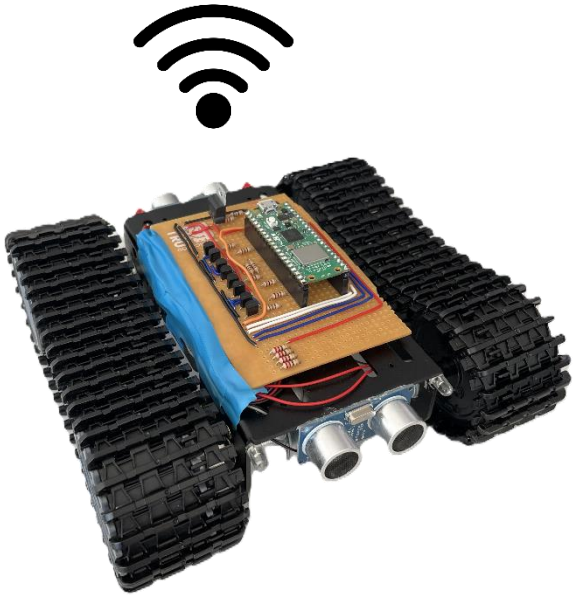
DHBW
Duale Hochschule
Baden-Württemberg

28. APRIL 2025



REMOTE-CONTROLE AUTO

SOFTWARE ENGINEERING

GRUPPE 5
JOHANNES WINTER(1937294), JACOB JÄGER(8790687), FELIX MAUNZ(5771729)

Inhalt

Abbildungsverzeichnis

1 Problem

In Zeiten zunehmender Digitalisierung und heranschreitender Technologien, gewinnen ferngesteuerte Systeme im Industriellen, aber auch in den privaten Bereichen immer mehr an Bedeutung. Für die Modernisierung und die Anpassung, an heutzutage zu Verfügung stehende Technologien, soll dieses Projekt eine Lösung sein. Auch der Aspekt der Nachhaltigkeit rückt zunehmend in den Fokus und spielt daher in diesem Projekt eine zentrale Rolle.

Vor diesem Hintergrund wurde im Rahmen dieses Projekts ein ferngesteuertes Auto entwickelt, das über ein Smartphone gesteuert werden kann und mithilfe von Ultraschallsensoren in der Lage ist, Hindernisse zu erkennen und entsprechend darauf zu reagieren. Ähnliche Entwicklungen finden insbesondere im Bereich privat genutzter Drohnen Anwendung, bei denen häufig das Smartphone in Kombination mit einer Fernsteuerung über Funkfrequenzen oder sogar ausschließlich das Smartphone genutzt wird.

Die zentrale Steuerung des Fahrzeuges bildet ein Microcontroller, der sowohl die Steuerung der Motoren als auch die Auswertung der Sensordaten und die Netzwerkkommunikation übernimmt. Die Verbindung zwischen Smartphone und Fahrzeug erfolgt über ein Netzwerk, in dem beide Geräte verbunden sein müssen. Die Benutzeroberfläche der Steuerung wurde als Android-App mit Kotlin und Java umgesetzt. Auf dem Microcontroller wurde die Software zur Motorsteuerung, Sensorauswertung und Netzwerkkommunikation in Python realisiert.

Zur Hinderniserkennung sind zwei HC-SR04-Ultraschallsensoren an der Front- und Rückseite des Fahrzeugs verbaut. Erkennt einer der Sensoren ein Objekt innerhalb eines definierten Abstandsbereichs, verhindert das System automatisch eine Weiterfahrt in diese Richtung und stoppt das Fahrzeug. Eine Kamera ist nicht integriert, so wie sie bei Drohnen oft der Fall ist, der Fokus liegt rein auf der auf der einfachen Handhabung, und der Kommunikation zwischen Steuerung und Fahrzeug.

Die Elektronik des Fahrzeugs basiert auf einer eigenen, mit Fritzing entworfenen Platine (siehe Schaltplan beigefügter Datei). Diese enthält unter anderem MOSFETs in H-Brücken-Schaltung zur bidirektionalen Motoransteuerung, diverse Widerstände sowie einen LDO-Spannungsregler zur stabilen Spannungsversorgung der Komponenten. Die Spannungsversorgung des Fahrzeugs wurde durch zwei Lithium-Ion Zellen realisiert, die genügend Energie für die Versorgung des Fahrzeugs zur Verfügung stellen.

Da „rund 94,85 Prozent der 14- bis 29-jährigen Personen in Deutschland ein Smartphone/Handy im Haushalt besitzen“ (Statista, 20.02.2025) [1], liegt es nahe, dieses Gerät zur Steuerung des Fahrzeugs zu verwenden. Durch die Nutzung eines bereits vorhandenen Geräts spart der potenzielle Kunde die Anschaffung zusätzlicher Hardware. Auch aus Herstellersicht ist dies ein Vorteil, da die Produktion eigener Fernsteuerungen entfällt und lediglich Kosten für die Softwareentwicklung anfallen – was wiederum der Nachhaltigkeit zugutekommt. Ein weiterer positiver Aspekt ist der integrierte

Notbremsassistent, der die Langlebigkeit des Fahrzeugs erhöht und potenzielle Schäden verhindert.

2 Anforderungen

2.1 Use-Case Diagramm

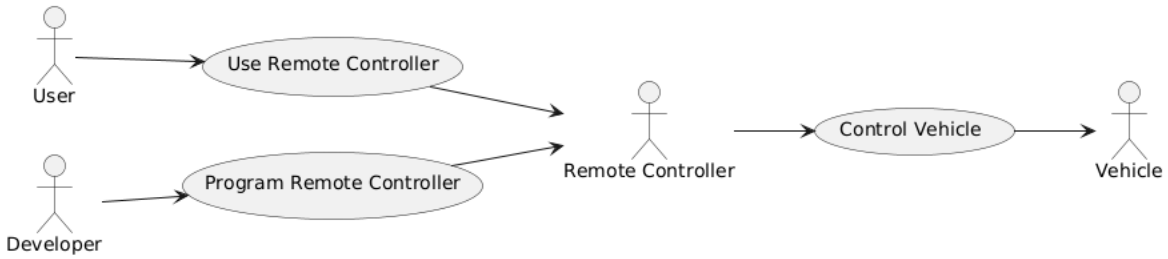


Abbildung 1: Use Case Diagramm

Um einen ersten Überblick darüber zu erhalten, wie das System funktioniert und welche Akteure daran beteiligt sind, eignet sich das Use-Case-Diagramm besonders gut.
Es zeigt die vier zentralen Akteure: User, Developer, Remote Controller und Vehicle.
Der User steuert das Fahrzeug über den Remote Controller, welcher zuvor vom Developer programmiert wurde.
Der Remote Controller wiederum stellt die Verbindung zum Vehicle her und überträgt die Steuerbefehle.
Auf die Details der Verbindungen und Interaktionen wird im weiteren Verlauf noch näher eingegangen.

2.2 Requirements

ID	Type	Test	VM	TestRef.	Status
1	FU	Das Fahrzeug soll remote gesteuert werden	T		PASS
2	IF	Richtung soll steuerbar sein	T	001	PASS
3	IF	Geschwindigkeit soll steuerbar sein	T		PASS
4	FU	Das Fahrzeug soll eine automatische Bremsung ausführen, wenn ein Hindernis erkannt wird	T		FAIL
5	FU	Das Fahrzeug soll einen sicheren Verbindungsaufbau zum Remote-Controller gewährleisten	A		PASS
6	IF	Das Fahrzeug soll Hinderniserkennung enthalten	T		FAIL
7	DE	Die Steuer-App soll mit gängigen Smartphone-Betriebssystemen (iOS und/oder Android) kompatibel sein	T		PASS
8	PF	Das Fahrzeug soll mit einer Maximalgeschwindigkeit von 8 km/h betrieben werden	T	002	PASS
10	OP	Steuerung soll schlicht aufgebaut sein	T		PASS
12	DE	Produkteinführung für das System soll 2 Monate nicht überschreiten	R		PASS
13	FU	Fahrzeug soll über einen Mobilen Akku betrieben werden	T		PASS
14	FU	Fahrzeug soll wieder aufladbar sein	T		PASS
15	FU	Fahrzeug soll im Dunkeln sichtbar sein	T		PASS
16	OP	Verbindungsstatus soll angezeigt werden	T		FAIL

Tabelle 1: Anforderungen

Abkürzung	Type
FU	Functional requirements
IF	Interface requirements
DE	Design constraints
PF	Performance requirements
OP	Operational requirements

VM	Verification Methods
T	Test
A	Analysis
R	Review

Die Tabelle zeigt die zu Projektbeginn definierten Anforderungen, die in verschiedene sogenannte Requirement-Typen unterteilt wurden. Im Folgenden werden diese kurz erläutert.
Die *Functional Requirements (FU)* beschreiben die grundlegenden Funktionen des Systems, also, was das System konkret leisten soll.
Interface Requirements (IF) beziehen sich auf die Schnittstellen des Systems, insbesondere auf die Interaktionen mit der Umgebung, mit anderen Objekten, Systemen oder mit dem Benutzer.
Design Constraints (DE) schränken die Gestaltungsmöglichkeiten während der Entwicklung gezielt ein, um beispielsweise unnötige Verzögerungen bei der Fertigstellung zu vermeiden (vgl. ID 12 in der Tabelle).
Die *Performance Requirements(PF)* konkretisieren die funktionalen Anforderungen, indem sie messbare Leistungsstandards und Qualitätskriterien definieren.
Die Operational Requirements(OP) schließlich stellen sicher, dass das System für die Nutzer möglichst einfach, effizient und weitgehendst problemlos zu bedienen ist.

Um die Anforderungen in der späteren Verifikations- und Validierungsphase überprüfen zu können, kommen unterschiedliche Verifikationsmethoden (VM) zum Einsatz.
In diesem Projekt lassen sich die meisten Anforderungen durch praktische Tests verifizieren. Allerdings können nicht alle Anforderungen auf diese Weise geprüft werden. So erfordert beispielsweise die Anforderung eines sicheren Verbindungsaufbaus eine genauere Analyse, etwa in Form einer Codeüberprüfung, da sie sich nicht allein durch einen einfachen Testfall validieren lässt.
Die letzte Spalte der Tabelle dient der Dokumentation und Nachverfolgung des aktuellen Status jeder Anforderung im Verlauf des Projekts. Zum jetzigen Zeitpunkt konnten sechs Anforderungen erfolgreich bestätigt werden.

3 Projektplan

Für eine strukturierte und effiziente Umsetzung des Projekts ist eine sorgfältige Planung unerlässlich. Diese umfasst die Definition konkreter Arbeitspakete, eine zeitliche Ablaufplanung in Form eines Gantt-Diagramms sowie eine realistische Kostenabschätzung.

3.1 Arbeitspaketstruktur

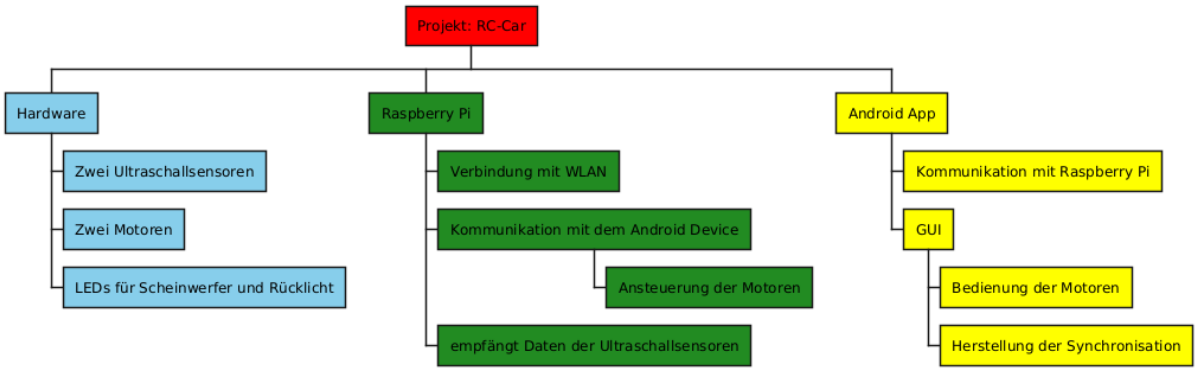


Abbildung 2: Arbeitspaketstruktur

Die Arbeitspaketstruktur dient der systematischen Aufgabenverteilung im Projekt. Die einzelnen Aufgabenbereiche werden dabei hierarchisch gegliedert und auf mehreren Ebenen immer weiter verfeinert, um eine klare und zielgerichtete Organisation zu ermöglichen.
In diesem Projekt lassen sich drei Hauptaufgabenbereiche identifizieren: Hardware, Raspberry Pi und die Android App.
Die Hardware unterteilt sich in drei wesentliche Arbeitspakete: Zum einen die Ultraschallsensoren, die zur Erkennung von Hindernissen an der Vorder- und Rückseite des Fahrzeugs eingesetzt werden, zum anderen die Motoren, die für die Fortbewegung und Lenkung zuständig sind, sowie die LEDs, welche als Scheinwerfer und Rücklichter dienen.

Der zweite Bereich, der sich mit dem Raspberry Pi beschäftigt, ist ebenfalls in drei Arbeitspakete untergliedert. Dazu zählt zunächst die Verbindung mit dem WLAN, um eine Verbindung zwischen dem Raspberry Pi und anderen Komponenten herzustellen. Ein weiteres Arbeitspaket umfasst die Kommunikation mit dem Android-Gerät und dadurch auch die Ausführung der Steuerbefehle, die der Benutzer über die App an das Fahrzeug sendet. Ergänzt wird dieser Bereich durch eine Funktion, die die empfangenen Messdaten der Ultraschallsensoren ausliest und zur weiteren Verarbeitung bereitstellt.

Als letzter Planungsschritt folgt die Entwicklung der Android App. Auch in diesem Bereich spielt die Kommunikation eine zentrale Rolle – insbesondere die Verbindung zum Raspberry Pi. Ein wesentliches Arbeitspaket innerhalb dieses Aufgabenbereichs ist die Gestaltung der grafischen Benutzeroberfläche (GUI). Sie ermöglicht es dem Benutzer, die Synchronisation mit dem Mikrocontroller herzustellen und die Steuerung der Motoren über die App zu bedienen.
Die hier dargestellte Arbeitspaketstruktur bildet die Grundlage für die nachfolgende zeitliche Projektplanung in Form eines Gantt-Diagramms sowie für die Abschätzung der anfallenden Kosten im weiteren Verlauf.

3.2 Gantt-Diagramm

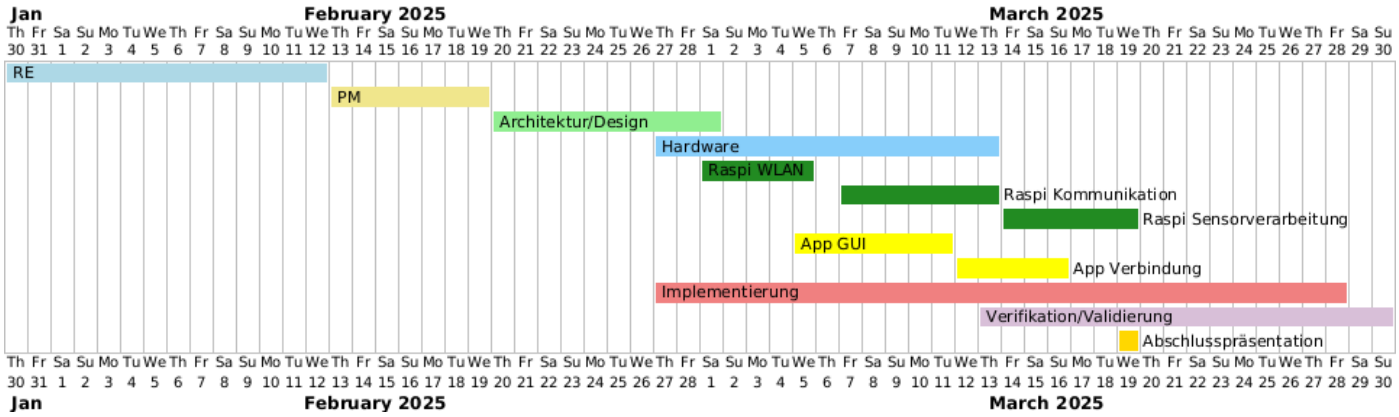


Abbildung 3: Gantt-Diagramm

Das dargestellte Gantt-Diagramm visualisiert die zeitliche Abfolge der einzelnen Projektplanungsschritte in Form von Balken. Um die verschiedenen Phasen klar voneinander abzugrenzen, wurden diese farblich unterschiedlich hervorgehoben.
Zu Beginn sind die ersten beiden Phasen, Requirement Engineering und Projektmanagement noch deutlich voneinander getrennt dargestellt. Im weiteren Verlauf kommt es jedoch zu Überschneidungen einzelner Phasen. So wurde mit der Implementierung bereits begonnen, bevor die Phase der Architektur vollständig abgeschlossen war.
Die Implementierungsphase selbst ist in drei Teilbereiche untergliedert, basierend auf der definierten Arbeitspaketstruktur des letzten Unterkapitels. Die anschließende Phase der Verifikation/Validierung setzt bereits während der laufenden Implementierung ein, da erste Komponenten (z. B. Raspi-Kommunikation) bereits abgeschlossen und überprüfbar sind.
Am Ende des Projektzeitraums wurde eine Präsentation des Zwischenstands gehalten, da das Projekt nicht vollständig abgeschlossen werden konnte. Einige Arbeitspakete blieben in Bearbeitung oder wurden nur teilweise umgesetzt.

3.3 Kosten

Um die Kostenschätzung des Projekts zu berechnen, werden bestimmte Formeln angewendet.

Unadjusted Function Point Count (UFPC):

$$ufpc = \sum (NumRequirementsType * Weight\ [2])$$

Nun eingesetzt:

$$ufpc\ = (6 \cdot 5) + (3 \cdot 4) + (2 \cdot 3) + (1 \cdot 4) + (2 \cdot 3) = 58$$

Jetzt berechnet man AdjustedFunctionPointCount (AFPC):

$$afpc = (GlobalFactor * ufpc) * (\left(1 + \frac{ChangedReq}{100}\right) * 0{,}3)$$

Eingesetzt für Global Factor = 1,1 und Changed Req = 10%:

$$afpc = (1.1 \cdot 58) \cdot ((1 + \frac{10}{100}) \cdot 0.3) = 21.05$$

Daraus die Source Lines of Codes (SLOC):

$$sloc = afpc * LanguageFactor$$

➔ $sloc = 21.05 \cdot 12 = 252.6 \approx 253$

Kostenabschätzung:

Gesamtzeit: 253/10 = 25,3h
Stundensatz: 10 €
Entwicklungskosten: 253€
Produktivität: 10 SLOC/h

4 Architektur und Verhalten

4.1 Top-Level Diagramm

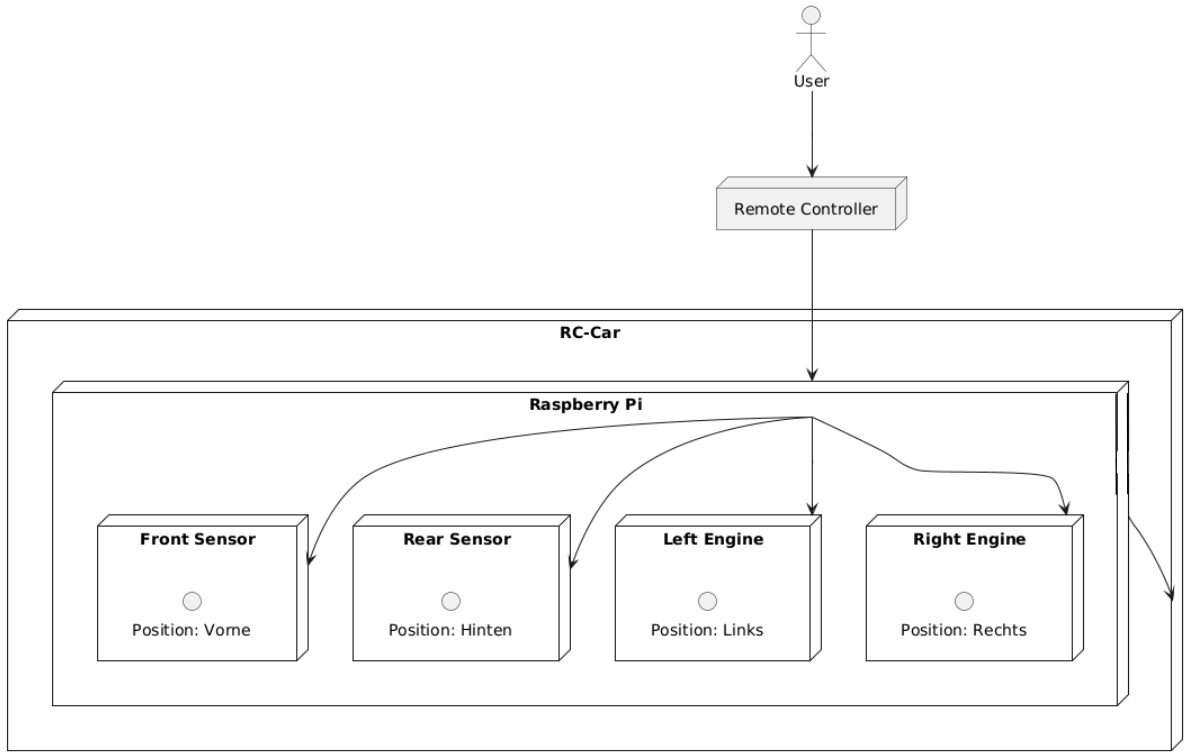


Abbildung 4: Top-Level Diagramm

Ziel des Top-Level-Diagramms ist es, die Zusammenhänge und Wechselwirkungen zwischen den einzelnen Systemkomponenten klar und übersichtlich darzustellen. Genau diese Funktion übernimmt das hier präsentierte Diagramm. Im Folgenden wird die dargestellte Systemkonstellation näher erläutert.
Der Benutzer erhält über den Remote Controller, in diesem Fall eine App, Zugriff auf den Raspberry Pi und stellt somit eine drahtlose Verbindung zum Fahrzeug her. Auf dem Mikrocontroller, der direkt auf dem Fahrzeug bzw. RC-Car verbaut ist, laufen mehrere miteinander verknüpfte Prozesse ab. Wie bereits in der Arbeitspaketstruktur beschrieben, gehören dazu die Ansteuerung der beiden seitlich angebrachten Motoren sowie der Empfang der Messdaten von zwei Ultraschallsensoren, die jeweils an der Vorder- und Rückseite des Fahrzeugs montiert sind.

4.2 Sequenz-Diagramm

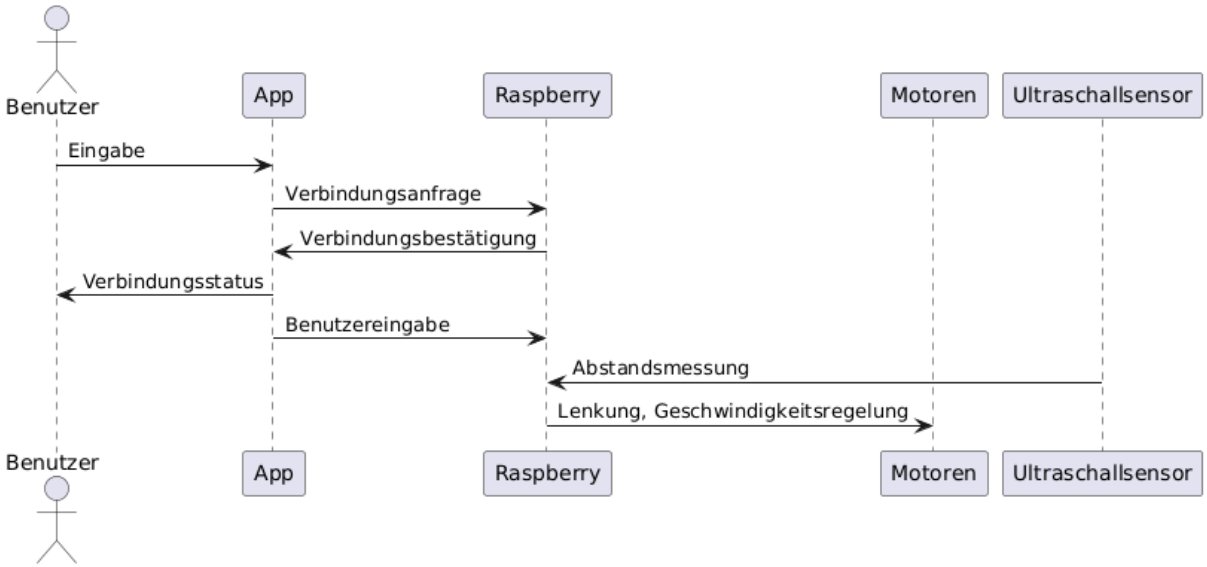


Abbildung 5: Sequenz-Diagramm

Neben dem Top-Level-Diagramm eignet sich insbesondere das Sequenzdiagramm, um die zeitliche Abfolge und Interaktion der Systemkomponenten präzise darzustellen. Es veranschaulicht die Kommunikation zwischen Benutzer, App, Mikrocontroller und weiteren Fahrzeugkomponenten.
Der Ablauf beginnt mit einer Eingabe durch den Benutzer über Joystick und Buttons innerhalb der Steuer-App. Diese sendet anschließend eine Verbindungsanfrage an den Raspberry Pi, welcher als zentraler Mikrocontroller im Fahrzeug fungiert. Nach erfolgreicher Verbindungsherstellung bestätigt der Raspberry Pi die Anfrage, woraufhin die App dem Benutzer den aktuellen Verbindungsstatus anzeigt.
Im weiteren Verlauf übermittelt die App Steuerbefehle an den Raspberry Pi, welcher die Lenkung und Geschwindigkeit der beiden Motoren entsprechend anpasst. Zeitgleich empfängt der Mikrocontroller kontinuierlich Entfernungsdaten von den an Vorder- und Rückseite installierten Ultraschallsensoren und verarbeitet diese zur Hinderniserkennung. Das Sequenzdiagramm macht deutlich, dass eine bidirektionale Kommunikation zwischen allen beteiligten Komponenten besteht.

5 Implementierung

5.1 Implementierung App

Bei der Auswahl von Programmiersprachen für die Android-Entwicklung stehen mehrere Optionen zur Verfügung, darunter Kotlin, C++, Flutter und andere. Allerdings sind Kotlin und Java die am häufigsten verwendeten Sprachen, insbesondere in der Android Studio IDE. Kotlin hat sich in den letzten Jahren als die bevorzugte Sprache für die Android-Entwicklung etabliert, da sie moderne Sprachmerkmale bietet, die die Entwicklung effizienter und weniger fehleranfällig machen. Sie ist vollständig interoperabel mit Java, was bedeutet, dass bestehende Java-Bibliotheken und -Frameworks problemlos genutzt werden können. Java hingegen hat eine lange Geschichte in der Android-Entwicklung und bietet eine breite Basis an Ressourcen, Tutorials und Community-Support. Die Android Studio IDE ist speziell für die Entwicklung mit diesen beiden Sprachen optimiert, bietet umfassende Tools wie Code-Vervollständigung, Debugging und Refactoring, die den Entwicklungsprozess erheblich erleichtern.

5.1.1 Backend

Die Verwendung des Singleton-Designmusters für eine TCP-Socket-Klasse ist sinnvoll, da es sicherstellt, dass nur eine einzige Instanz der Socket-Verbindung während der gesamten Lebensdauer der Anwendung existiert. Dies verhindert Probleme wie Mehrfachverbindungen und Ressourcenverschwendung, da die Verwaltung von Netzwerkressourcen zentralisiert wird. In Kotlin kann das Singleton-Muster elegant mit dem object-Schlüsselwort implementiert werden, das eine Klasse definiert, deren Instanz automatisch erstellt wird und global zugänglich ist. Die TCP Socket Klasse besitzt die Methoden connect, um sich mit dem Raspberry zu verbinden, sendMessage, um Nachrichten an den Raspberry zu schicken und disconnect, um sich wieder zu entkoppeln. (RaspberryConnection.kt und ConnectionManager.kt)

Die CustomWheel-Klasse ist eine benutzerdefinierte Ansicht in Android, die ein interaktives Rad darstellt, das auf Berührungen reagiert. Sie ermöglicht es Benutzern, den Winkel des Rades durch Wischbewegungen zu ändern, wobei der aktuelle Winkel zwischen 20 und 160 Grad begrenzt ist. Die Klasse verwendet die onDraw-Methode, um das Rad und die Indikatorlinien zu zeichnen, die den aktuellen Winkel anzeigen. Durch die Implementierung des onTouchEvent
-Handlers wird die Benutzerinteraktion ermöglicht, indem der Winkel basierend auf der Berührungsposition berechnet und aktualisiert wird. Der Listener on WheelChange wird verwendet, um dann in der MainActivity Klasse eine Nachricht an den Raspberry zu schicken. (CustomWheelView.kt)

5.1.2 Frontend

Die Struktur der UI wird in XML-Dateien festgelegt. In der activity_main.xml wird ein Container festgelegt, der das Rad und eine vertikalen Schieberegler zur Geschwindigkeitssteuerung enthält. Der andere Container ist für die Navigation zwischen Verbindungsverwaltung und Steuerung des RC-Autos. In der activity_connection.xml gibt es auch zwei Container. Das eine enthält ein Textfeld, um die IP-Adresse des Raspberry einzugeben. Der andere enthält wieder die Buttons zur Navigation in der App.

Was bei Eingaben des Nutzers jeweils passieren soll, wird in der MainActivity.kt und der ConnectionActivity.java festgelegt. Wenn das Rad oder der Schieberegler verändert wird und die Änderung signifikant ist, dann wird eine Nachricht über den TCP-Socket gesendet. Wenn eine Eingabe in das Textfeld erfolgt und diese bestätigt wird, dann wird versucht eine Verbindung herzustellen.

5.2 Implementierung Python Code

In der main.py gibt man die SSID und das Passwort des Netzwerks ein, mit dem man sich verbinden will. Dann wird eine endlose Schleife aufgerufen, um auf dem Port 1050 nach TCP-Verbindungen zu hören.

In der wifi_check.py wird mithilfe der python network Bibliothek versucht, eine Verbindung zu dem Netzwerk herzustellen.

In der Scoket.py Datei wird zuerst auf eine Verbindungsanfrage auf dem Port 1050 gewartet. Bei einer erfolgreichen Verbindung werden die Nachrichten in maximalen Paketgrößen von 1024 Byte empfangen und die SplitMessage Methode wird aufgerufen.

In der SpitMessage Methode wird zuerst die Nachricht getrennt, da sie einen Winkel und einen Wert für den Schieberegler enthält. Jetzt werden basierend auf diesen Werten die einzelnen Pins angesteuert, um das Fahrzeug fahren zu lassen.

5.3 Kodierungsrichtlinien

Kodierungsrichtlinien sind entscheidend für die Lesbarkeit und Wartbarkeit von Code. Ein zentrales Element dieser Richtlinien ist die Namenskonvention für Klassen, die stets mit einem Großbuchstaben beginnt, um sie klar von anderen Elementen zu unterscheiden. Im Gegensatz dazu werden Objekte, Variablen und Funktionsnamen in Kleinbuchstaben geschrieben, was eine einheitliche und verständliche Struktur fördert. Darüber hinaus sollten zusammengesetzte Wörter in Variablen durch Großschreibung des ersten Buchstabens jedes nachfolgenden Wortes hervorgehoben werden, wie beispielsweise bei der Variable *currentAngle*. Diese Konventionen tragen dazu bei, den Code übersichtlich zu gestalten und die Zusammenarbeit im Team zu erleichtern.

5.4 Agiles Arbeiten mit Jira

1. Zuweisung eines Jira-Tasks

Der erste Schritt des Arbeitsprozesses beginnt mit der Zuweisung eines Jira-Tasks. In Jira wird ein Task erstellt, der eine spezifische Aufgabe oder ein Feature beschreibt, das entwickelt werden soll. Teammitglieder können diese Aufgaben durch das Zuweisen an sich selbst oder an andere Teammitglieder übernehmen. Die Zuweisung erfolgt in der Regel durch einen Projektmanager oder Teamleiter, der die Verantwortlichkeiten klar definiert. Der Task enthält wichtige Informationen wie die Beschreibung, Priorität, Fälligkeit und eventuell Anhänge, die dem Entwickler helfen, die Anforderungen besser zu verstehen.

2. Erstellung eines neuen Zweiges

Sobald ein Task zugewiesen wurde, ist der nächste Schritt die Erstellung eines neuen Zweiges in Git. Hier verwenden wir die von Atlassin entwickelte UI Sourcetree. Dadurch können neue Features implementiert werden, ohne die Funktionalität des aktuellen Standes einzuschränken.

3. Zusammenführung des Zweiges

Sobald einer der Teammitglieder ein Feature fertig erstellt hat, erstellt er einen Pull-Request. Mindestens ein anderes Mitglied schaut sich diesen an und gibt Feedback zum Code. Wenn es keine offenen Punkte mehr gibt, dann wird der neue Zweig mit dem main Zweig zusammengeführt und der dazugehörige Jira Task wird geschlossen.

6 Test

DHBW
Duale Hochschule
Baden-Württemberg

Test-Protokoll (Interface Requirements)

Testprotokoll zur Verifizierung von Interface Requirements.

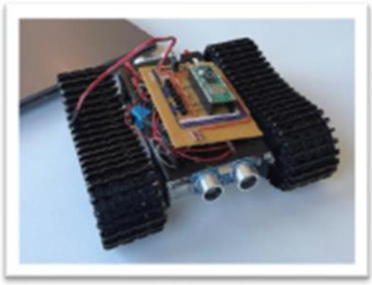
Projekt: RC-Auto	Datum: 27.04.2025
TestRef. : 001	Version: 1.0
Requirement: Richtung soll steuerbar sein	

Test Procedere:

Fahrzeug wird auf einer ebenen Fläche platziert und mit dem Remote Controller verbunden.
Die zu testende Manöver sind:

- Fahrende Linkskurve in vorwärts Richtung
- Fahrende Rechtskurve in vorwärts Richtung
- Fahrende Linkskurve in rückwärts Richtung
- Fahrende Rechtskurve in rückwärts Richtung
- Linksdrehung auf der Stelle
- Rechtsdrehung auf der Stelle

Bilder Testaufbau:



Testergebnis:

Fahrzeug hat folgende oben aufgelisteten Manöver durchführen können.

- Fahrende Linkskurve in vorwärts Richtung PASS
- Fahrende Rechtskurve in vorwärts Richtung PASS
- Fahrende Linkskurve in rückwärts Richtung PASS
- Fahrende Rechtskurve in rückwärts Richtung PASS
- Linksdrehung auf der Stelle PASS
- Rechtsdrehung auf der Stelle PASS

Zur Verifizierung des Testes wurde der Test fünf Mal erprobt, um die Zuverlässigkeit sicherzustellen. Anwesend waren zwei Entwickler einer zum Steuern des Fahrzeugs, der andere, um ausschließlich die vom Fahrzeug durchgeführten Bewegungen zu beobachten.

DHBW
Duale Hochschule
Baden-Württemberg

Test-Protokoll (Performance Requirements)

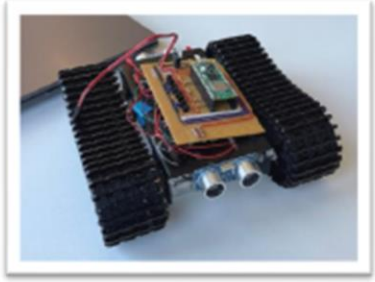
Testprotokoll zur Verifizierung von Performance Requirements.

Projekt: RC-Auto	Datum: 27.04.2025
TestRef. : 002	Version: 1.0
Requirement: Das Fahrzeug soll mit einer Maximalgeschwindigkeit von 8 km/h betrieben werden	

Test Procedere:

Fahrzeug wird auf einer ebenen Fläche platziert und mit dem Remote Controller verbunden. Auf der Oberfläche werden zwei parallele Linien mit einem Abstand von 5m aufgebracht. Das Fahrzeug startet an der ersten Linie und es wird die Zeit bis zum Erreichen der zweiten Linie gemessen. Danach wird mit der Formel $v = \frac{s}{t}$ $[m/s]$ die Geschwindigkeit berechnet.

Bilder Testaufbau:



Testergebnis:

Oben geschilderte Test wurde vier Mal durchgeführt, die Messergebnisse waren:

1. t = 3,84s -> 2,08m/s -> 7,45km/h -> PASS
2. t = 3,92s -> 2,04m/s -> 7,34km/h -> PASS
3. t = 3,90s -> 2,05m/s -> 7,38km/h -> PASS
4. t = 3,87s -> 2,07m/s -> 7,45km/h -> PASS

Die Umrechnung von m/s in km/h erfolgte über die Website:
https://www.omnicalculator.com/de/umrechnungen/ms-in-kmh-umrechner

7 Installationsanleitung

Um die App herunterladen zu können muss man die Installation aus unbekannten Quellen zulassen. Danach schaut man unter folgendem Link: https://github.com/Baljet389/SE_Car unter dem Ordner release v1.0. Dort befindet sich eine APK-Datei, die entweder auf Android Geräten oder auf Windows 11 Systemen installiert werden kann.
Um die Python Dateien auf den Raspberry Pi Pico zu laden, kann man entweder das Repository klonen oder manuell alle Python Dateien aus dem Ordner Raspberry herunterladen. Diese Dateien können dann mit einer IDE wie z.B. Thonny direkt auf den Microcontroller geladen werden.
Das Verteilungsdiagramm (Abb. 6) stellt die physische Verteilung der Softwarekomponenten und deren Interaktionen in einem Projekt dar, bei dem ein RC-Auto von einem Raspberry Pi über eine Android-App gesteuert wird. In diesem Diagramm ist das Android-Gerät, auf dem die App läuft, als AndroidDevice dargestellt, das über einen TCP-Socket mit dem RaspberryPi kommuniziert. Der Raspberry Pi empfängt die Steuerbefehle über den TCP-Server und leitet diese an die Steuerungssoftware weiter. Diese Software steuert dann das RC-Car durch die Motortreiber.

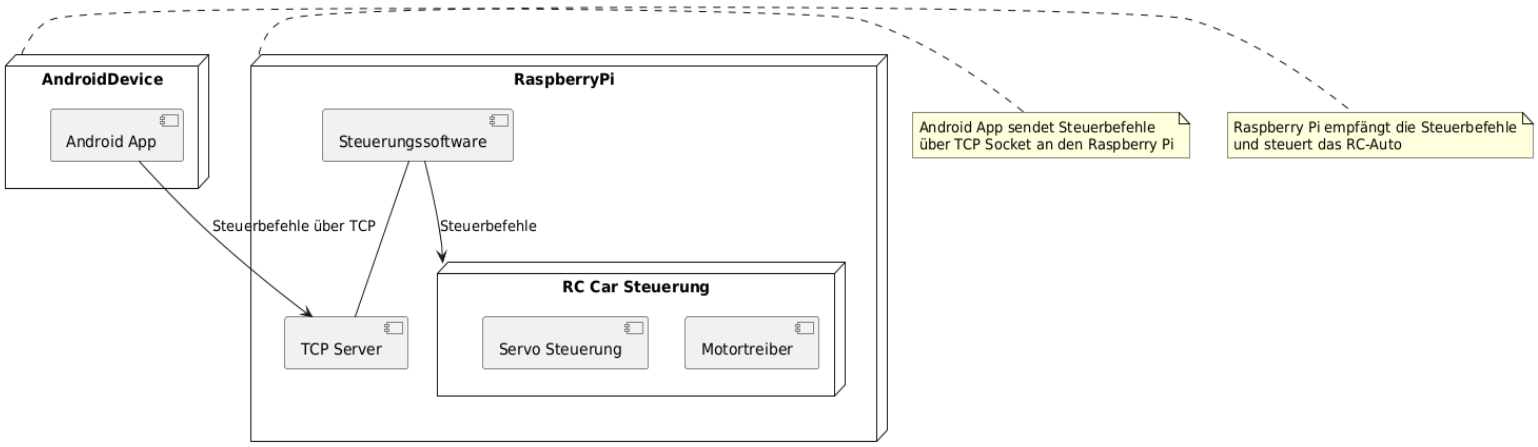


Abbildung 6: Verteilungsdiagramm

8 Bedienung

Als erstes muss der Raspberry pi pico mit einem PC verbunden werden und in der main.py die SSID und das Passwort auf das Netzwerk gesetzt werden, womit auch das Handy (oder PC) verbunden ist. Dann muss die main.py gestartet werden. Bei einer erfolgreichen Verbindung mit dem Netzwerk wird im Terminal die IP- Adresse des Raspberry angezeigt. Diese muss man dann in der App unter Connection eingeben und bestätigen. Eine erfolgreiche Verbindung wird im Terminal angezeigt und man kann jetzt den Raspberry abkoppeln und das Fahrzeug mit der App steuern. Wenn das Passwort einmal gesetzt ist, wir bei einer Stromversorgung die main.py automatisch ausgeführt und die IP-Adresse bleibt meistens auch gleich. So kann man dann direkt loslegen.

9 Fazit und Ausblick

9.1 Vergleich geschätzte und reale Komplexität

In diesem Kapitel werden die Kostenschätzungen und realen Kosten verglichen. Außerdem werden die Probleme, die im Laufe des Projekts geschehen sind, aufgegriffen.

Im Zusammenhang mit den Projektkosten werden die Source Lines of Code (SLOC) betrachtet. In Kapitel 3 wurde eine erste Schätzung vorgenommen, die auf 252,6 Zeilen Code kam. Der aktuelle Stand liegt jedoch bei etwa 430 Codezeilen, was eine Abweichung von rund 177 zusätzlichen Zeilen bedeutet.
Um dieser Entwicklung Rechnung zu tragen, muss der Global Factor entsprechend angepasst werden.

- ➔ $afpc = \frac{430}{12} = 35{,}83$
- ➔ $Global\ Factor = \frac{35{,}83}{\left(1+\frac{10}{100}\right)*0{,}3*58} = 1{,}872$

Daraus ergibt sich gemäß der in Kapitel "Kosten" vorgestellten Formel ein neuer Global Factor von 1,872.
Bei einer Produktivität von 10 SLOC pro Stunde beträgt die Gesamtzeit 430 SLOC / 10 SLOC/h = 43 Stunden. Daraus ergeben sich Gesamtkosten von 43 Stunden * 10 € = 430 €, basierend auf einem Stundensatz von 10 €.

Im Verlauf des Projekts traten mehrere Probleme auf. Zum einen funktionierte die WLAN-Verbindung mit dem Raspberry Pi zunächst nicht, da er sich nicht einloggen konnte. Zum anderen wurden anfänglich die falschen Transistoren verwendet, was dazu führte, dass nicht genügend Strom an die Basis gelangte und die Transistoren somit nicht durchschalteten.

9.2 Erweiterungen

9.2.1 Software

Eine mögliche Erweiterung der Android-App könnte die Implementierung einer umfassenden Benutzeroberfläche zur Anzeige wichtiger Fahrzeugdaten umfassen. Diese Erweiterung könnte eine visuelle Anzeige des Verbindungsstatus beinhalten, die dem Benutzer in Echtzeit anzeigt, ob die Verbindung zum Fahrzeug stabil ist oder ob es Verbindungsprobleme gibt. Darüber hinaus könnte die App Informationen zur aktuellen Geschwindigkeit des Fahrzeugs bereitstellen, um dem Benutzer ein besseres Gefühl für die Fahrzeugdynamik zu geben. Eine weitere nützliche Funktion wäre die Anzeige der Distanz zum nächsten Hindernis, die durch Sensoren im Fahrzeug ermittelt wird. Diese Informationen könnten in Form von Grafiken oder Warnmeldungen dargestellt werden, um die Sicherheit und Benutzerfreundlichkeit der App zu erhöhen und dem Fahrer zu helfen, informierte Entscheidungen während der Fernsteuerung zu treffen.
Eine mögliche Erweiterung des Python-Programms könnte das Programm so erweitert werden, dass es die Daten der Ultraschallsensoren in Echtzeit an ein verbundenes Smartphone zurücksendet, sodass der Benutzer jederzeit Informationen über die Umgebung des Fahrzeugs erhält, wie z.B. die Entfernung zu Hindernissen. Dadurch würde man nicht nur

die Steuerung des Fahrzeugs optimieren, sondern auch die Interaktivität und Benutzerfreundlichkeit des Systems erheblich steigern, indem sie dem Benutzer wertvolle Informationen zur Verfügung stellt und ihm mehr Kontrolle über die Fahrzeugbewegungen gibt.

Literaturverzeichnis

[1] Statista, „Statista,“ 20 02 2025. [Online]. Available: https://de.statista.com/themen/6137/smartphone-nutzung-in-deutschland/#topicOverview. [Zugriff am 26 04 2025].

[2] I. 20926, „it-gost,“ 1 10 2003. [Online]. Available: https://it-gost.ru/images/files/ISO_20926.pdf. [Zugriff am 22 April 2025].

[3] „ChatGPT,“ OpenAI, [Online]. Available: https://chatgpt.com/.

[4] „PlantUML,“ [Online]. Available: https://plantuml.com/.