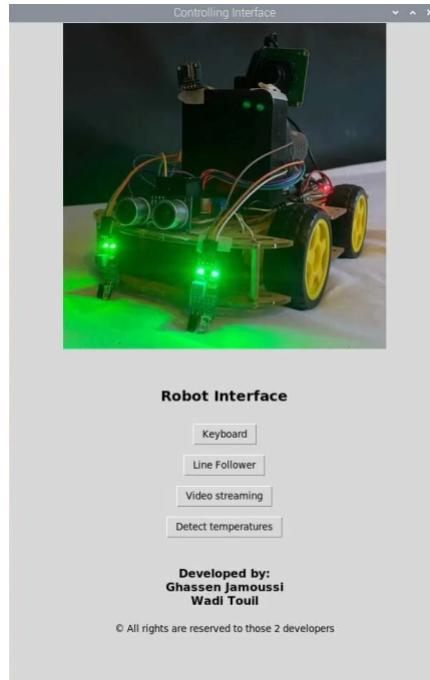


Abbildung 50: GUI

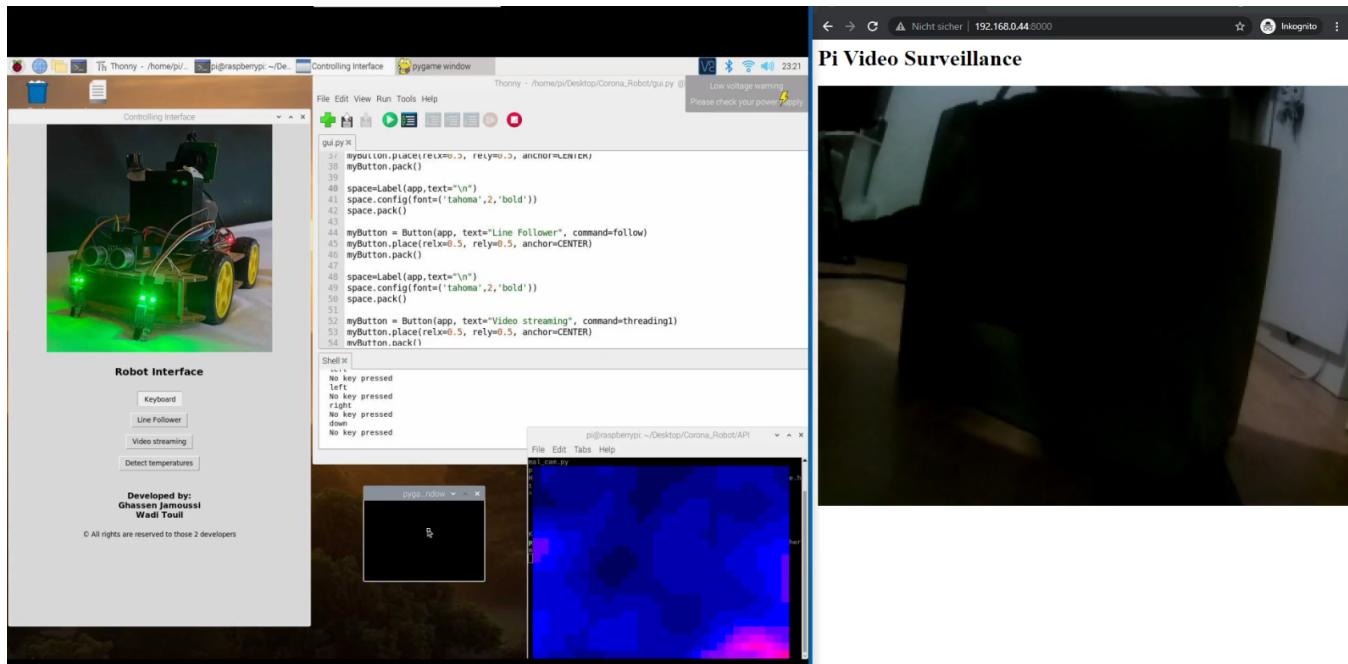


Abbildung 51: Robot Interface

Hier können wir zwischen Keyboard und Line_follower wählen. Wir können nicht beide gleichzeitig laufen lassen, deshalb sind sie nicht multithreadfähig. Andernfalls können wir mit einer dieser Funktionen gleichzeitig Videostreaming und Temperaturmessung ausführen, indem wir sie anklicken. Das Video für die Integration befindet sich im folgenden [Link](#).

5 Zusammenfassung und Ausblick

5.1 Zusammenfassung

Im Rahmen dieser Arbeit wurde ein Design eines Roboters und dessen Implementierung für den Einsatz auf einer spezifischen Hardware in einer Kennzeichenvorbereitungsstation entwickelt.

Zu Beginn der Arbeit wird der Analyse- und Entwurf erstellt, der anhand von drei Szenarien analysiert wird und somit die Grundlage für die Erstellung des Entwurfs bildet. Hier werden im ersten Schritt die Systembeschreibung und die Anforderungen an die Hard- und Software analysiert. Dort haben wir eine abstrakte Struktur des Projekts erstellt, um dem Leser eine Vorstellung zu vermitteln. Dann haben wir ein Anwendungsfalldiagramm gezeichnet und deren Funktionalität beschrieben und schließlich ein Flowchartdiagramm, das unsere Computeralgorithmen darstellt und die komplexen Prozesse verdeutlicht, die wir implementiert haben.

Nachdem die verschiedenen Anforderungen und Funktionen für das System festgelegt waren, begannen wir mit der Implementierung. Wir haben jede Hardware, die wir für unser Projekt verwendet haben, im Detail beschrieben. Für jede Komponente haben wir eine separate Verdrahtung vorgenommen, um sie zu verdeutlichen, und sie dann zu einem Gesamtentwurf zusammengeführt. Nach der Beschreibung der Hardware-Spezifikationen und ihrer Ziele sind wir zur Software-Perspektive übergegangen, wo wir unseren Code ausführlich erklärt haben und warum wir diese Funktionen implementiert haben. Fast am Ende jedes Abschnitts haben wir die Funktionalität aufgezeichnet und verlinkt.

Unser gesamtes Projekt befindet sich in Gitlab unter diesem Link :

https://gitlab.beuth-hochschule.de/s83006/ap_corona_robot

5.2 Ausblick

Die Arbeit hat gezeigt, dass Raspberry Pi sehr robust ist und es unglaublich viel Spaß macht, damit zu basteln. Die Anwendungen, die wir für unseren Corona-Roboter mit Raspberry Pi verwendet haben, waren unbegrenzt. Wir haben verschiedene Features implementiert und eine Menge aus dem Projekt gelernt.

Wir haben maschinelles Lernen durch Gesichtserkennung, Videostreaming, Multithreaded Umgebung, Signalfunktionen, grafische Benutzeroberfläche und Designarchitekturen implementiert. Das Projekt hat immer noch das Potenzial, sich weiter zu entwickeln und komplexer zu werden. Wir könnten zum Beispiel zusätzliche Ultraschallsensoren auf der rechten und linken Seite und hinten am Auto anbringen, anstatt nur vorne, so dass wir die vollständige Sicherheit des Autos gewährleisten können. Wir könnten auch eine bessere Wärmebildkamera einbauen, um Temperaturen direkt zu erkennen und die Gesichtserkennung direkt von der Wärmebildkamera und nicht von einer normalen Kamera zu realisieren. Wir könnten den Roboter auch zu einem 2-Wege-Kommunikationsgerät entwickeln, was bedeutet, dass wir die Antwort des Beifahrers erhalten und ihm in Echtzeit antworten, als ob es sich um ein normales Gespräch würde, aber es findet aus der Ferne statt, und so viele andere Funktionen könnten eingebaut werden ...

Im Grunde genommen haben wir fast alles, was wir im Studiengang Technische Informatik gelernt haben, in diesem Projekt umgesetzt. Wir sind stolz darauf und bereit, in Zukunft weitere interessante Projekte durchzuführen, die uns akademisch und geistig weiterbringen.

Literatur

[1] Methodologie

<https://www.teamwork.com/project-management-guide/project-management-methodologies/>

[2] Definition vom Raspberry Pi

<https://www.raspberrypi.org/help/what-%20is-a-raspberry-pi/>

[3] GPIO

<https://projects.raspberrypi.org/en/projects/physical-computing/1>

[4] Definition des Flowchartdiagramms

www.lucidchart.com/pages/what-is-a-flowchart-tutorial

[5] Audio

www.circuitbasics.com/how-to-play-audio-with-the-raspberry-pi/

[6] Tkinter

<https://realpython.com/python-gui-tkinter/>

[7] Thermokamera

<https://makersportal.com/blog/thermal-camera-analysis-with-raspberry-pi-and-opencv>

Abbildungsverzeichnis

1	Beispiel von unserem Kanban Board	3
2	Raspberry Pi	3
3	GPIO Pins	4
4	IOT Konzept	5
5	Robot Kit	6
6	Hardware Komponenten	7
7	Elektronische Materialien	8
8	Abstrakter Aufbau des Robots	9
9	Anwendungsfalldiagramm des Corona-Robots	12
10	Flowchartdiagramm des Corona Robots	13
11	Raspberry Pi Verbindungen	15
12	Pi Desktop	15
13	Anschluss der L29N Module mit Raspberry Pi	16
14	Python code für Motorsteuerung	17
15	Roboter-Richtungen	17
16	Python code für Motorsteuerung mit Keypads	18
17	Anschluss der IR Sensors mit Raspberry PI	19
18	Python code für Linienverfolger	20
19	Ultrasonic Sensor	21
20	Anschluss des Ultrasonic-Sensors mit Raspberry Pi	22
21	Python code für den Abstandssensor	23
22	Python code für den Test des Abstandssensors	24
23	Anschluss des Audiogeräts mit Raspberry Pi	24
24	Python code für das audioPlay Programm	25
25	Pixelarray und Sichtfeld	26
26	Anschluss des AMG88-Kameras zu Raspberry Pi	26
27	Pixelarray und Sichtfeld	27
28	Python code für audio/amg88temp.py 1	27
29	Python code für audio/amg88temp.py 2	27
30	Python code für audio/amg88temp.py 3	28
31	8x8 Matrize vom AMG88xx	29
32	Innomaker Kamera	30
33	Anschluss der Innomaker Kameras mit Raspberry Pi	30
34	Struktur der Videostreaming-Funktion	31
35	Python Code für Videostreaming Implementierung 1	31
36	Python Code für Videostreaming Implementierung 2	32
37	Python Code für Videostreaming Implementierung 3	32
38	Python Code für Videostreaming Implementierung 4	33
39	Html Code für den Flask-Webbrowser	33
40	Python Code für Videostreaming Implementierung 5	34
41	Python Code für Videostreaming Implementierung 6	34
42	Gesamte Schaltung	35
43	Python code für die Integration 1	36
44	Python code für die Integration 2	36
45	Python code für die Integration 3	36
46	Python code für die Integration 4	36
47	Python code für die Integration 5	37
48	Python code für die Integration 6	37
49	Python code für die Integration 7	37

50	GUI	38
51	Robot Interface	38

Tabellenverzeichnis

1	Kosten der Komponenten	10
2	Spezifikation der Ultrasonic Sensor	21
3	Spezifikation der Innomaker Kamera	30