



Dokumentation

Projekt Stanley



Projektmitglieder:

Sofie Rücker

Florian Gehler

Max Grundmann

Projektpartner:

Evelyn Cimander

Lukas Becker

Datum:

18.08.2022



Inhalt

1. Projektaufgabe	3
1.1. Ziel	3
1.2. Idee	3
1.3. Projektplan	4
2. Hardware	5
2.1. Anforderungen an die Hardware & finale Auswahl	5
2.2. PiCar-V	5
Limitationen	7
2.3. Raspberry Pi	7
Limitationen	8
3. Code	9
3.1. Pi Car Server	9
3.2. main Methode	10
3.3. Stoppschild Erkennung	10
3.4. Linienerkennung	14
3.5. Herausforderungen	15
3.6. 4G & 5G Modus	15
3.7. Installation	16
3.8. Inbetriebnahme	17
3.9. Steuerung	20
3.10. Bekannte Probleme	21
3.10.1. Nicht erkannte Kamera	21
3.10.2. IP wird nicht erkannt	21
3.11. Laufzeiten	21
4. Abrechnungsmodelle	22
4.1. Lokale Anwendungen	22
4.2. Weitere Modelle für Endnutzer	22



1. Projektaufgabe

Die WISTA Management GmbH möchte für Ihren 5G Showroom ein Ausstellungsstück angeboten bekommen, das die Vorteile von 5G und damit ihrem Technologiepark in Adlershof präsentiert. Zudem sollen Ansätze für ein Abrechnungsmodell für 5G entwickelt werden.

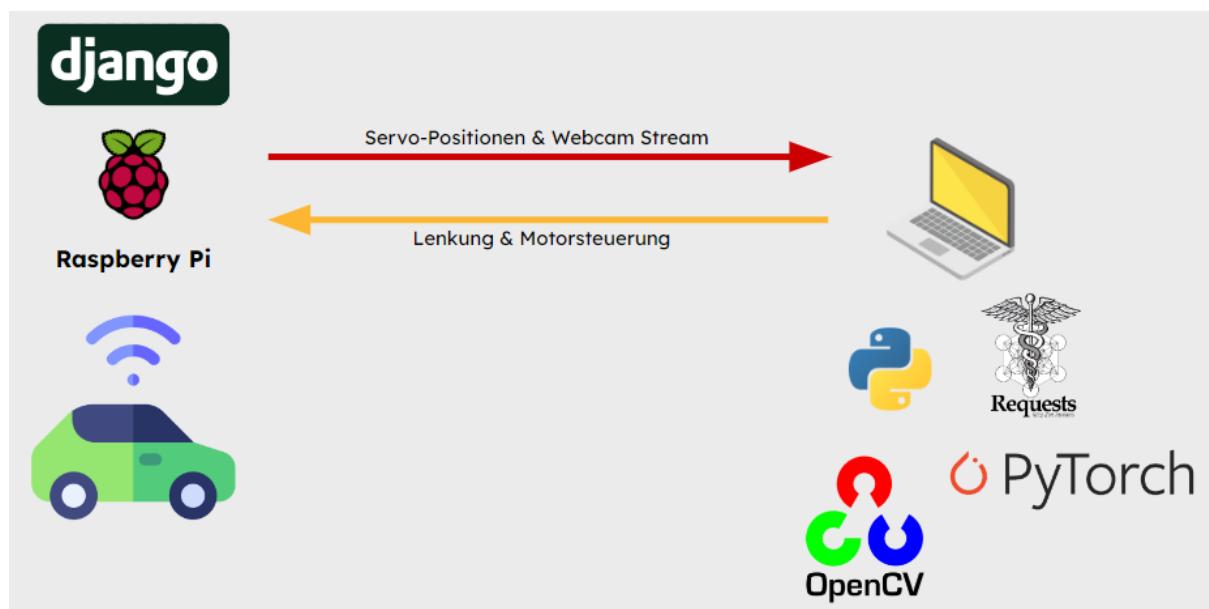
1.1. Ziel

Hauptziel dieses Projekts ist die Visualisierung der Latenz von 5G. Besonders in Hinsicht auf den Unterschied zu 4G soll ein visueller Vergleich dargestellt werden.

1.2. Idee

Die Kernidee zu dem Ausstellungsstück entwickelte sich in die Richtung autonomes Fahren. Dazu sollte ein ferngesteuertes Modellauto mit Kamera, einem Raspberry Pi und einem 5G Modul zum Einsatz kommen. Das Auto soll eine vordefinierte Route fahren, die durch eine Linie markiert wird, der das Auto mittels Kamera folgt. Auf der Strecke werden zwei bis drei Stoppschilder platziert, bei denen das Auto auch wieder mit Hilfe der Kamera und Bilderkennung halten soll.

Im Unterschied zu "echten" autonomen Autos soll dabei jedoch die gesamte Steuerungslogik (Bilderkennung etc.) nicht auf dem Auto selbst, sondern auf einem externen Rechner laufen. Mit diesem Versuchsaufbau wollen wir anschließend die Unterschiede in der Latenz zwischen 4G und 5G Netzwerken zeigen. Um das Ganze noch interaktiver zu gestalten, bekommt der Nutzer zudem die Möglichkeit, zwischen einem 4G und 5G "Modus" hin und her zu schalten. Im 5G Modus soll das Auto rechtzeitig vor dem Stoppschild stehen bleiben. Bei 4G soll es die höhere Latenz verdeutlichen und nicht rechtzeitig zum Stehen kommen.



1.3. Projektplan

In Absprache mit der WISTA AG haben wir uns bei der Durchführung dieses Projekts bewusst nur grob an unserem anfangs aufgestellten Projektplan orientiert. Grund dafür war, dass wir mit diesem Thema viele neue, uns unbekannte Technologien und Konzepte ausprobieren wollten und auch Unsicherheitsfaktoren wie die Beschaffung der Hardware und deren Inbetriebnahme die Planung erschwerten. Der nachfolgende Projektplan ist die in Absprache angepasste Variante und bildet nun die tatsächlich benötigten Aufwendungen der einzelnen Projektphasen ab.

Aufgabe	Start	Tage	Mai	Juni	Juli	August
Elarbeitung in die Aufgabenstellung	03.05.2022	23				
Kick-Off Meeting		1				
Entwickeln von Ideen zum Ausstellungsstück		12				
Erstellen des Projektplans		3				
Recherche zum Auto/Pi/5G Modul		8				
Beschaffung der Hardware	26.05.2022	2				
Bestellen PiCar-V		2				
Bestellen Raspberry Pi		2				
Inbetriebnahme	28.05.2022	10				
Zusammenbauen PiCar-V		3				
Installation Raspberry Pi		3				
Testen der vorhandenen Software		6				
Entwicklung der Software	07.06.2022	48				
Entwicklung der Stoppschilderkennung		19				
Entwicklung der Liniererkennung		15				
Fertigstellung der Funktionen		17				
Projektabschluss	25.07.2022	24				
Fertigstellen der Dokumentation		14				
Vorbereiten der Abschlusspräsentation		9				
Live Präsentation der Ergebnisse		1				

2. Hardware

Im Folgenden erklären wir, mit welche Hardware wir in diesem Projekt verwendet haben, wie wir zu dieser Auswahl kamen und beschreiben anschließend grob den Aufbau der verwendeten Komponenten.

2.1. Anforderungen an die Hardware & finale Auswahl

Während die Auswahl von steuerbaren und programmierbaren Modellfahrzeugen zunächst groß erscheint, stellte sich schnell heraus, dass tatsächlich nur wenige fertige Kits unseren Ansprüchen genügen. Dazu zählten unter anderem folgende Anforderungen:

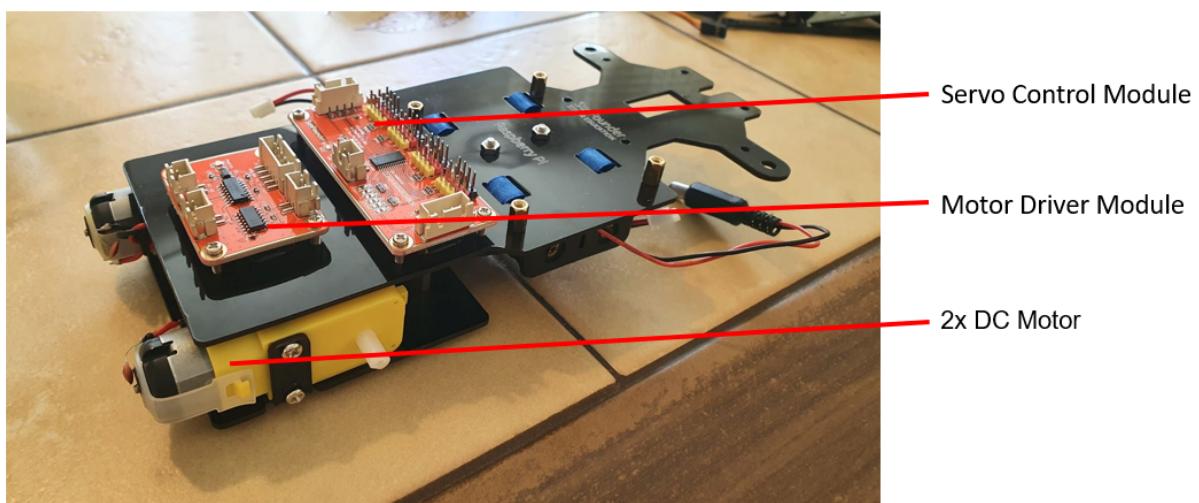
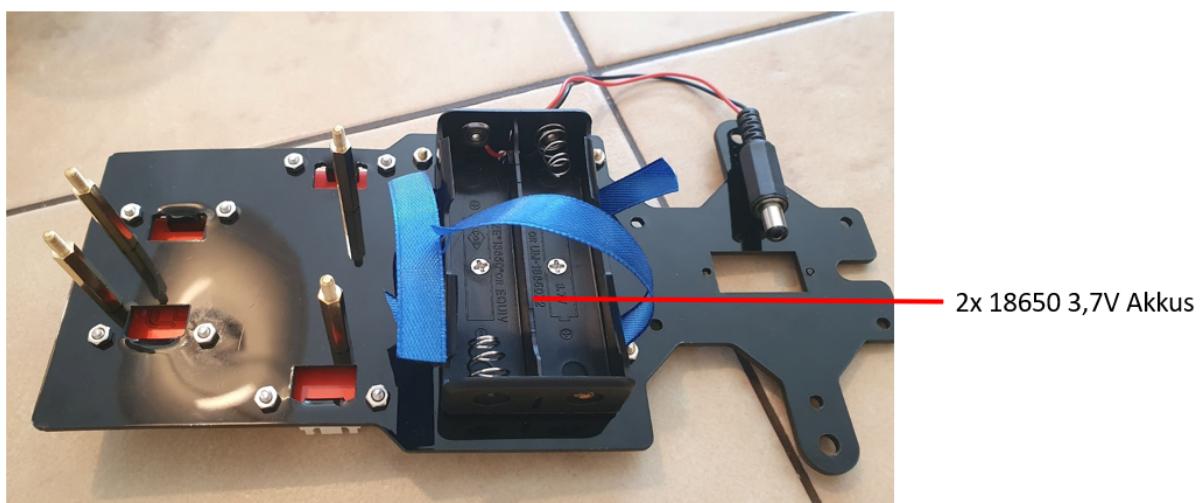
- Stufenlose Lenkung
- Frei zugängliche und dokumentierte Schnittstellen
- Erweiterbar

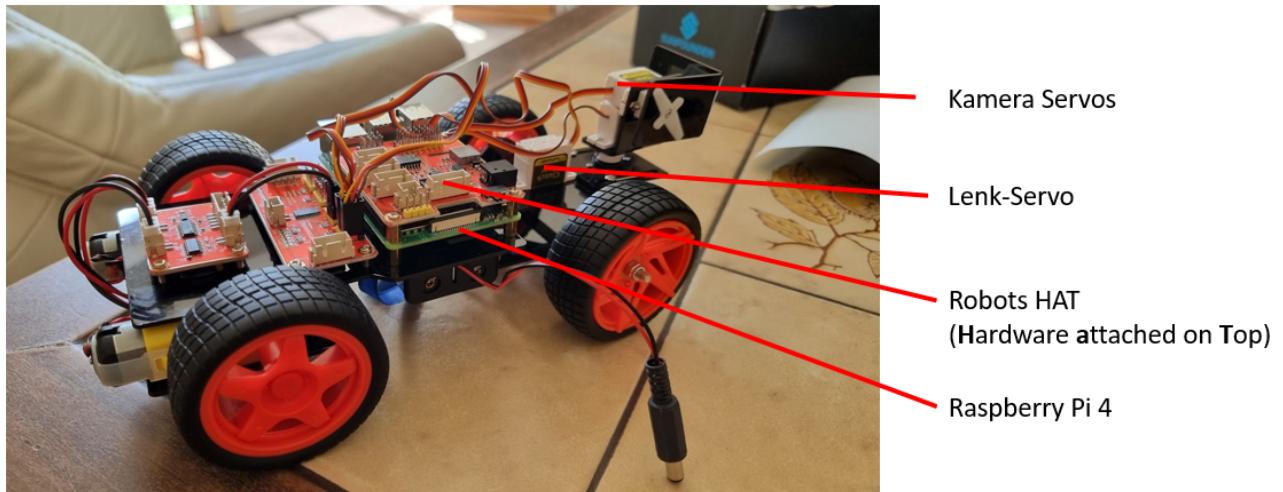
Dadurch schieden etwa Modelle wie der DJI RoboMaster S1 (nicht frei programmierbar) oder diverse andere Kits aus, die oft nicht mit einem Servo (=stufenlose Lenkung) ausgestattet waren, sondern nur binär (voller Lenkeinschlag links/rechts) oder nur per Allradantrieb lenken konnten. Letztendlich fiel unsere Wahl auf das [PiCar-V von SunFounder](#). Das Kit kommt mit Chassis, Motoren, Servos, Batteriefach, Kamera und Verkabelung. Als "Gehirn" des Fahrzeugs dient ein Raspberry Pi, der im Fahrzeug montiert wird und für die Steuerung zuständig ist. Wir entschieden uns für einen Raspberry Pi 4 mit 4 GB Arbeitsspeicher, dieser muss separat erworben werden.

Die Beschaffungsphase selbst nahm nicht viel Zeit in Anspruch. Die Lieferung erfolgte sehr schnell. Für die Stromversorgung mussten zusätzlich Akkus des Typs 18650 bestellt werden (wir haben diese aufgrund eines überlegenen Preisleistungsverhältnisses im [diesem Paket](#) gekauft).

2.2. PiCar-V

Das PiCar-V wurde mit einer detaillierten Anleitung (<https://docs.sunfounder.com/projects/picar-v/en/latest/>) in vielen kleinen Einzelteilen geliefert. Es musste zunächst also erstmal zusammengebaut werden. Dies nahm insgesamt ca. 2-3 Tage in Anspruch. Alle Werkzeuge, die zum Installieren benötigt wurden, waren im Lieferumfang enthalten.





Bei der ersten Inbetriebnahme konnte leider keine Steuerung der Servomotoren erfolgen. Damit fielen Lenkung und Kamerasteuerung aus. Das Auto war zunächst leider damit unbenutzbar. Nach langer Recherche und Kontaktaufnahme konnte ermittelt werden, dass ein Hardwaredefekt bzw. Kabelbruch vorlag. Diese Prozedur kostete leider mehrere Tage. Vorsorglicherweise und wahrscheinlich, weil solche Probleme bekannt sind lagen bei den Einzelteilen auch viele Ersatzteile mit bei, wodurch das Kabel ausgetauscht werden konnte. Zusammenfassend ließ sich das Auto, mit der beigelegten Anleitung, gut zusammenbauen und funktionierte nach den Servo-Problemen wie erwartet.

Limitationen

Leider haben wir erst nach der Bestellung und dem Aufbau des Pi Cars festgestellt, dass der typische Steckplatz für Erweiterungen des Raspberry Pi's bereits durch das Pi Car in Anspruch genommen wird. Dadurch ist die Verwendung eines 5G HATS (Hardware attached ontop) nicht mehr möglich. Da der Markt für mobile 5G Module ohnehin bereits stark eingeschränkt und entsprechend kostspielig ist (ab etwa 400€), haben wir uns dazu entschieden, den Fokus des Projekts zunächst auf den Aufbau und die Inbetriebnahme des Pi Cars zu legen und einen Unterschied in der Latenz zwischen 4G und 5G softwareseitig zu simulieren.

2.3. Raspberry Pi

Aufsetzen des Pi's erfolgte nach Schritt für Schritt [Anleitung des Herstellers](#). Diese liegt der Abgabe bei und ist auch online verfügbar. Dazu gehört neben dem Aufsetzen des

Betriebssystems auch die Installation der Serversoftware (Python django Webserver), die frei auf GitHub verfügbar ist.

https://github.com/sunfounder/SunFounder_PiCar-V

Limitationen

Eine der größten Einschränkungen und Ärgernisse während der Entwicklung ist die fehlende Funktion des Raspberry Pi's, sich automatisch mit bekannten und verfügbaren WLAN-Netzwerken zu verbinden, sofern diese nicht zuletzt manuell ausgewählt wurden. Praktisch heißt das, dass sich der Pi zwar problemlos mit dem letzten aktiven Netzwerk wiederverbindet. Startet man den Pi nun jedoch an einem anderen Ort mit einem anderen Netzwerk, verbindet er sich nicht automatisch mit diesem, selbst wenn es früher bereits verbunden war. Im Laufe des Projekts haben wir zahlreiche Tutorials und Workarounds ausprobiert, um diese eigentlich grundlegende Funktion zu aktivieren - leider jedoch ohne Erfolg. Soll das Pi Car also in einem anderen Netzwerk verwendet werden, muss dieses stets manuell ausgewählt werden.

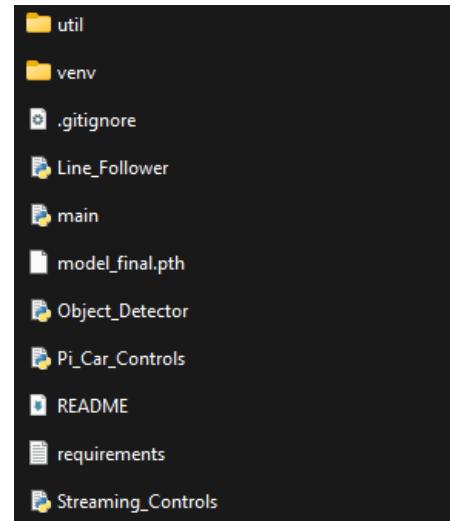


3. Code

Der von uns entwickelte Code für die Steuerung des Pi Cars liegt in der abgebildeten Struktur vor. Im Folgenden werden die wichtigsten Komponenten und deren Funktionsweise kurz vorgestellt und im Anschluss beschreiben wir, wie das Projekt lokal aufgesetzt werden kann.

Der gesamte Code ist auch auf GitHub verfügbar.

https://github.com/max-gmann/5G_PiCar



3.1. Pi Car Server

Der Pi Car Server ist ein django Webserver, der eingehende GET requests entgegen nimmt und diese in Steuerungsbefehle umsetzt. Außerdem stellt er eine rudimentäre Webseite zur Verfügung, über die man das Auto fernsteuern kann und das Bild der Webcam streamt. Diese Seite ist über <http://raspberrypi:8000> erreichbar.

Für die weitere Implementierung muss dieser Code nicht weiter angepasst werden. Wir haben nur einen Fehler beim Laden der Servo-Konfiguration behoben. Standardmäßig muss der Server gemäß der Anleitung mit folgenden Befehlen gestartet werden.

```
Shell
Reset Copy Run Stop
1 cd ~/SunFounder_PiCar-V/remote_control
2 python3 manage.py migrate
3 sudo ./start|
```

Damit für die zukünftige Nutzung nicht jedes Mal auf den Pi direkt zugegriffen werden muss, haben wir einen CRON Job eingerichtet, der das Script automatisch bei jedem Start ausführt. Um einen laufenden Server dennoch zu beenden, kann das Kommando

sudo pkill -f python



genutzt werden. Der Neustart des Servers erfolgt dann entweder entsprechend den oben abgebildeten Kommandos oder durch Nutzung des auf dem Desktop abgelegten Shell Scripts.

3.2. main Methode

Das Python Script "main.py" beinhaltet die zentrale Steuerungsschleife des Pi Cars. Kern dieser Endlosschleife ist ein Scheduler, der dafür sorgt, dass die Bearbeitung und Anzeige eines neuen Bilds in gleichen Abständen erfolgt. Standardmäßig orientieren wir uns dabei an der maximalen Bildwiederholrate (Frames per Second, FPS) der Webcam des Pi Cars von 25 FPS.

In der Main Methode wird eine Instanz der "pi_car" Klasse aus der Datei "Pi_Car_Controls.py" erstellt. Mit dieser kann das Auto und die Kamera gesteuert werden. Des Weiteren wird eine Instanz der "video_streamer" Klasse erstellt, mit welcher jegliches Bildmaterial der Fahrzeugkamera verfügbar gemacht wird.

Außerdem werden aus der Datei "Object_Detector.py" Instanzen der "stop_sign" Klasse für die Stoppschilderkennung und der "person" Klasse für Personenerkennung erstellt. Für die Verfolgung der Linien wird eine Instanz der "LineFollower" Klasse erstellt.

Anschließend wird das Keymapping, sprich eine Belegung der Tastatur Tasten vorgenommen, damit das Fahrzeug im manuellen Modus mit Tasturbefehlen gesteuert werden kann.

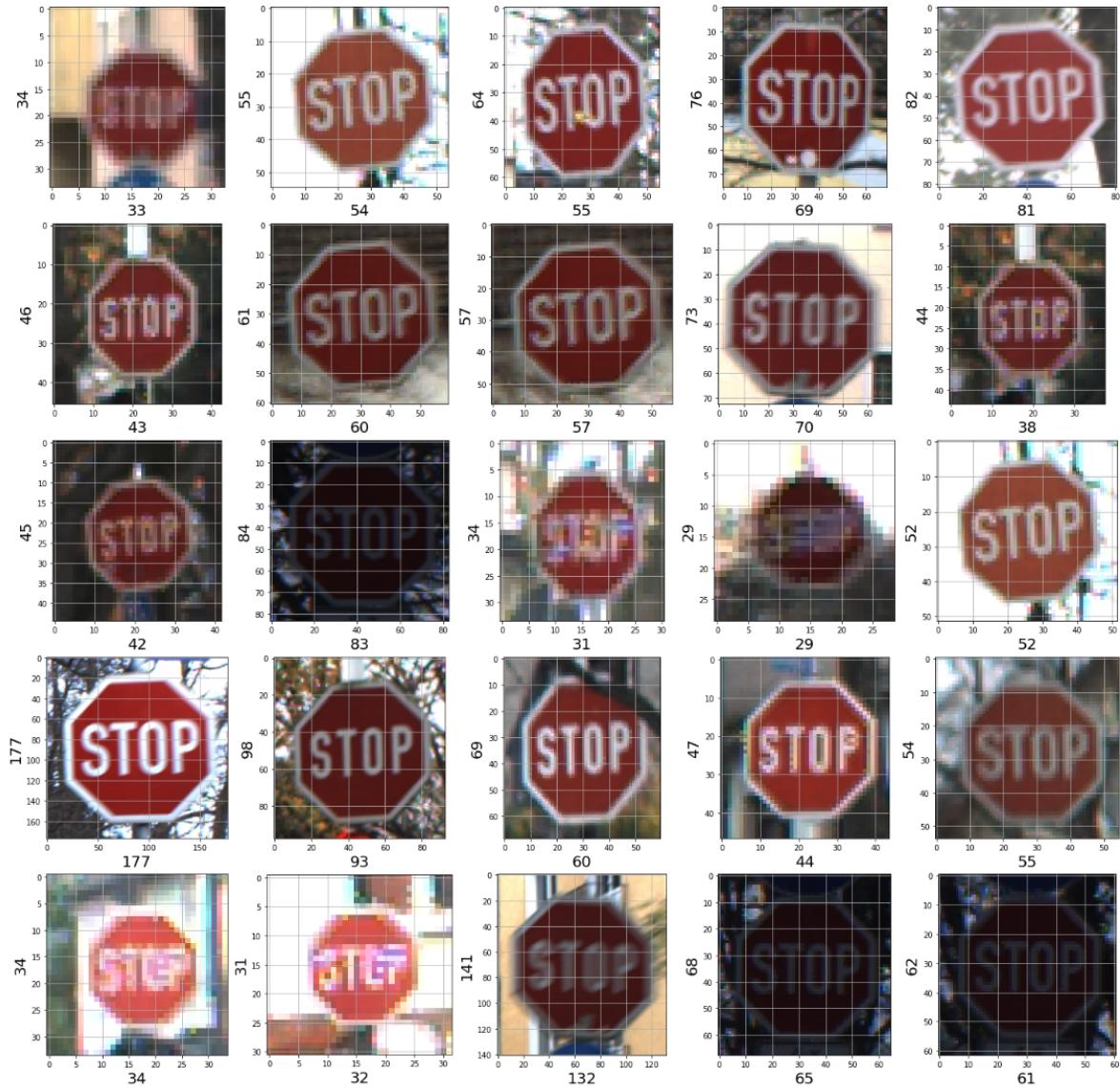
Im automatischen Modus wird ein farbiger Rahmen auf Basis der Stoppschilderkennung angezeigt. Grün heißt fahren, rot heißt stehen bleiben, blau heißt fahren, nachdem an einem Stoppschild angehalten wurde und das Stoppschild noch im Sichtfeld ist.

Die callback_fnc Funktion wird aufgerufen, wenn eine Taste gedrückt wird. Es wird die Geschwindigkeit des Autos gesetzt und es kann zwischen manuellem und automatischem Modus umgeschaltet werden.

3.3. Stoppschild Erkennung

Für die Stoppschilderkennung wurde ein eigenständiges Neuronales Netzwerk trainiert. Konkret wurde ein Faster RCNN Modell mit einem Resnet 101 Backbone verwendet. Das von Facebook AI Research entwickelte Open Source Framework detectron2 stellt dazu auch bereits auf einem allgemeineren Datensatz vortrainierte Modelle bereit. So können wir auch

bereits mit wenig Trainingsdaten gute Ergebnisse erzielen. Als Datensatz dient der German Traffic Sign Recognition Benchmark, der Bilder diverser deutscher Verkehrsschilder enthält. Für unser Modell haben wir den Datensatz nur auf Bilder von Stoppschildern gefiltert. Die nachfolgende Abbildung zeigt davon einen Auszug:





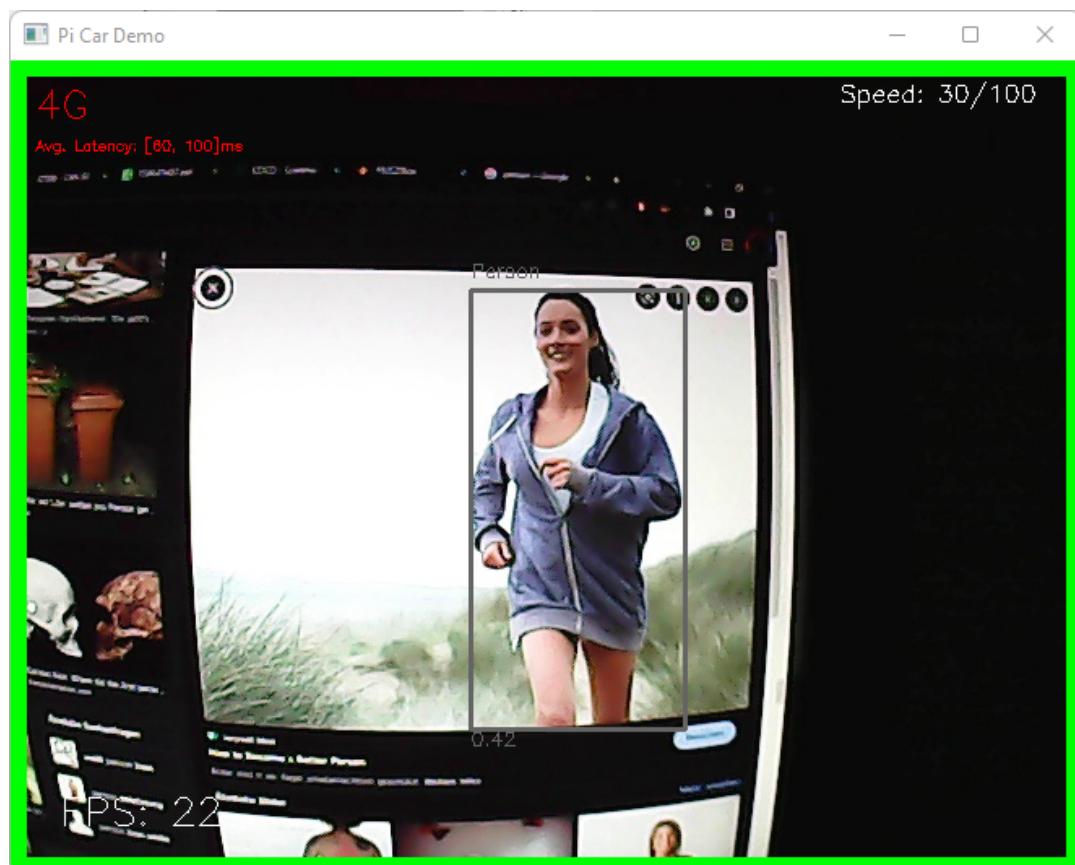
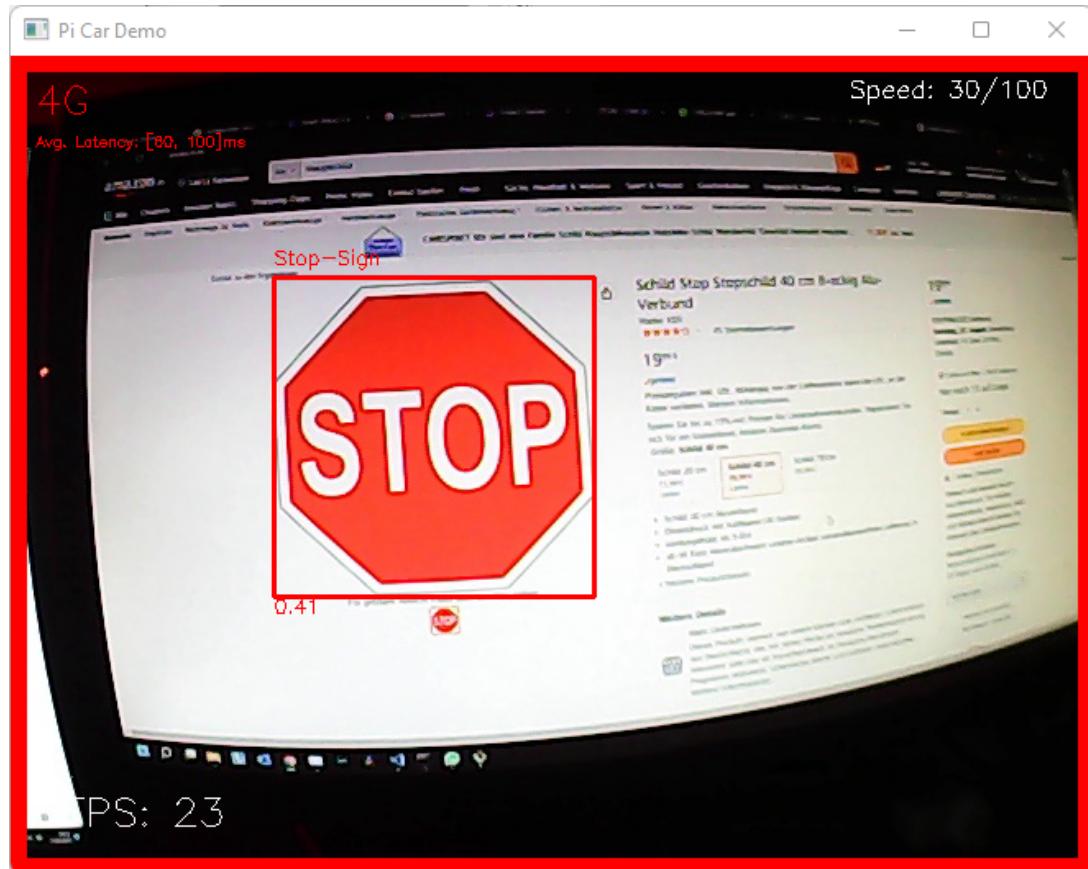
Das so trainierte Modell war anschließend in der Lage, zuverlässig Stoppschilder selbst in sehr großen Entfernungen zu erkennen.



Die Nutzung dieses Modells zusammen mit dem Pi Car war anschließend unkompliziert. Das Auto konnte meist zuverlässig anhalten und fuhr nach einer festgelegten Zeit automatisch weiter. Wir mussten jedoch feststellen, dass die verwendete Netzwerkarchitektur einen verhältnismäßig hohen Berechnungsaufwand hat. Dadurch konnten selbst mit einer NVIDIA RTX 3070Ti nur etwa 12 FPS erreicht werden. Während diese Performance für langsame Geschwindigkeiten noch ausreichend war, stieß das Auto bei höheren Geschwindigkeiten schnell an seine Grenzen.

Um eine schnellere Inferenz zu erreichen (Vorhersage eines neuronalen Netzwerkes), verwenden wir nun die sogenannte YOLO Architektur (You only look once). Diese ermöglicht eine deutlich schnellere Vorhersage mit einer für unsere Zwecke immer noch ausreichenden Genauigkeit. Das Modell wurde mit den Daten des COCO Benchmarks trainiert. Dieser umfasst über 200.000 annotierte Bilder und 80 Objektkategorien, darunter auch Stoppschilder. Mit diesem Modell konnten wir in der Vorhersage unsere angepeilten 25 Bilder pro Sekunde erreichen bei deutlich geringeren Hardwareanforderungen.

Um nicht sofort bei der Erkennung eines Stoppschildes in der Entfernung anzuhalten, berechnen wir die relative Größe der erkannten Bounding Box. Erst ab einem im Code festgelegten Schwellenwert wird der Befehl zum Anhalten gesetzt. Um möglichen Anomalien und falschen Erkennungen vorzubeugen, muss ein Stoppschild außerdem über eine festgelegte Anzahl aufeinander folgender Bilder erkannt werden. Erst wenn dies der Fall ist, wird das Stoppschild als "aktiv" behandelt. Gleichfalls verhält es sich im umgekehrten Fall, damit das Auto nicht fälschlicherweise los fährt, nur weil in einem Frame kein Schild erkannt wurde, im nächsten aber doch. Die entsprechenden Konfigurationen können in der Datei *Object_Detector.py* vorgenommen werden.

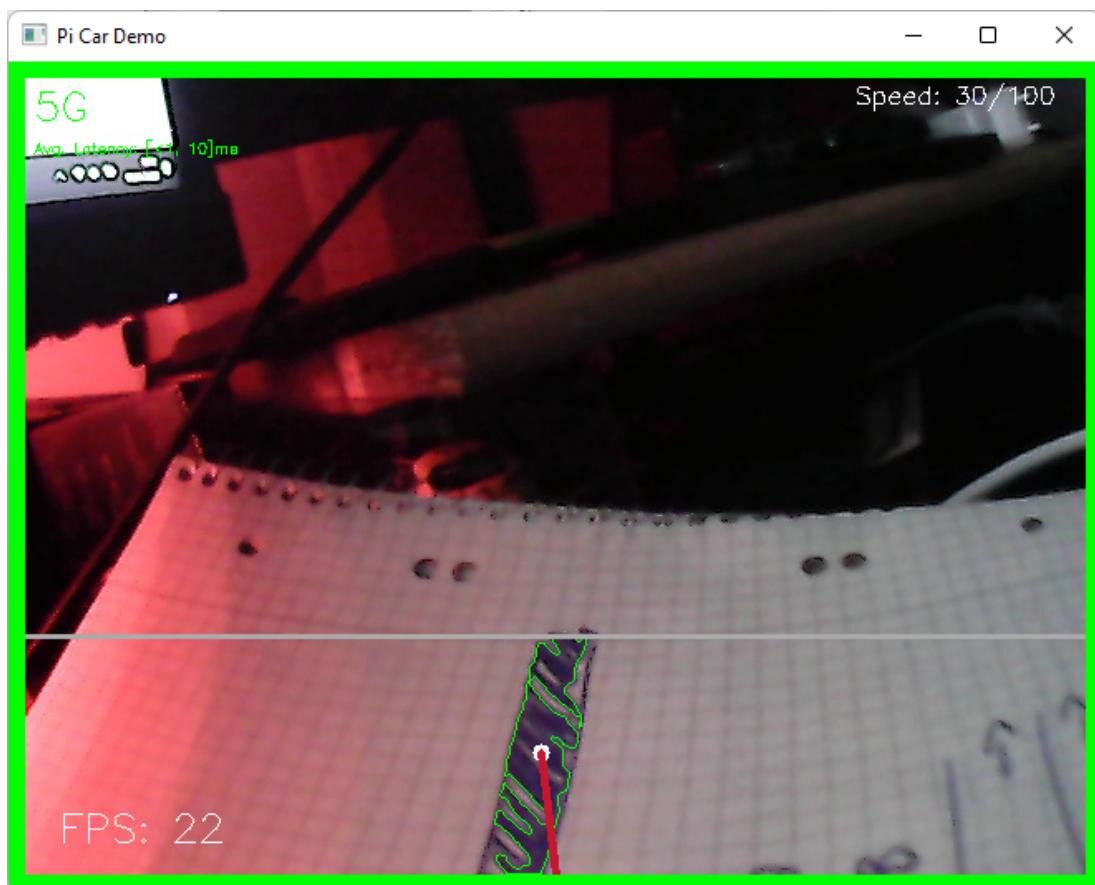




3.4. Linienerkennung

Die Erkennung der Linie geschieht in der "Line_Follower.py" Datei. Diese beinhaltet alle Funktionen, die zum Folgen der Linie notwendig sind. Zunächst werden die Weitwinkelbilder der Kamera auf den unteren Teil, zur Vermeidung von Fehlererkennung anderer Objekte zugeschnitten.

In der Funktion zum ermitteln des korrekten Lenkwinkels, um der Linie zu folgen, sind zunächst 2 Kontrastfarbtöne einprogrammiert, um eine helle oder dunkle Linie zu erkennen. Es wird der Linienmittelpunkt ermittelt und mit einem Kreis versehen, um damit X und Y Werte zu erhalten. Anhand des X Wertes wird der notwendige Lenkwinkel bestimmt. Der Lenkwinkel wird bei jedem dritten Bild ausgewertet und übertragen. Dieser Parameter kann in der *main.py* angepasst werden.



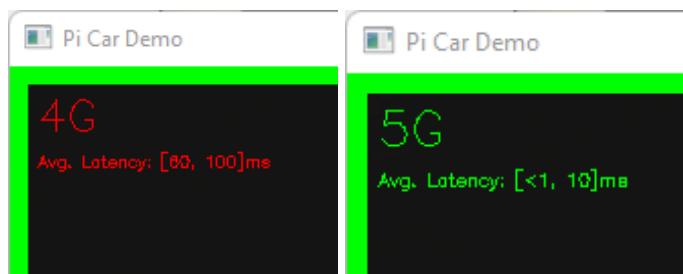


3.5. Herausforderungen

Für die Verarbeitung und Darstellung der übertragenen Webcam Bilder verwenden wir die weit verbreitete Bibliothek openCV. Schon früh mussten wir jedoch zu unserer Überraschung feststellen, dass allein das Zeichnen von Rechtecken oder Text auf ein Bild nicht sehr performant ist. Auch das Auslesen des MJPG Videostreams war sehr langsam und wurde daher in einen separaten Thread ausgelagert. Für das Einzeichnen des Overlays verlieren wir in etwa 2-3 FPS.

3.6. 4G & 5G Modus

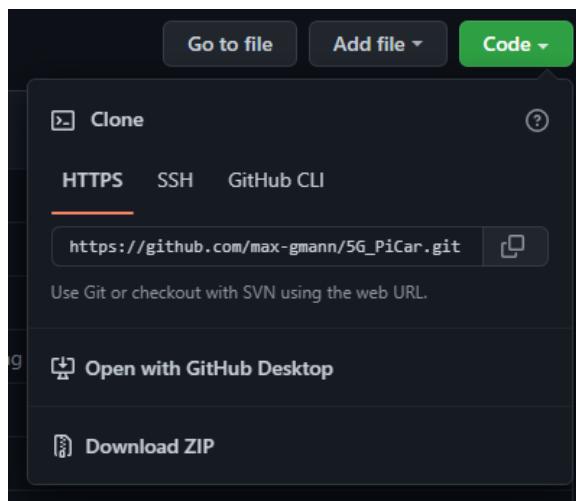
Wie zuvor beschrieben wird 4G vs. 5G aufgrund der Hardware Limitation nicht mit einem 5G/4G Modul durchgeführt, sondern simuliert. Der 5G Modus stellt den Modus dar, welcher ohne Verzögerung mit bester Performance läuft. Im 4G Modus wird eine künstliche Latenz generiert. Diese wird durch verzögerte Verarbeitung der Bilder erzeugt. Sprich die Bilder werden zwischengespeichert. Es wird immer ein 3 bis 5 Bilder altes Bild verarbeitet und somit eine Verzögerung von 100 bis 200ms erzeugt. Dieser Wert entspricht in etwa dem doppelten der realen Verzögerung von 4G. Aus Gründen der Sichtbarkeit bei der Simulation wurde der Ausgangswert von etwa 80ms erhöht. Der Verbindungsmodus kann bei laufendem Programm mit der Taste **C** geändert werden.



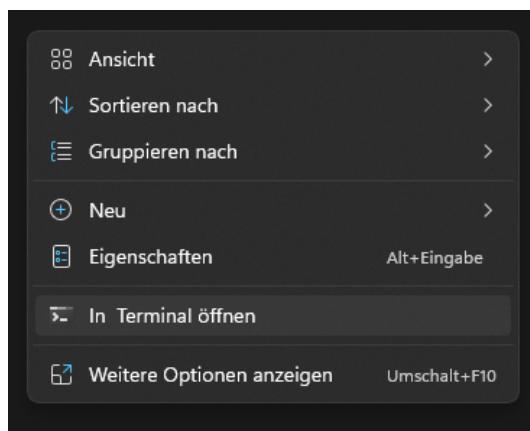
3.7. Installation

Im Folgenden werden die Schritte zur Installation auf einem Windows System beschrieben.
Das Vorgehen für Linux oder MacOS kann sich leicht unterscheiden.

1. Python manuell oder über den Windows Store herunterladen und installieren.
<https://www.python.org/downloads/release/python-3912/>
2. ZIP Archiv von https://github.com/max-gmann/5G_PiCar herunterladen und am gewünschten Ort entpacken.



3. Im Ordner mit den entpackten Dateien ein Terminal öffnen, beispielsweise per Rechtsklick.





-
4. Neue virtuelle Umgebung erstellen:

```
python -m venv venv
```

5. Virtuelle Umgebung aktivieren:

```
.\venv\Scripts\activate
```

Sollte die Ausführung in der Windows PowerShell geblockt werden, kann folgendes Kommando helfen:

```
Set-ExecutionPolicy -ExecutionPolicy RemoteSigned -Scope CurrentUser
```

Wurde die Umgebung erfolgreich aktiviert, wird der Name (venv) am Beginn der Zeile angezeigt (siehe Abbildung).

```
PS C:\Users\max24\Downloads\5G_PiCar-master\5G_PiCar-master> .\venv\Scripts\activate  
(venv) PS C:\Users\max24\Downloads\5G_PiCar-master\5G_PiCar-master>
```

6. Benötigte Pakete installieren:

```
pip install -r requirements.txt
```

7. Programm starten:

```
python main.py
```

Für zukünftige Starts ist entscheidend, dass das Script aus dem Kontext der virtuellen Umgebung ausgeführt wird. Nur so kann der Python Interpreter alle relevanten Bibliotheken finden. Gegebenenfalls müssen also die Schritte ab Schritt 5. erneut ausgeführt werden.

3.8. Inbetriebnahme

Ist der Code fertig aufgesetzt und die Laufzeitumgebung wurde mit Anaconda Prompt erstellt kann die Inbetriebnahme starten. Es beginnt mit dem Starten des Fahrzeugs über den An und Aus Schalter. Wichtig ist zu beachten, dass die Fahrfunktionen nicht über das Netzteil funktionieren. Um das Auto fahren zu lassen muss die Akkuversorgung aktiviert werden. Nach dem Start des Fahrzeugs wird zunächst eine Internetverbindung durch den PI aufgebaut zum letzten verbundenen Netzwerk (wie bereits unter 2.3.1 Limitation erklärt). Sollte das letzte verbundene Netzwerk nicht verfügbar sein so muss der PI per Micro HDMI Kabel an einen Bildschirm angeschlossen werden und es muss sich mit dem verfügbaren Netzwerk verbunden werden. Ist der Pi verbunden und startklar, wird der Server des Fahrzeugs automatisch gestartet und der Code kann ausgeführt werden. Dazu wird in der Anaconda Prompt Konsole in den Ordner navigiert, in dem sich der Code befindet. Dies erfolgt über die bekannten navigationsbefehle, in diesem Fall cd.

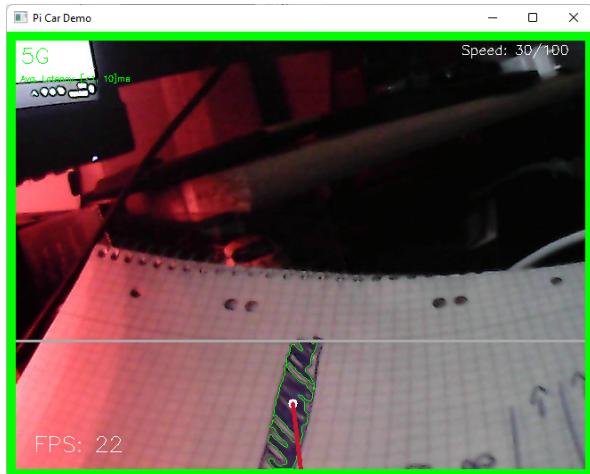


Beispiel: “cd C:\OFFLINE\IV_Projekte\picarv_Stopschild\5G_PiCar”. Um Fehler zu vermeiden kann der Ordnerpfad aus dem Datei Explorer kopiert werden. Nachdem der Ordner angewählt wurde muss die Laufzeitumgebung aktiviert werden. Dies erfolgt mit dem Befehl “conda activate pi_car”. Zuletzt wird die “main.py” Datei mit “python main.py” ausgeführt. Wurden die Schritte richtig ausgeführt sieht der Verlauf wie folgt aus:

```
(base) C:\Users\fgehl>cd C:\OFFLINE\IV_Projekte\picarv_Stopschild\5G_PiCar  
(base) C:\OFFLINE\IV_Projekte\picarv_Stopschild\5G_PiCar>conda activate pi_car  
(pi_car) C:\OFFLINE\IV_Projekte\picarv_Stopschild\5G_PiCar>python main.py
```



Ist der Code ausgeführt und die Kamera funktioniert (siehe bekannte Probleme) wird folgendes Fenster geöffnet:



Die Steuerung des Fahrzeugs wird in dem anschließenden Kapitel beschrieben.

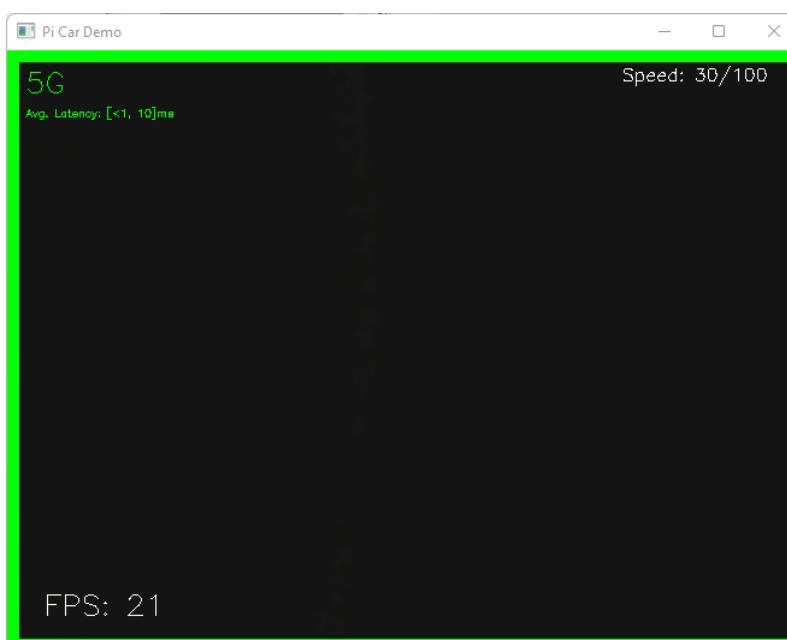


3.9. Steuerung

Wurde das Fahrzeug erfolgreich gestartet, kann die Steuerung des Fahrzeugs beginnen. Es wird im manuellen Modus gestartet. In diesem Modus kann über die Tastatur mit folgender Tastenbelegung gesteuert werden:

W	vorwärts
S	rückwärts
A	links
D	rechts
C	4G / 5G Verbindungsmodus umschalten
M	Umschalten manueller/automatischer Modus
H	Umschalten Erkennung heller/dunkler Linie
F	Linienerkennung aktivieren / deaktivieren (nur automatischer Modus)
P	Stoppschilderkennung aktivieren/deaktivieren (nur automatischer Modus)
Leertaste	Automatisches Vorwärtfahren umschalten (nur automatischer Modus)
Pfeiltasten	Feinjustierung der Kamera
1 bis 9	Geschwindigkeit in 5-Schritten anpassen

In dem automatischen Modus kann die Erkennung von heller und dunkler Linie umgeschaltet werden, sowie in den manuellen Modus zurück gewechselt werden.





3.10. Bekannte Probleme

3.10.1. Nicht erkannte Kamera

Ein fortwährendes Problem ist die Nicht-Erkennung bzw. Initialisierung der Kamera. Der Pi erkennt die Kamera in diesem Fall nicht und es kann kein Bild verarbeitet werden. Es kommt zu einer Fehlermeldung. Dieser Fehler tritt nicht reproduzierbar auf und konnte auch in Absprache mit dem Hersteller nicht gefunden werden.

Die Lösung in diesem Fall ist es, das Fahrzeug komplett neu zu starten, indem es über den Schalter komplett ausgeschaltet wird. Mit Hilfe des Server-Auto-Starts fahren sich alle benötigten Programme automatisch neu hoch.

3.10.2. IP wird nicht erkannt

In bestimmten Netzwerkkonfigurationen kann es dazu kommen, dass der Router den DNS Namen <http://raspberrypi> nicht in eine IP Adresse auflösen kann. In diesen Fällen muss manuell die IP Adresse des Autos im Konsolenfenster eingegeben werden.

3.11. Laufzeiten

In unseren Tests kann das Pi Car mit einer Akkuladung etwas über eine Stunde bei dauerhafter Fahrt und mittlerer Geschwindigkeit bewegt werden. Die genaue Akkulaufzeit variiert jedoch je nach gewähltem Fahrmodi, Geschwindigkeit, verwendeten Akkus und dem Untergrund. Sobald die Akkus keine ausreichende Spannung für den Raspberry Pi mehr liefern können, schaltet sich dieser selbstständig ab. Leider verfügt der Pi nicht über die Möglichkeit, die verbleibende Spannung auszulesen, weshalb es nicht möglich ist, den Ladezustand anzuzeigen. Sobald der Akkustand zu niedrig wird, fangen die hinteren beiden grünen LEDs an zu blinken.



4. Abrechnungsmodelle

5G Anbieter können nicht auf die traditionellen Abrechnungsmodelle zurückgreifen, da 5G sehr variable Attribute, wie z.B. Bandbreite, Latenz und Dienstqualität besitzt, die je nach Komplexität und Wert einen anderen Ansatz für die Preisgestaltung erfordern. Herkömmliche Modelle sind nicht flexibel oder agil genug, um 5G Dienste angemessen zu monetarisieren. Daher ist es notwendig, neue Ansätze zur Abrechnung zu entwickeln.

4.1. Lokale Anwendungen

Die Bundesnetzagentur hat die 5G-Frequenzgebühren für lokale Anwendungen folgendermaßen festgelegt. Demnach wird in jedem Einzelfall die Gebühr nach der Gebührenformel berechnet.

$$\text{Gebühr} = 1000 + B \cdot t \cdot 5(6a_1 + a_2)$$

1000	Sockelbetrag in €
B	Bandbreite (10 bis max. 100 MHz)
t	Laufzeit der Zuteilung (in Jahren bzw. anteilig je angefangenem Monat)
a_1	Fläche des Zuteilungsgebietes in km^2 (Siedlungs- und Verkehrsflächen)
a_2	Fläche des Zuteilungsgebietes in km^2 (andere Flächen)

(vgl. Bundesnetzagentur 2019)

4.2. Weitere Modelle für Endnutzer

Latenz in die Abrechnung mit einbeziehen

Eine weitere Möglichkeit der Abrechnung ist es, die Latenz mit in die Kalkulation einzubeziehen. 5G bietet den Vorteil Latenzen von bis zu unter 1 ms zu bieten. Netzbenutzer, die diese niedrige Latenz benötigen, würden dementsprechend einen Aufpreis bezahlen. Sonstige Nutzer würden dann beispielsweise nur noch eine Latenz, die der Hälfte



von 4G, also ca. 40 ms Sekunden entspricht erhalten. Für z.B. autonome Fahrzeuge oder die zunehmende Anzahl an Mobile Gamern wäre dies eine mögliche zusätzliche Abrechnungsmethode zu dem herkömmlichen Modell. Für private Endkonsumenten könnte ein Aufpreis von 5 Euro im Monat eine mögliche Summe sein.

Herkömmliches Modell

Für normale Konsumenten sollte das bekannte Abrechnungsmodell über die Datenvolumen Menge nicht vernachlässigt werden. Dies dient auch dazu, die Netzstabilität zu gewährleisten, denn längst nicht jeder Benutzer benötigt eine unbegrenzte Menge Datenvolumen unterwegs. Es ist nicht sinnvoll und notwendig, dass plötzlich mit 5G jeder Nutzer unterwegs alle seine Apps und Programme aktualisieren muss. Mit begrenztem Datenvolumen führen viele Nutzer solche Datenvolumen intensive Aktivitäten zu Hause meist über Nacht im WLAN aus.



Literaturverzeichnis

Bundesnetzagentur (2019): 5G-Frequenzgebühren für lokale Anwendungen, [online]
https://www.bundesnetzagentur.de/SharedDocs/Pressemitteilungen/DE/2019/20191031_LokalesBreitband.html [abgerufen am 05.07.2022].