Hochschule Aalen

Mobile Embedded Software

RobiWaiter

Professor Oberhauser

Florian Merlau (81775), Daniel Stuckenberg (83281), Johannes Preisach (), Andre Derjagin (83091)

Inhaltsverzeichnis

Projektübersicht

Projektverantwortlichkeiten

Anfangs haben wir uns alle mit dem Aufsetzen von ROS und der Implementierung von ROS mit unserer Hard- und Software beschäftigt. Im späteren Verlauf des Projektes haben wir die Aufgabenverteilung in folgende Aufgabengebiete aufgeteilt:

Johannes Preisach & Florian Merlau:
- Einrichtung, Konfiguration und das Arbeiten mit ROS (Robot Operating System) und dem Roboter (Robi)
- Erstellung und Verwaltung der Karte mit dem LiDAR Sensor und mit der Methode SLAM (was bedeutet das ausgeschrieben?)

Daniel Stuckenberg:
- Entwicklung eines Konzeptes und einschließliche Implementierung der Bilderkennung für das Geschirr

Andre:
- Design und Entwicklung der Web-Applikation
- Entwicklung und Integration der Sprachsteuerung

Projektbeschreibung

Entwicklung eines unterstützenden Roboters für Gastronomiebetriebe

Überblick:
In der Gastronomie zählt jeder Moment und jeder Handgriff. Mit dem Ziel, Arbeitsabläufe zu straffen und den Service für die Gäste zu verbessern, arbeiten wir an der Entwicklung eines Service Roboters. Dieser soll in Restaurants und Cafés zum Einsatz kommen und das Personal bei alltäglichen Aufgaben, wie z.B. der Auslieferung von Getränken und Speisen, entlasten, damit dieses sich mehr auf die Betreuung der Gäste konzentrieren kann.

Projektziele:
Wir möchten einen Roboter entwickeln, der nicht nur funktional, sondern auch benutzerfreundlich ist. Folgende Hauptfunktionen sind geplant:

1. Webinterface-Steuerung: Der Roboter wird über eine klare und einfache Benutzeroberfläche gesteuert. Das ermöglicht dem Personal, Aufgaben zu delegieren und den Status des Roboters zu überwachen.

2. Autonome Navigation: Der Roboter soll sich eigenständig durch die Räumlichkeiten bewegen können. Er findet selbstständig zu bestimmten Orten wie Tischen oder der Spülküche.

3. Objekterkennung: Der Roboter nutzt eingebaute Kameras und intelligente Algorithmen, um zu erkennen, ob Geschirr auf dem Anhänger platziert ist.

4. Sprachsteuerung: Durch die Integration eines Sprachassistenten kann das Personal einfach und schnell mit dem Roboter kommunizieren. Zudem kann der Roboter auf Kundenwünsche eingehen und deren Fragen beantworten.

Anwendungsbereich:
Der Roboter ist primär als Unterstützung für das Personal gedacht. Er übernimmt Routinetätigkeiten wie das Transportieren von Geschirr, sodass das Personal mehr Zeit für Service und direkten Kundenkontakt hat.

Projektziel:
Unser Ziel ist es, den Arbeitsalltag in der Gastronomie durch den Einsatz unseres Service Roboters zu erleichtern. Wir möchten Routineaufgaben automatisieren und die Interaktion zwischen Personal und Gästen verbessern, um einen reibungsloseren Ablauf und eine angenehmere Atmosphäre für die Gäste zu schaffen. Der Roboter soll als zuverlässige Unterstützung im Tagesgeschäft dienen, einfach zu bedienen sein und durch ein Touchscreen-Interface intuitiv von Personal und Gästen gesteuert werden können.

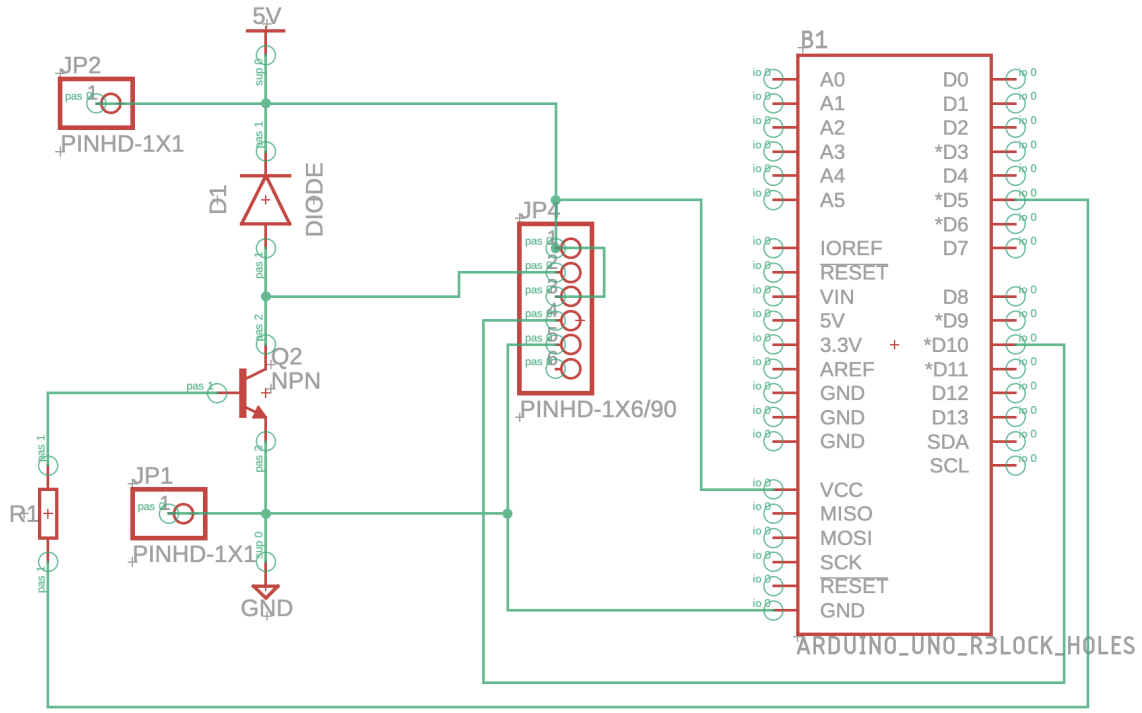
Anfangsidee

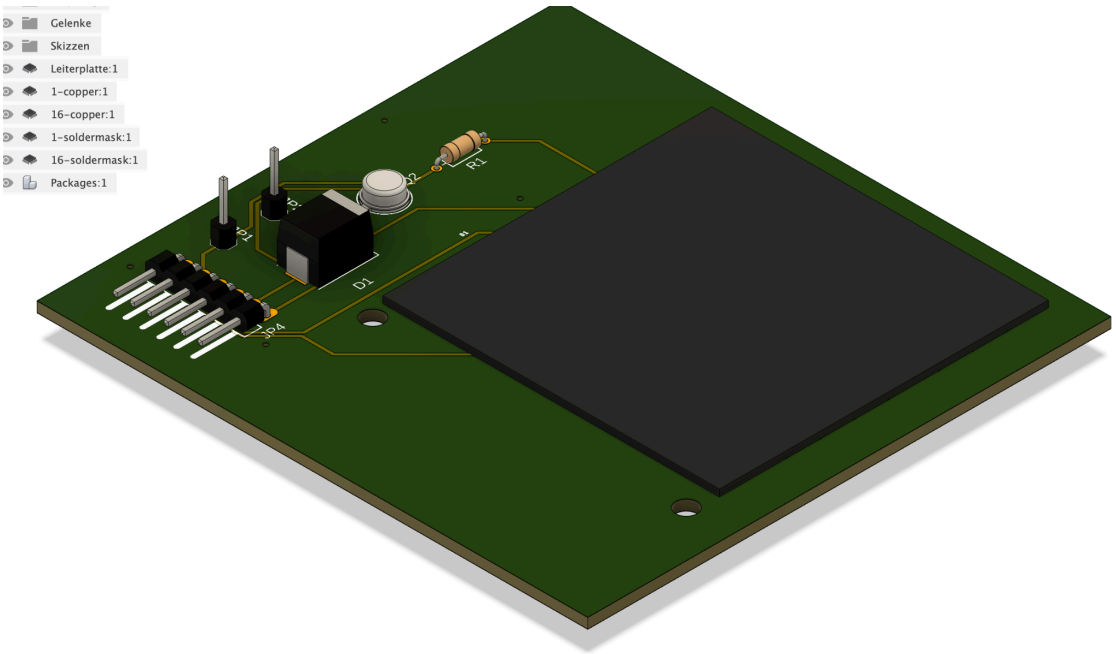
Anfänglich waren wir ambitioniert, einen eigenen Roboter zu konzipieren, inklusive einer speziell entwickelten Motorsteuerung und einem innovativ modifizierten Lidar, den wir aus einem Staubsaugerroboter extrahieren wollten. Wir hatten vor, eine eigene Platine zu entwerfen, die mit einer ausgeklügelten Elektronik ausgestattet sein sollte, um den Motor des Lidars präzise anzusteuern. Unsere Vision war es, einen Roboter zu bauen, der nicht nur durch seine Funktionalität, sondern auch durch seine Einzigartigkeit besticht.

Im Laufe des Projekts mussten wir jedoch feststellen, dass unsere Zeitressourcen begrenzter waren, als wir angenommen hatten. Der Prozess der Entwicklung und Integration der einzelnen Komponenten erwies sich als komplexer und zeitaufwendiger als erwartet. Nach reiflicher Überlegung und Abwägung der verfügbaren Optionen entschieden wir uns dazu, unsere Pläne anzupassen.

Wir verlagerten unseren Fokus auf den Turtlebot, eine bereits etablierte und zuverlässige Plattform, die alle notwendigen Komponenten fertig integriert hatte. Dieser Schritt ermöglichte es uns, unsere Zeit effizienter zu nutzen und uns auf die Optimierung und Implementierung der spezifischen Funktionen zu konzentrieren

Platinendesign





Funktionen und Fähigkeiten

1. Spracherkennung:
Der Roboter kann verstehen, was gesagt wird, fast so, als würde man mit einem Menschen reden. Egal ob das Personal ihm sagt, wohin er gehen soll oder ob Gäste eine Frage haben – der Roboter hört zu und reagiert. Sobald das Kommando “Ok Robi” ausgesprochen wird, weiß der Roboter, dass das Personal mit ihm spricht und reagiert dementsprechend anders, als wenn er mit Kunden sprechen würde. Wenn der oben genannte Befehl nicht gesagt wird, geht der Roboter davon aus, dass er mit Kunden des Restaurants spricht.

2. Autonomes Abholen des Geschirrs:
Einer der coolsten Features ist, dass der Roboter von selbst loszieht, um benutztes Geschirr von den Tischen einzusammeln. Das bedeutet, das Personal hat mehr Zeit für wichtigere Aufgaben.

3. Bilderkennung des Geschirrs:
Der Roboter ist nicht nur ein einfacher Helfer, der Dinge hin und her trägt. Er hat auch Augen – also Kameras, die ihm helfen, zu erkennen, ob Geschirr vorhanden ist oder nicht.

4. Zielgerichtete Navigation:
Das Personal hat die Kontrolle: Mit ein paar Klicks auf einem Bildschirm, oder durch die Spracherkennung des Roboters, können sie den Roboter an einen bestimmten Tisch schicken. So wissen sie immer, dass der Roboter genau dort ist, wo er gebraucht wird.

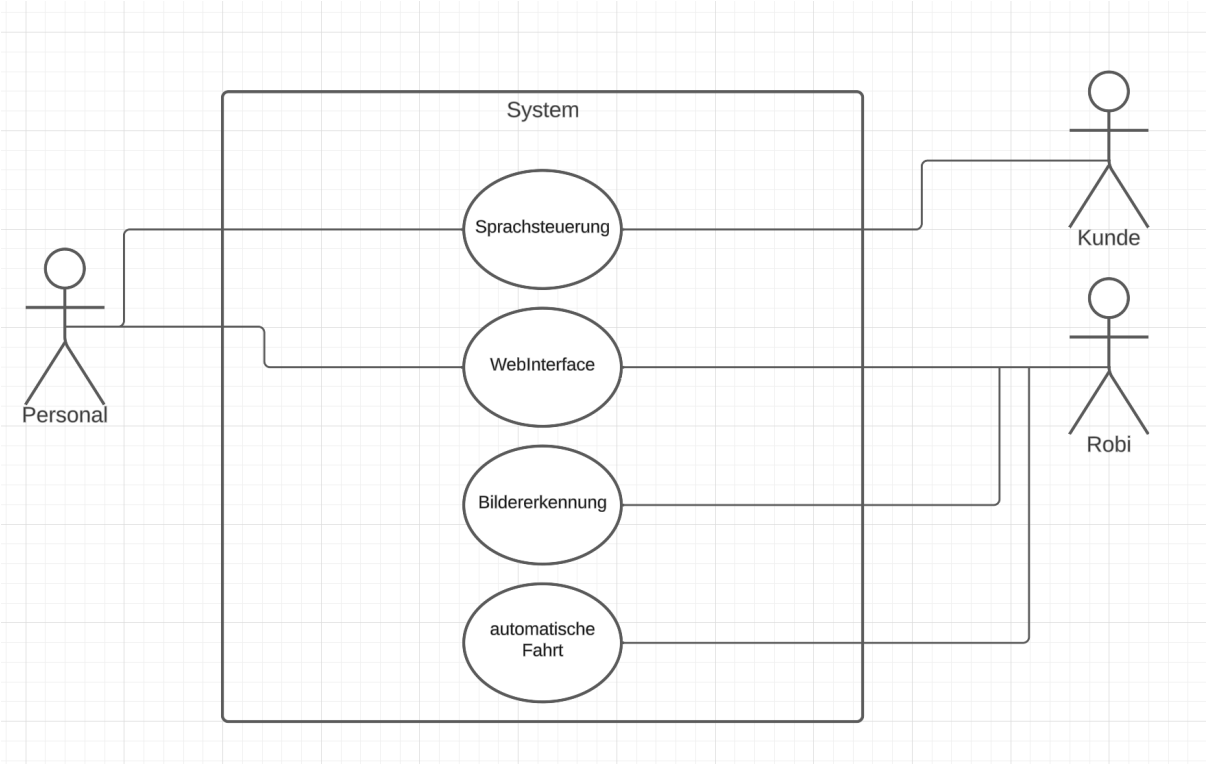
Szenariobeschreibung

Szenario

In unserem Restaurant führt der starke Kundenstrom zu einer hohen Arbeitsbelastung, die unser begrenztes Personal kaum bewältigen kann. Um die Effizienz zu steigern und den Arbeitsaufwand ohne Einstellung zusätzlichen Personals für Routineaufgaben zu managen, setzen wir auf innovative Lösungen.

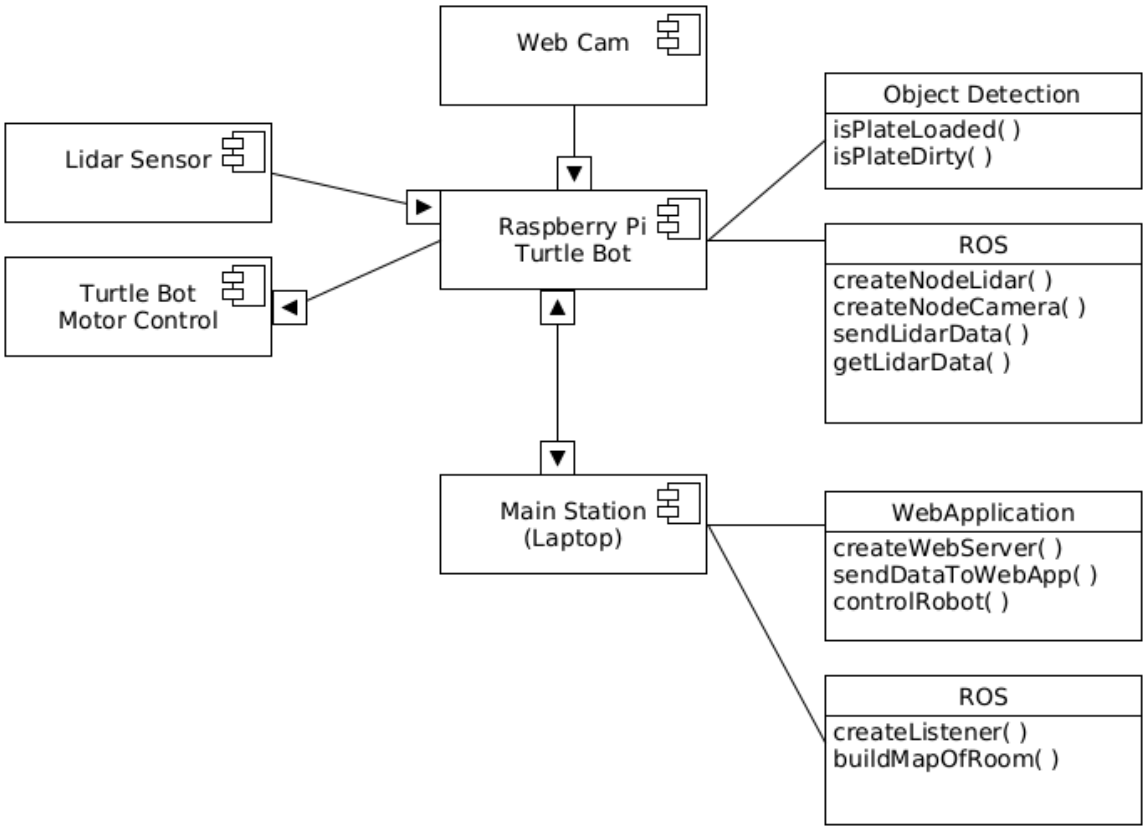
Sobald das Personal bemerkt, dass Tisch 5 fertig gegessen hat und das Geschirr bereit zur Abholung ist, wird am Steuerungs-Terminal der Standort des Tisches auf einer digitalen Karte ausgewählt. Daraufhin setzt sich unser autonomer Service-Roboter in Bewegung Richtung des Tisches. Bei Ankunft am Ziel interagiert der Roboter höflich mit den Gästen und wartet darauf, dass das Geschirr auf seinem Transportwägelchen platziert wird. Mittels eingebauter Kamera überprüft der Roboter, ob das Geschirr korrekt aufgeladen wurde. Ist das Wägelchen beladen, navigiert der Roboter selbständig zurück zur Küche, um das Geschirr abzuladen. Nach einer abschließenden Überprüfung, ob das Geschirr entladen wurde, kehrt der Roboter zu seiner Ladestation zurück, um sich für den nächsten Einsatz vorzubereiten. So unterstützt der Roboter das Personal bei der Bewältigung der hohen Arbeitsbelastung, indem er einfache, aber zeitintensive Aufgaben übernimmt.

Use Case



Architektur

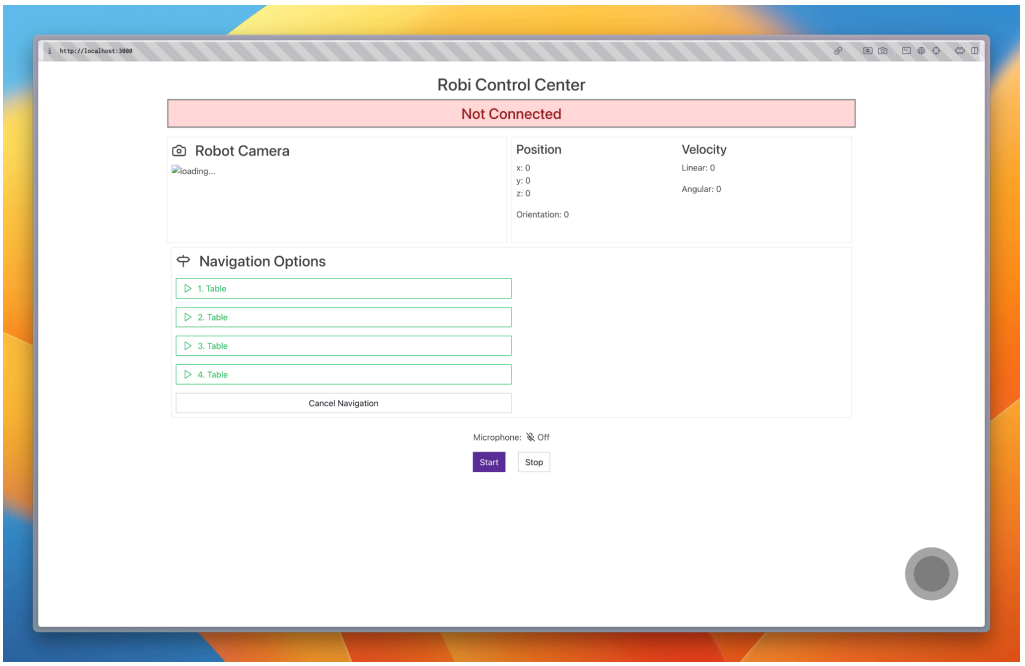
Unsere Systemarchitektur ist ziemlich einfach und übersichtlich, was größtenteils auf die Verwendung von ROS (Robot Operating System) zurückzuführen ist. ROS liefert viele Standardfunktionen, die wir nutzen. Wir haben ROS auf zwei Geräten eingerichtet: einem Remote-PC und einem Raspberry Pi, der unseren Roboter 'Robi' antreibt. Diese beiden Geräte sind über ein Netzwerk miteinander verbunden, sodass sie Daten austauschen können.
Die Kamera ist auch in das ROS-Netzwerk integriert. Alle Daten werden über ROS Topics veröffentlicht und abonniert, was die Kommunikation und Steuerung vereinfacht.


Web Cam
Object Detection
isPlateLoaded( )
isPlateDirty( )
Lidar Sensor
Raspberry Pi
Turtle Bot
ROS
createNodeLidar( )
createNodeCamera( )
sendLidarData( )
getLidarData( )
Turtle Bot
Motor Control
Main Station
(Laptop)
WebApplication
createWebServer( )
sendDataToWebApp( )
controlRobot( )
ROS
createListener( )
buildMapOfRoom( )

Web-Applikation

Die Web-Applikation läuft auf React 17 und wird nicht als Progressive-Web-App (PWA) angeboten. In der ersten Iteration der App habe ich noch eine PWA Funktionalität eingebunden. Aufgrund von Kompatibilitätsproblemen musste ich diesen initialen Entwurf leider verwerfen und mit einem neuen Entwurf fortfahren, bei dem ich (Andre) aufgrund der zeitlichen Komponente nicht in der Lage war, die gewünschte PWA Funktionalität einzubinden. Jedoch hat der Benutzer unserer App, dennoch die Möglichkeit die App auf seinem mobilen Endgerät zu verwenden, da wir während der Entwicklung unseren Fokus auf ein Responsive Web Design gelegt haben und die Applikation dadurch auf dem Handy, dem Desktop Computer, aber auch auf einem Tablet, die einhundertprozentige Funktionalität bietet.

User Interface



Sprachsteuerung

Die Sprachsteuerung läuft auf der Web-Applikation und wird mit Hilfe dem “SpeechRecognition” Interface der “Web Speech”-API des Browsers, sowie die “Text-Completion”- und “Text-To-Speech”-API von OpenAI gehandhabt.
Das SpeechRecognition Interface ist für die Eingabe der Stimme des Benutzers sowie die Umwandlung der Stimme in ein Textformat verantwortlich. Dieser Text wird auf dem Bildschirm in einer Box ausgegeben. Dies soll dem User helfen, seinen Input nachvollziehen zu können. Nachdem der Text vom Benutzer gesprochen und durch die Web Speech API verarbeitet wurde, wird dieser an die Text Completion API von OpenAI gesendet, es wird eine Antwort generiert und an den Browser zurückgeschickt. Mit dieser Antwort wird erneut eine Anfrage an OpenAI gesendet. Bei der zweiten Anfrage wird die Anfrage an die Text-To-Speech API geschickt und es wird eine entsprechende Audioausgabe der zuvor generierten Textes erstellt. Diese Ausgabe wird im Code in der “callOpenAiTTS” Funktion in

ein Blob-Objekt umgewandelt und mit Hilfe der “Web Audio”-API im Browser des Benutzers ausgegeben.

Objekterkennung

Die Daten der Kamera, die an den Raspberry Pi des Roboters angeschlossen ist, werden über ROS übertragen. Dazu werden die Bilder über ein Python-Skript, welches OpenCV verwendet, als Nachricht über einen ROS Node übertragen. Diese Daten werden über die ROS Bridge an die React-Anwendung übertragen. Dazu muss in der Anwendung auf das von der ROS-Anwendung veröffentlichte Topic subscribed werden. Die eingehenden Daten werden in der Anwendung zu einem Bild transformiert,gleichzeitig wird die API von Tensorflow.js aufgerufen, um die Objekte auf dem Bild zu erkennen. Als Modell wird MobileNet verwendet. Die Bilddaten werden sekündlich an die API gesendet, die das Bild auswertet und die Klasse (welches Objekt erkannt wurde) und die berechnete Wahrscheinlichkeit, mit der die Klasse erkannt wurde, angibt. Wenn Geschirr auf dem Anhänger erkannt wird, wird es als grüner Rahmen in der Anwendung optisch erkennbar gemacht.

ROS & Robi

Der Robi ist ein fortschrittliches Robotersystem, das in zwei Hauptkomponenten unterteilt ist: einen Single-Board-Computer (SBC) und einen Remote-PC. Diese zweigeteilte Architektur ermöglicht eine effiziente Aufgabenverteilung und optimiert die Leistung des gesamten Systems.

1. Single-Board-Computer (SBC):
Der SBC, häufig ein Raspberry Pi, fungiert als Kern des Roboters. Er steuert die grundlegenden Funktionen des Roboters, einschließlich Motorsteuerung, Sensor Ablesung und die Ausführung einfacher, lokaler Berechnungen. Seine kompakte Größe und Energieeffizienz machen ihn ideal für die direkte Integration in den Roboter.

2. Remote-PC:
Der Remote-PC ist für rechenintensive Aufgaben vorgesehen, die über die Kapazitäten des SBC hinausgehen. Dies schließt insbesondere anspruchsvolle Prozesse wie die Objekterkennung ein. Der Remote-PC ist in der Regel leistungsfähiger und kann komplexe Algorithmen und Berechnungen durchführen, die für eine intelligente Navigation und Interaktion des Roboters erforderlich sind.

Beide Systeme sind durch ein gemeinsames Netzwerk verbunden, welches eine reibungslose Kommunikation und Datenübertragung ermöglicht. Die Kommunikation zwischen ihnen erfolgt über das ROS, ein vielseitiges Framework für die Entwicklung von Robotik-Software. ROS ermöglicht eine strukturierte Kommunikationsumgebung, in der Nachrichten über sogenannte Topics ausgetauscht werden.

Innerhalb des ROS-Ökosystems können die beiden Komponenten des Roboters:

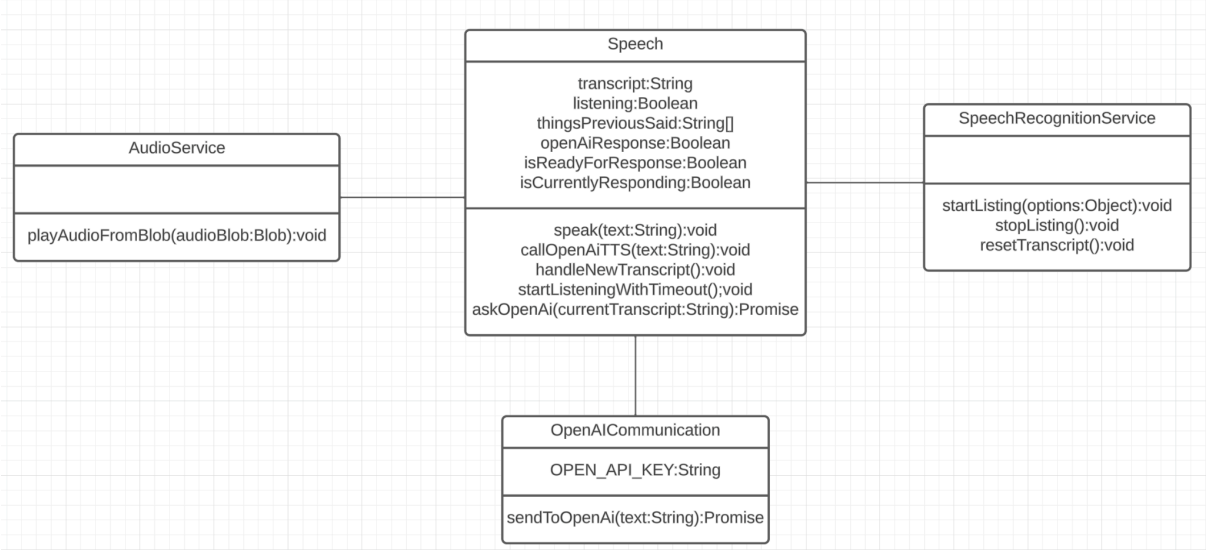
- Topics abonnieren (Subscribe): Hier können die Systeme kontinuierliche Datenströme von Sensoren oder Statusaktualisierungen empfangen. Der SBC könnte beispielsweise ein Topic abonnieren, um kontinuierlich Sensordaten zu erhalten.

- Topics publizieren (Publish): Beide Systeme können Daten oder Befehle senden. Wenn der Remote-PC beispielsweise eine Objekterkennung durchgeführt hat, könnte er die Ergebnisse auf einem Topic veröffentlichen, sodass der SBC entsprechend darauf reagieren kann.

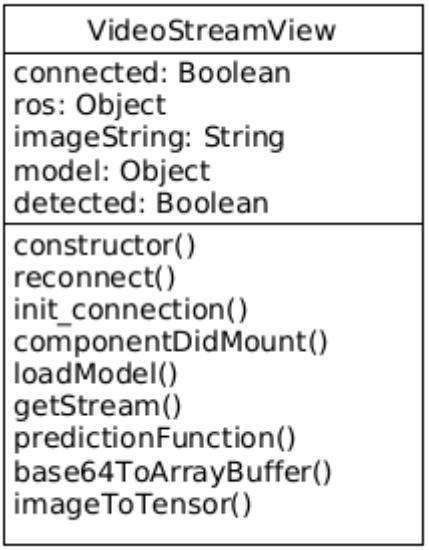
Statische Ansicht der internen Komponenten

Klassendiagramme

Sprachsteuerung - speech.tsx



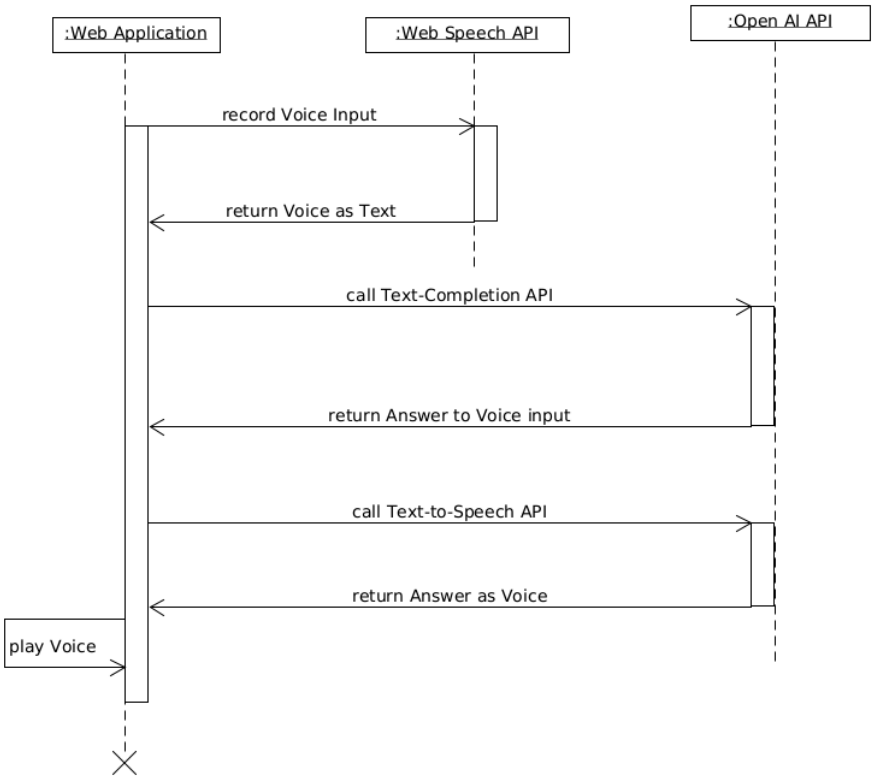
Objekterkennung



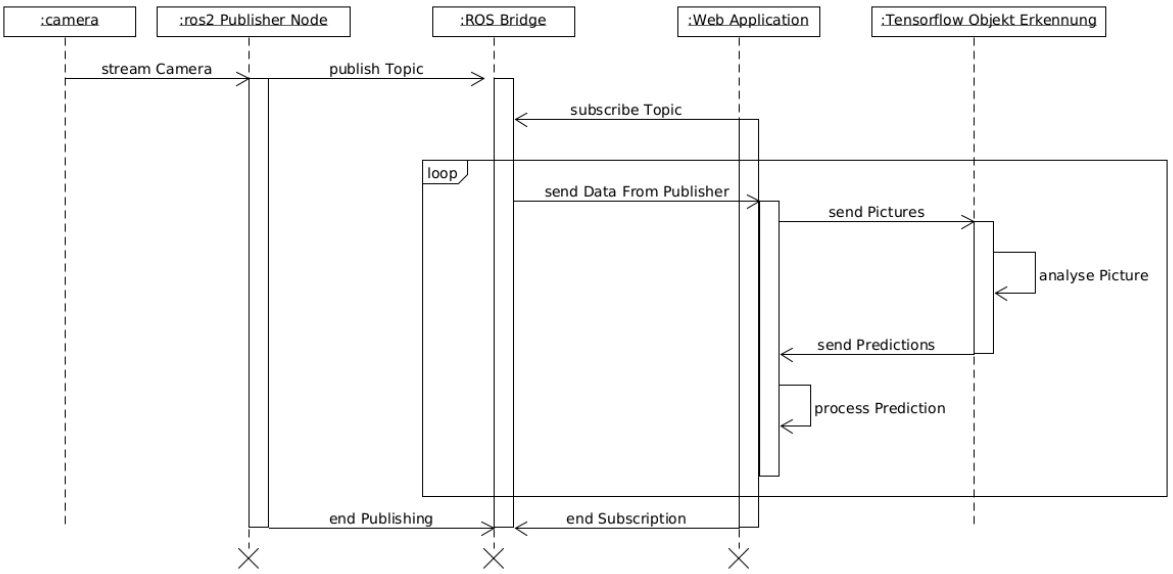
Dynamische Ansicht

Sequenz Diagramm

Sprachsteuerung - speech.tsx



Objekterkennung



Referenzen

Web-Applikation

Die Inspiration, sowie der Code für die Oberfläche und die Verbindung zwischen der Applikation und dem Roboter entstammt aus diesem Video:
Developing a ReactJS Robot Control Web App with ROSlib.js | Code Included.
Die im src/components liegenden .jsx Dateien sind aus dem Video und wurden vom Ersteller des Videos geschrieben. Einige von uns verwendete Komponenten wurden jedoch verändert und auf unseren Anwendungsbereich zugeschnitten. Zudem habe ich die Benutzeroberfläche aus dem Video ein bisschen verändert, etwas anschaulicher umgesetzt und für unseren Anwendungsfall angepasst. Die speech.tsx Datei im src/audio Ordner sind von mir (Andre) geschrieben und erstellt worden. Die gesamte Speech-Komponente beinhaltet alle Funktionen, die für die Audio-Ein- und Ausgabe verantwortlich sind.

TurtleBot 3

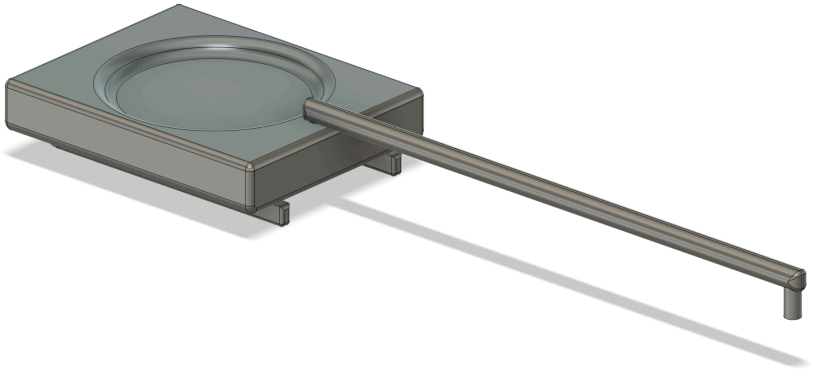
Für die Einrichtung und Inbetriebnahme des TurtleBots haben wir die offizielle Anleitung des TurtleBot-Herstellers als Leitfaden genutzt und auf Basis dieser unser Projekt aufgesetzt.

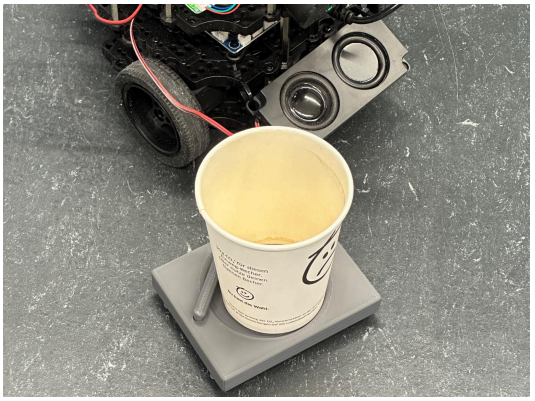
Objekterkennung

Das Python-Skript für die Übertragung der Kameradaten über ROS wurde größtenteils aus dem Artikel Getting Started with OpenCV übernommen, wobei der Nachrichtentyp und die zu übertragenden Daten geändert wurden.

3D Druck

Für unseren Prototypen haben wir speziell angepasste 3D-Modelle entworfen, um eine Tasse sicher platzieren und transportieren zu können. Nach der erfolgreichen Modellierung wurde das Design mittels eines 3D-Druckers realisiert, und die Funktionsfähigkeit des gedruckten Teils konnte in der Praxis erfolgreich bestätigt werden.





Probleme und Gelerntes

Sprachverarbeitung

Probleme bei der Sprachausgabe hatte ich (Andre) an sich nicht, am Anfang habe ich es mir schwer getan, herauszufinden, wie ich die Umsetzung gestalten soll, aber als ich ein passendes Tutorial gefunden hatte, wusste ich, wie ich an das Problem herangehen musste. Das Problem bei meinem gefundenen Tutorial war, dass die komplette Applikation auf der “Web Speech”-API aufgebaut war und für unseren Anwendungsfall war das für uns zu ungenau. Meine Lösung war es, die aus dem Tutorial bestehenden Funktionen, mit Hilfe von weiteren Tutorials, so umzubauen, dass sie eine Fetch-Anfrage an OpenAI senden und den entsprechenden Output zurückbekommen. Um passende Antworten von der Chat-Completion API von OpenAI zu erhalten, habe ich an die unsere Anfrage ein Prompt geschrieben, der unseren Anwendungsfall und unsere wünsche an die Antwort beschreibt. Ansonsten gab es soweit keine weiteren Probleme mit dem Sprachassistenten.

Web-Applikation

Die erste Version der Web-Applikation wurde von mir (Andre) selber designt und geschrieben. Das Problem dabei war, dass ich damals die neuesten Versionen von React und Nextjs verwendet habe. Dadurch hatte ich Probleme mit der Verwendung der verschiedenen ROS-Packages, die für die Verbindung der App zum Roboter nötig waren, da diese sehr veraltet waren. Zudem wollte ich das Package “React-ROS” verwenden, die mir eine vereinfachte Verbindung zwischen ROS und der App ermöglicht hätte, aber das war mir auch nicht möglich, da die React Versionen zwischen unserer Applikation und dem React-ROS Package nicht kompatibel waren. In unserer App haben wir React 18 verwendet, das React-ROS Package lief jedoch auf React 16. Nach mehreren fehlgeschlagenen Versuchen, die Versionen miteinander zum Laufen zu bringen, habe ich mich dazu entschieden, die React Version der Applikation herunterzusetzen. Als ich im Internet nach Inspirationen für die Verbindung zwischen unserer Applikation und ROS gesucht habe, bin ich auf das bereits im oberen Kapitel genannte Video gestoßen und habe mich dazu entschlossen, den Code des Videos für unsere Applikation zu verwenden und ggf. Anpassungen vorzunehmen. Mit dem Code aus dem Video hatte ich für die Applikation soweit keine nennenswerten, großen Probleme.

ROS

Wir hatten ziemlich Probleme damit, ROS auf dem Remote-PC zum Laufen zu bringen, da wir eine eigene virtuelle Maschine dafür aufgesetzt haben, in der ROS Humble laufen sollte. Da aber oft eine Spezifikation nicht passte, mussten wir die VM einige Male erneut aufsetzen, bis wir ein lauffähiges System mit ROS hatten. . Hinzu kamen Netzwerkprobleme, weil der Roboter sich nicht richtig mit unserem aufgebauten Netzwerk verbinden wollte. Ein weiteres großes Problem war es, die Karte mit SLAM zu erstellen und diese in der Navigations-Node zu verwenden, um den Roboter durch diese steuern zu lassen. Dies alles wurde zusätzlich dadurch erschwert, dass die Dokumentation für den Turtlebot sehr spärlich war und uns nur das Nötigste lieferte. Deshalb mussten wir selbst viel recherchieren und experimentieren, was zusätzlich viel Zeit in Anspruch nahm. Ein zusätzliches Hindernis war der Einsatz von Gazebo; anfangs konnten wir es nicht nutzen, da wir einen ARM-basierten Rechner (Virtuelle Maschine auf einem Macbook Pro mit einem M1 Chip) verwendeten. Später sind wir auf einen x86-Rechner umgestiegen, weil wir vermuteten, dass die Probleme vielleicht am ARM-Prozessor liegen könnten. Zwar konnte dadurch Gazebo nun genutzt werden, dennoch stellte sich heraus, dass das nicht die Lösung war. Denn auch als wir mit dieser VM den Roboter navigieren wollten, funktionierte die Navigationnode nicht (es wurde immer noch keine Karte, die zuvor aufgenommen wurde geladen). Auch das ständige Neuaufsetzen des Raspberry Pi kostete uns viel Zeit. Um diese Herausforderungen zu bewältigen, haben Johannes und ich (Florian) unsere Kräfte gebündelt und sind gemeinsam an dem Problem angegangen. Wir konnten das Problem dann zunächst lösen, indem wir von der ROS-Version Humble auf Foxy umgestiegen sind, was die Navigation Node nun endlich zur gewünschten Ausführung verhalf . Dabei haben wir eine Menge über ROS lernen können, wie zum Beispiel alles rund um Topics, Services, SLAM, Mapping und die Steuerung.

Bilderkennung

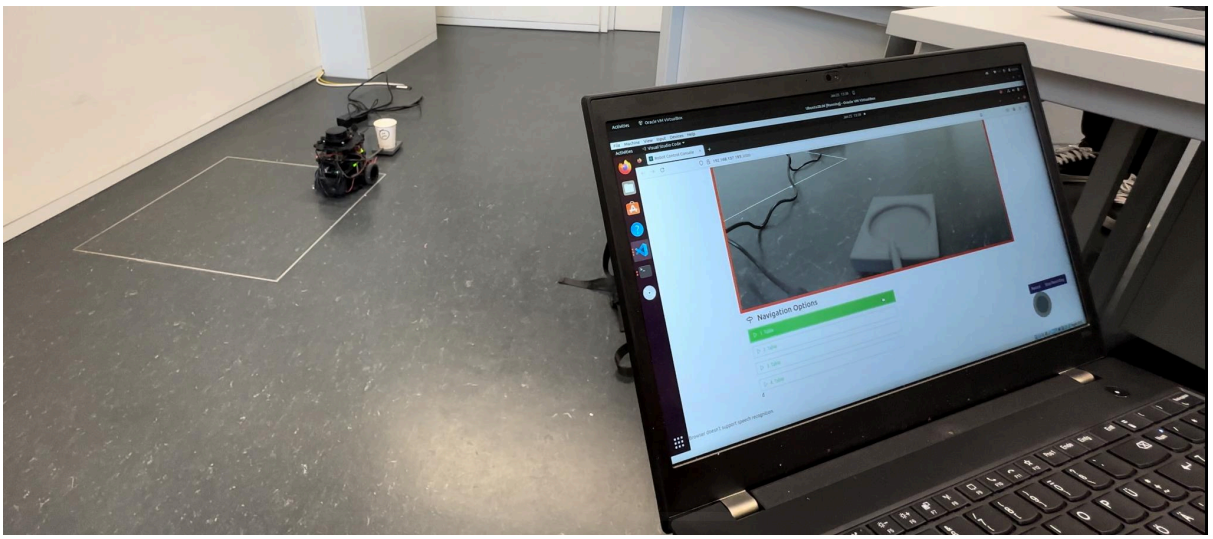
Unsere erste Herausforderung hierbei entstand, als wir das Raspberry Camera Module V3 verwenden wollten. Dieses kann zwar einfach unter Raspberry PI OS angesprochen werden, aber nicht so einfach unter Ubuntu. Aufgrund von Kompatibilitätsproblemen von ROS mussten wir Ubuntu als Operating System hernehmen. Diese Entscheidung führte dazu, dass wir eine USB-Kamera verwenden mussten. Diese kann mit OpenCV ausgelesen werden. Ein weiteres Problem war, dass die Objekterkennung auf dem Raspberry Pi zu viele Ressourcen benötigt und gleichzeitig mit ROS den Raspberry Pi überlasten würde. Aus diesem Grund haben wir uns dafür entschieden, die Objekterkennung in der Web-Anwendung über einen API-Call aufzurufen.

Arduino

Die erste Idee war die Motorsteuerung über eine Verbindung zwischen einem Arduino (Microcontroller) und einem auf derselben Plattform verbauten Raspberry PI zu realisieren. Dazu sollte der Arduino eine Schnittstelle bekommen, die es dem Pi ermöglicht, Kommandos für die Steuerung an den Arduino zu senden und dieser dann Sensordaten einer IMU (Rotation und Beschleunigung) zurücksendet um die gefahrene Strecke zu berechnen.. Zunächst wurde auch an diesem Konzept festgehalten und einige kleine Tests durchgeführt, die eine Machbarkeit zeigen sollten. Als wir aber einen LIDAR als Umgebungssensor nutzen wollten, fiel uns die fehlende Steuerung dieses Sensor auf die wir auch leider nicht dń der noch verbleibenden Zeit hätten selbst erstellen können. Aus diesem Grund haben wir uns entschieden die Plattform auf der wir unseren Roboter aufbauen wollen zu wechseln und benutzen von da an den Turtlebot3 der eben auch diese LIDAR-Steuerung bereits verbaut hatte.

Produktanpassungen nach Absprache

Aufgrund von Herausforderungen mit ROS und der unzureichenden Genauigkeit des Navigationssystems unter suboptimalen Bedingungen, haben wir beschlossen, die ursprünglich geplante Karten-Funktionalität zu entfernen. Stattdessen haben wir vorgegebene Pfade implementiert. Diese Pfade werden zunächst aufgezeichnet und können im Anschluss in der Web-App abgespielt werden, sodass Robi ausschließlich entlang dieser festgelegten Routen navigiert.


Navigation Options

Sprint Log


Workspace visible
Board
Power-Ups
Automation
Filters
Product Backlog (features)
Add a card
Sprint Backlog
Aufnahme von Wegen und abfahren der Aufzeichnungen
3D Druck des Tablets
Einrichten der Camera am Roboter
Verbindung Roboter Web App stabilisieren
Add a card
In Process
Erkennung der "Sache" auf dem Tapplet
Objekterkennung einrichten
Add a card
Done MS Report
Verbindung Raspberry zu Arduino testen
Frontend Designen
Add a card
Done MS Demo
Technical Analysis erstellen
Raspberry einrichten
Lidar Sensor ausbauen und ausprobieren
Add a card
Done Alpha
ROS installieren
Motorsteuerung Lidar
Web-App hosten
Daten / Befehle zwischen Arduino und Raspberry definieren und entwickeln
Funktionen des Roboters (Fahren, Stoppen...)
Roboter Scanner (Funktioniert es eine Karte aufzubauen?)
Object Detection auf Raspberry Pi
Gescannte Daten als Map aufbauen (Abhängig von den gesendeten Daten)
Add a card
Done Beta
Sprachbefehle geben
Mikrofon + Audio ein-/ausgabe
Web Applikation fertigstellen
Add a card
Risks
Wie kommunizieren alle Komponenten mit einnander
Spracherkennung
Navigation auf er Erstellten Map
Add a card
Impediments
Add a card

Abkürzungsverzeichnis

- ROS: Robot Operating System
- SBC: Single-Board-Computer
- API: Application Programming Interface
- SLAM: Simultaneous Localization and Mapping
- PWA: Progressive Web App
- VM: Virtual Machine
- LIDAR: Light detection and ranging