THM
TECHNISCHE HOCHSCHULE MITTELHESSEN

ITI
Institut für Technik und Informatik

Ausarbeitung

II2021 Entwicklung eines autonomen Fahrzeugs

Sommersemester 2022

Sven Thomas und Maximilian Biebl
sven.thomas@mni.thm.de
maximilian.biebl@mni.thm.de

Dozent:
Jakob Czekansky, M.Sc.

5. August 2022
Technische Hochschule Mittelhessen, Gießen

Inhaltsverzeichnis

1 Einleitung

Nachfolgend ist unsere Ausarbeitung zu unserem Projekt für das Modul 'II2021 Entwicklung eines autonomen Fahrzeugs' zu lesen. Im Laufe des Projektes haben wir verschiedene Lösungsansätze für die einzelnen Teilprobleme durchlaufen und auch verworfen. Einige dieser Ansätze und unsere finale Lösung der vorhandenen Problemstellungen sind auf den folgenden Seiten beschrieben. Ermitteltet Ansätze wie es nicht geht oder nur mit weiteren neuen Problemen sind hier der Vollständigkeit auch aufgeführt.

1.1 Ausgangsposition

Gegeben ist der Gazebo Simulator mit einer Simulation des cITIcars und einer Simulationswelt, welche mehrere Teststrecken beinhaltet. Das gegebene Modell des cITIcars besitzt zunächst nur einen an der linken Seite angebrachten Time-of-Flight-Sensor und eine zentral nach vorne gerichtete Kamera, die auf dem Dach des cITIcars positioniert ist. Weiterhin sind bereits verschiedene ROS-Knoten gegeben, unter anderem, um die Geschwindigkeit und den Lenkwinkel des Autos zusteuern. Für die Umsetzung ist die Programmiersprache Python in Kombination mit dem Framework ROS vorgegeben. Der Wagen darf beliebig um Sensoren erweitert werden, dazu zählen Kameras, Time-of-Flight-Sensoren und Lidar-Sensoren.

1.2 Zielsetzung

Ziel des Projektes ist es, die gegebene Teststrecke (Abb. 12 im Anhang) in möglichst geringer Zeit zu überwinden. Beim Bewältigen der Strecke sind verschiedene Aufgaben bzw. Probleme zu lösen. Die erste grundsätzliche Aufgabe des Projekts ist es, mit dem Auto den Verkehrslinien zu folgen und entsprechend der vorhandenen Linien zu lenken. Weiterhin ist es Aufgabe, eine Parklücke zu erkennen und dort rückwärts ein und wieder auszuparken. Es gilt, Hindernisse auf der Fahrbahn festzustellen und diese zu überholen. Für das Überholen soll ein Wechsel der Fahrspur jeweils von rechts nach links und nach Überwinden des Hindernisses wieder zurück erfolgen. Das Fahrzeug soll seine Geschwindigkeit selbständig anpassen und auch problemlos eine Kreuzung überqueren können.

2 Lenkung anhand der Linien

2.1 Die erste „naive“ Idee und ihre Probleme

Wir entschieden uns zunächst Canny Edge zur Kantenerkennung zu nutzen und Houghline, um aus den ermittelten Kanten dann die Verkehrslinien zu bestimmen. Zu Beginn des Projektes hatten wir über verschiedene Ansätze nachgedacht, wie wir den Lenkwinkel anhand der von Houghline erkannten Vektoren bestimmen können. Einer dieser frühen „naiven“ Ansätze war, den Lenkwinkel anhand eines „durchschnittlichen Vektors“ zu bestimmen, der sich aus den rechts und links von Houghline erkannten Vektoren zusammen setzt. Der Lenkwinkel sollte dann der Winkel zwischen diesem „Durchschnittsvektor“ und der X-Achse sein.

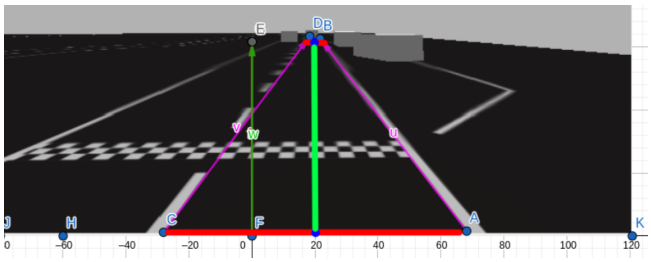

Abbildung 1: durchschnittlicher Vektor am Start

Diese „naive“ Idee wurde beim ersten Testen in GeoGebra bereits schnell wieder verworfen. Das Problem hierbei ist, dass wir eigentlich außerhalb von Kurven (Abbildung 2) den Vektor gespiegelt um die Y-Achse bräuchten, um zur Soll-Fahrbahn zu lenken. In den Kurven (Abbildung 3) wiederum bräuchten man wieder den zuvor genannten „Durchschnittsvektor“. Daher haben wir das Ganze verworfen, aber konnten daraus eine andere Idee entwickeln.

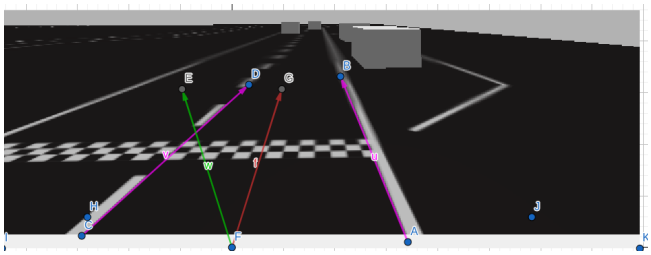

Abbildung 2: zu weit rechts auf der Geraden

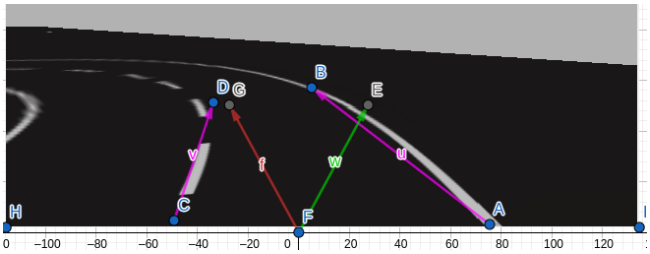

Abbildung 3: durchschnittlichen Vektor in Kurve

- Rot ⇒ Durchschnittsvektor Grün ⇒ gespiegelter Durchschnittsvektor

2.2 Ansatz A: Weiterentwicklung der ersten Idee

Nach Reflektieren unseres ersten Ansatzes stellen wir fest, dass die Y-Koordinaten eigentlich egal sind und dass man lediglich eine X-Koordinate benötigt, um zu wissen, ob man nach links oder rechts lenken muss. So kam uns die Idee, einfach den Durchschnitt aus den linken und rechten X-Koordinaten, die durch Houghline erkannt werden, zu bilden und anhand dessen die Fahrbahn zu bestimmen. Die Idee hierbei ist, die X-Koordinate der Bildmitte, die auch die Mitte des Autos ist, einfach mit der errechneten X-Koordinate zu vergleichen und anhand dessen zu lenken.

Das Ganze haben wir zunächst mit der Mittellinie und der rechten Außenlinie versucht, dabei hatten wir einige Problem mit der Mittellinie. Unter anderem, dass es bei zu hoher Empfindlichkeit von Houghline zu viel Beifang gibt und bei einer zu geringen Empfindlichkeit gibt es das Problem die Linien in den Kurven nicht mehr ausreichend zu erkennen. Darauf hin haben wir auf die beiden Außenlinien gewechselt und unsere Soll-Fahrbahn anhand des 1.25-Fachen deren Durchschnitts-X bestimmt.

Zur Bestimmung des exakten Lenkwinkels wird die X-Koordinate der Bildmitte von dem ermittelten 1.25-Fachen des Durchschnitts-X abgezogen. Das Ganze wird noch auf $\pm$ 45° abgeriegelt.

$$\text{Lenkwinkel} = \text{SollFahrbahn}_X - \text{Bildmitte}_X$$

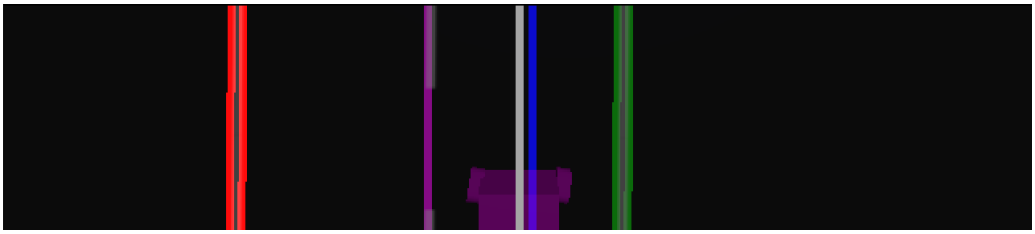

Abbildung 4: Schaubild des Ansatz A

In der Abbildung 4 sind die in der linienbasierten Lenkung erkannten und berechneten Linien zusehen.

- von Houghline erkannte linke Außenlinie
- von Houghline erkannte rechte Außenlinie
- errechnete Mittellinie = Druchschnitts-X aus linken und rechten X-Koordinaten
- 1.25-Fache der Mittellinie ⇒ Soll-Fahrbahn
- die weiße Linie ist die Bildmitte und somit auch die Automitte

Es wird angestrebt, dass die blaue und weiße Linie aufeinander liegen.

2.2.1 Region-of-Interest

Für die beiden Außenlinien gibt es je eine Region-of-Interest. Diese bekommen zunächst einen festen Startbereich. Die Region-of-Interests hatten zu Beginn bei uns eine Trapezform, die den Linien angepasst war, durch den Perspektivenwechsel zur Top-Down-Sicht reichen nun einfache Rechtecke. Da nur der unmittelbare Bereich vor dem Auto relevant ist, wird die Höhe auch stark zugeschnitten. In den beiden Regionen nimmt der Algorithmus erstmal alles, was an X-Koordinaten zu finden ist und bilden draus das zuvor genannte „Durchschnitts-X“ zum Ermitteln der Fahrbahn. Unsere Idee beim Ermitteln des Durchschnitts aus so vielen Koordinaten ist, dass wenn in der Region-of-Interest Störfaktoren sind, dies einfach mit einer Überzahl an richtigen Koordinaten korrigiert wird. Zeitweise haben wir auch versucht, Störfaktoren wie Boxen anhand ihres Grauwertes auszumaskieren, das scheiterte aber an unterschiedlichen Helligkeiten in der Simulation auf verschiedenen Systemen. Sollte in einer Region-of-Interest nichts gefunden werden, wird diese bis zu einem festgelegten Maximalwert schrittweise verbreitert mit dem Ziel etwas zu finden. Ist selbst bei maximaler Breite keine einzige X-Koordinate zu finden, wird einfach der letzte gefundene Wert für die entsprechende Region-of-Interest angenommen. Ist etwas gefunden worden, fällt die Region-of-Interest zurück auf eine minimale Breite und die Suche beginnt erneut.

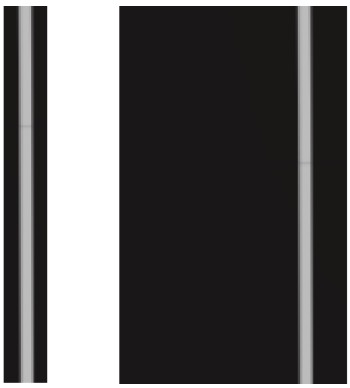

Abbildung 5: unterschiedlich Breite RoI

2.2.2 Verbesserung durch Top-Down-View

Unser Ansatz funktioniert noch deutlich besser durch einen Blick von oben. Hierfür haben wir die Kamera leicht vor dem Auto senkrecht nach unten schauend platziert. Diese vielleicht nicht ganz realistische Änderung hat zu einer solchen Verbesserung geführt, dass wir es dabei belassen haben. Eine reelle Umsetzung wäre vielleicht mit einer Drohne oder einer Art von Angel möglich. Die bessere von beiden Lösungen ist wahrscheinlich mit einer Angel, an deren Ende die Kamera befestigt ist. In unserem Fall wäre diese noch in einer sehr unrealistischen Höhe von einem Meter. Man könnte die Höhe, mit Vergrößern des Öffnungswinkels der Kamera, durch eine kürzere Brennweite und einen größeren Bildsensor verringern. Aus Softwareseite wäre eine Bildtransformation zu einer „pseudo“ Top-Down-Sicht möglich und sinnvoll. Zeitweise hatten wir dahin gehend auch einige Versuche unternommen, aber aufgrund von neuen Problemen, unter anderem wie wir mit zu stark stürzenden Linien umgehen, und aus Zeitgründen haben wir es bei einer nicht so realistischen Lösung belassen müssen.

2.2.3 Probleme des Ansatz A

Der bis jetzt beschriebene Ansatz A zur Lenkwinkelbestimmung ist in der beschriebenen Form vollständig funktionsfähig und kann der Strecke ohne Weiteres folgen. Ansatz A hat jedoch einige weitere Probleme, neben denen, die im Abschnitt zur Top-Down-View bereits angesprochen wurden. Zum Einen schränkt die simple Umsetzng der Region-Of-Interest die Höchstgeschwindigkeit insofern ein, dass gerade in Kurven die Linien nur sehr knapp vor dem Fahrzeug überhaupt erkannt werden. Desweiteren sorgt die strikte Verfolgung der Soll-Fahrbahn für Instabilitäten auf den Geraden, da es konstant zu leichten Schwankungen in der Linienerkennung kommt. Das Resultat dieser beiden Kernprobleme ist letztlich eine langsame Höchstgeschwindigkeit von 0,8 m/s, sowie ein eher unflüssiges Fahrverhalten auf den Geraden. Dies hat uns dazu bewogen, nach einem neuen Ansatz zur Linienerkennung und dem damit verbundenen Lenkvorgang zu suchen.

2.3 Ansatz B: PID-Reglung und "RoI-freie" Linienerkennung

Um den zuvor erwähnten Problemen des Ansatz A entgegenzuwirken, haben wir uns zunächst eine Linienerkennung überlegt, welche nicht länger auf eine Region-Of-Interest im Sinne von Bildmasken angewiesen ist. Die Idee hierbei war es, ausgehend von der Vorder- und Hinterachse, symbolische „Sucher“ nach links und rechts zu schicken, die die jeweiligen Außenlinien der Straße finden sollten. Durch die gefundenen Linien lässt sich hierbei eine Soll-Fahrbahn errechnen, welche nicht nur stabiler, sondern auch flexibler als die des Ansatz A ist.

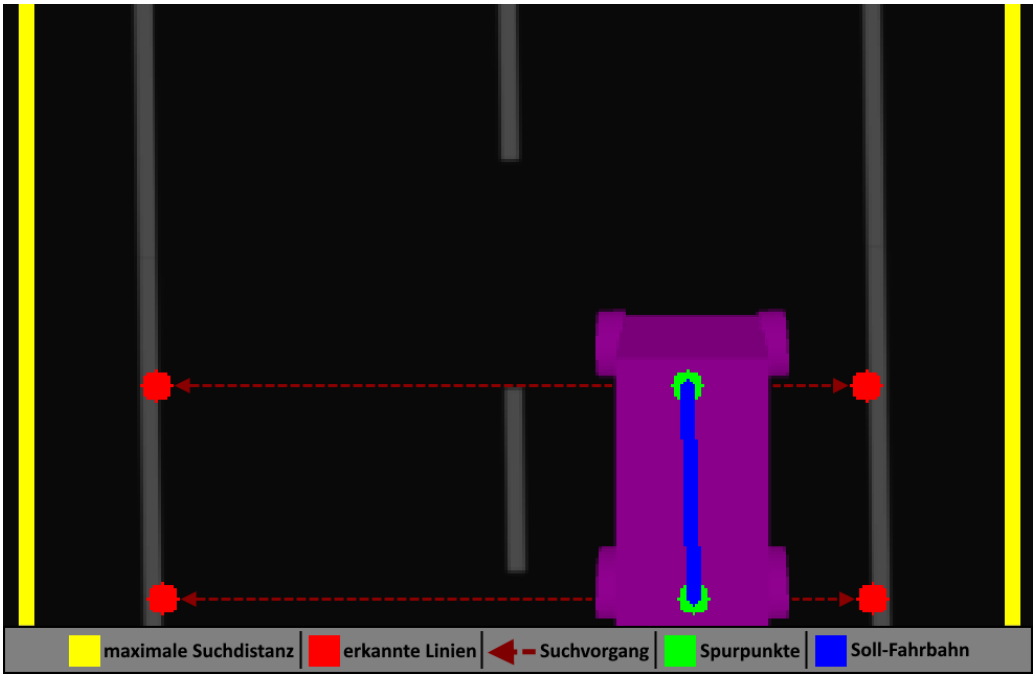

Abbildung 6: Schaubild des Ansatz B

2.3.1 Allgemeines zur Linienerkennung

Wie in Abbildung 6 abgebildet, gehen wir an vier verschiedenen Punkten so lange nach rechts (bzw. links) bis wir auf eine Linie treffen. Im Gegensatz zum Schaubild verwenden wir hierzu allerdings ein Bild, welches die Mittellinie entfernt. Der eigentliche Suchvorgang läuft hierbei über eine Schleife, welche so lange weiterläuft, bis sie auf einen hellen Punkt im Bild stößt oder die maximale Suchdistanz erreicht und auf den zuvor gefundenen Punkt zurückgreift. Die resultierenden Spurpunkte ergeben sich dabei aus dem 0.75-fachen der Distanz zwischen den erkannten Linien.

2.3.2 Filterung der erkannten Linien

Nachdem die Sucher auf einen hellen Punkt im Bild stoßen müssen sie zunächst noch durch zwei Filter laufen, welche eine Fehlerkennung an den Hindernissen sowie an horizontalen Linien (bspw. der Start- und Ziellinie) vermeiden sollen. Im ersten Filter schauen wir dabei auf den X-Wert des zuvor gefundenen Punktes des jeweiligen Suchers. Überschreitet die Differenz der beiden Punkte einen gewissen Grenzwert, so wird der Punkt abgelehnt und stattdessen der alte verwendet. Durch diesen Filter werden bereits beide oben benannten Problemfälle abgedeckt, jedoch mit einem gewissen Restfehler, wenn die horizontalen Linien nicht ganz im 90°-Winkel zur Kamera stehen. Deshalb gibt es noch einen zweiten Filter, welcher von dem gefundenen Punkt eine Distanz größer der Linienbreite weiterspringt, mit der Erwartung nun einen schwarzen Pixel vorzufinden. Ist dies nicht der Fall wird auch hier der Punkt verworfen. Mit dieser Kombination an Filtern werden die Problemfälle sehr zuverlässig abgedeckt.

2.3.3 Lenkwinkelbestimmung durch PID-Regler

Das Problem der bisherigen Linienerkennung ist (wie in 2.2.3 beschrieben), dass bereits kleine Schwankungen des Soll-Wertes zu Instabilitäten im Fahrverhalten führen. Entsprechend haben wir uns an dieser Stelle einen Ansatz mittels zweier PID-Regler überlegt, welche den jeweiligen Spurpunkt (Soll-Wert) mit der Fahrzeugmitte (Ist-Wert) auf gleicher Höhe abgleicht. Um das Lenkverhalten weiterhin zu stabilisieren, wird für den letztlichen Lenkwinkel der Durchschnitt der beiden errechneten Error-Werte verwendet.

2.3.4 Vorteile gegenüber Ansatz A

Ein zentraler Vorteil gegenüber des ursprünglichen Ansatzes ist die Robustheit der Lenkwinkelbestimmung. Selbst wenn einer der Spurpunkte durch eine fehlerhafte Linienerkennung kurzzeitig versetzt ist, bleibt das Fahrzeug weiterhin stabil auf seiner Fahrbahn (s. Abbildung 7). Weiterhin ist das Lenkverhalten im Allgemeinen deutlich flüssiger als zuvor, auf den Geraden ist keine wirkliche Lenkbewegung mehr zu erkennen und in den Kurven kann das Fahrzeug problemfrei eine mittige Position halten ohne dabei zu schwanken (s. Abbildung 8). Als Resultat liegt die Höchstgeschwindigkeit bei diesem Ansatz bei 2,0 m/s, sowohl in Kurven wie auf der Geraden. Im Allgemeinen hat der neue Ansatz also alle Kernprobleme des Vorgängers beseitigt, ausgenommen der Kameraperspektive (mehr dazu in Abschnitt 6.3).

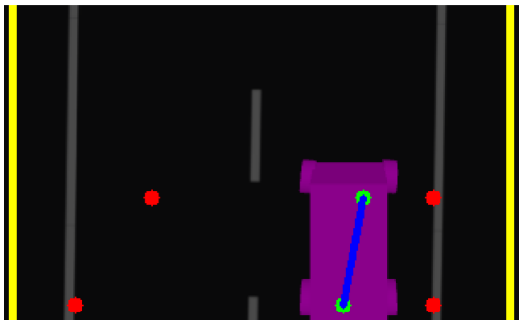

Abbildung 7: Verhalten im Fehlerfall

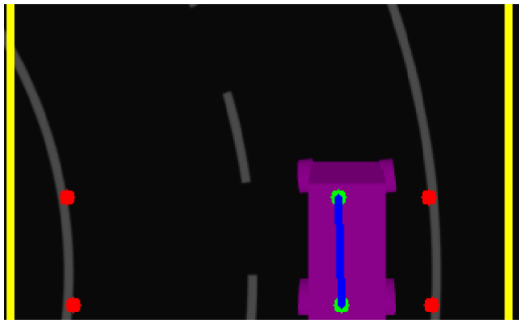

Abbildung 8: mittige Kurvenlage

3 Überholmanöver

3.1 Verworfene Idee des „blinden Mannes“

Ein erster Versuch das Überholmanöver zu realisieren war es, das Auto rein mit dem rechten Time-of-Flight-Sensor an dem Hindernis vorbeihangeln zu lassen, nachdem das Manöver durch den Fornt-Time-of-Flight-Sensor eingeleitet wird. Der Ansatz sollte ähnlich zu dem Beispiel aus der Vorlesung zum PID-Regler funktionieren, bei dem es darum ging, mittels eines Time-of-Flight-Sensors um eine Box im Kreis zu fahren. Bei dieser Lösung hätten wir auf die sonstigen Sensoren verzichtet und das Auto wie ein blinder Mann an einem Geländer an der Box entlang hangeln lassen. Problem für uns war bei diesem Ansatz zu erkennen, wann wir das Manöver sinnvoll beenden. Daher wurden die dahingehenden Versuche auch wieder verworfen.

3.2 Einleitung des Manövers

Für das Überholmanöver haben wir das Fahrzeug zunächst um weitere Time-of-Flight-Sensoren ergänzt (Abbildung 9). An der Front haben wir einen sehr schmalen, nach vorne gerichteten Time-of-Flight-Sensor. Die geringe Breite ist dafür da, dass nicht auf der Fahrbahn befindende Boxen ignoriert werden. Erreicht der vordere Sensor einen gewissen Schwellenwert, überprüfen wir am linken Sensor, ob die linke Fahrbahn frei ist. Wenn dieser „Schulterblick“ sein okay gibt, wird das Überholmanöver eingeleitet.
Vorbereitend wird hierbei die linienbasierte Lenkung und der Parkerknoten vorübergehend blockiert. Anschließend beginnen wir mit der ersten von fünf Phasen.

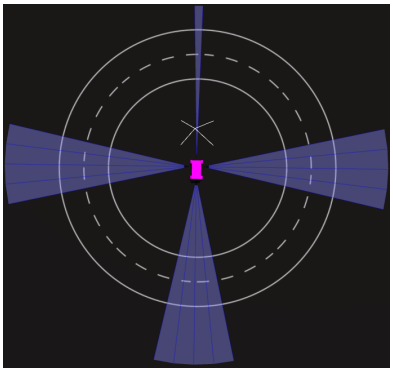

Abbildung 9: Verwendete ToF-Sensoren

3.3 5 Phasen zum Überholen

3.3.1 1. Phase: Spurwechsel

Hier wird vereinfacht angenommen, dass man die Strecke beim Spurwechsel im Überholvorgang als Gerade ansehen kann. In unserem Fall ist das ausreichend, weil das Fahrzeug nur grob auf die andere Spur kommen soll und ab da die linienbasierte Lenkung wieder ihre Aufgabe übernimmt. Dementsprechend kann man dann um die Überholgerade (rote Linie) ein Dreieck aufspannen. Um die gesuchte Strecke berechnen zu können, muss man nur 3 Werte kennen, was in unserem Fall die konstante effektive Spurwechseldistanz (grüne Linie), der 90°-Winkel γ und der 25° Lenkwinkel β sind.

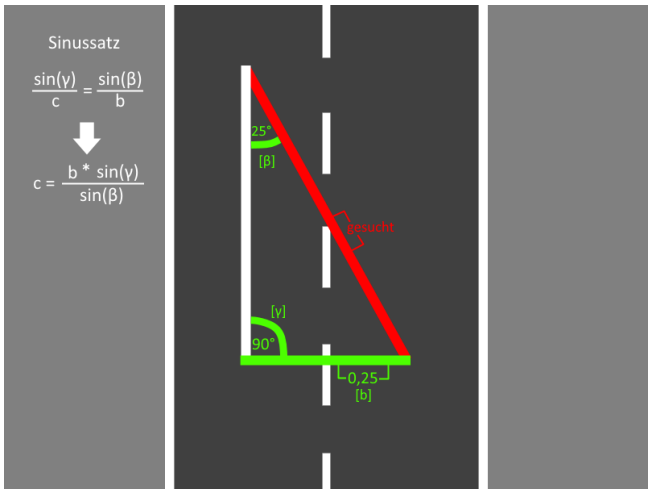

Abbildung 10: Spurwechselstrecke

Über den Sinussatz wird dann einfach die Länge der Überholgeraden berechnet. Aus der zurückzulegenden Strecke und der aktuellen Geschwindigkeit wird die Dauer des Spurwechsels bestimmt. Durch Ausprobieren haben wir festgestellt, dass die berechnete Dauer konstant um den Faktor 0,9 abweicht, daher multiplizieren wir unsere errechnete Dauer noch mit 0,9. Unsere Vermutung ist, dass diese Abweichung durch die Annäherung an die tatsächlichen Überholstrecke entstanden ist.

Der ganze Aufwand ist nötig, da man die Region-of-Interests nicht ohne weiteres einfach so versetzen kann, diese würden bei zu frühem Versetzen unter anderem die Mittellinie ungewollt erfassen, was zu einer falschen Fahrbahnbestimmung führen würde. Daher warten wir die Dauer des Spurwechsels, um dann in Phase 2 überzugehen.

3.3.2 2. Phase: Sichtbereich-Wechsel und Warten auf Box

An diesem Punkt muss zwischen den beiden Ansätzen unterschieden werden. In Ansatz A wird nach Ablaufen des in Phase 1 errechneten Delays, nun die Region-of-Interest für das Fahren auf der linken Spur angepasst. Zusätzlich muss jetzt auch unser Driveway-Factor von 1,25 zu 0,85. geändert werden. Hier würde man vermutlich 0,75 erwarten, aber durch Testen haben wir festgestellt, dass dieser Wert besser funktioniert. In Ansatz B wiederum muss beim Wechsel nur die maximale Suchdistanz angepasst werden.

Jetzt kann die linienbasierte Lenkung wieder übernehmen. Mit dem rechten Sensor warten wir darauf, die Box zu erkennen, damit wechselt das Fahrzeug dann in Phase 3.

3.3.3 3. Phase: Warten bis die Box verschwindet

Der Wagen ist nun auf der linken Spur und fährt an der Box vorbei. Gibt der rechte Sensor sein ok, dass die rechte Spur wieder frei ist, wechseln wir jetzt in die 4 Phase des Manövers.

3.3.4 4. Phase: Region-of-Interest-Wechsel

Die 4. Phase funktioniert jetzt spiegelverkehrt zur 2. Phase. Diese kümmert sich um den Wechsel der Region-of-Interests bzw. des Suchbereiches zurück auf die Startposition und das Zurücksetzen des Driveway-Factors zu 1,25. Nach Abschluss erfolgt der Wechsel in Phase 5.

3.3.5 5. Phase: Spurwechsel

Die 5. Phase ist nun die gespiegelte Version der 1. Phase. Diese kümmert sich jetzt abschließend um den Spurwechsel zurück nach rechts und die wieder Freigabe des Parkerknotens.

4 Einparken

4.1 Erkennen einer Parklücke in 4 Phasen

4.1.1 1. Phase: Erkennung der ersten Box

Zunächst wird auf der rechten Seite mittels des Time-of-Flight-Sensors nach einem Erstkontakt mit einer Box geschaut. Wird eine Box erkannt, wird zunächst der Überholer-Knoten blockiert und anschließend wird in Phase 2 gewechselt.

4.1.2 2. Phase: Erkennen der Lücke

In der zweiten Phase wird gewartet, bis das Auto an der ersten Box vorbei ist und der rechte Time-of-Flight eine Lücke erkannt hat. Ist diese erkannt, wird in die 3 Phase gewechselt.

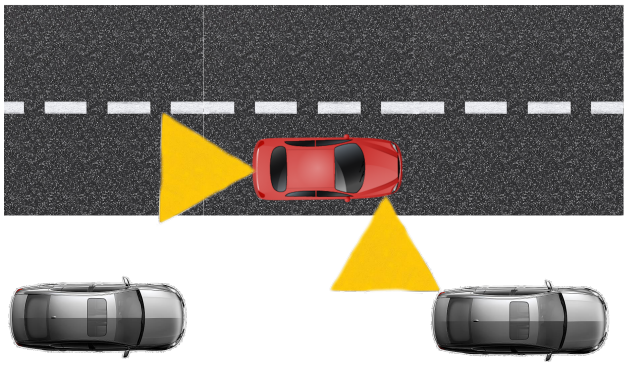

Abbildung 11: Parklücke erkennen

4.1.3 3. Phase: Erkennung der zweiten Box

Diese Phase funktioniert gleich wie die 1. Phase. Es wird wieder darauf gewartet, dass auf der rechten Seite eine Box erkannt wird. Ist das der Fall, wird die linienbasierte Lenkung blockiert und Phase 4 eingeleitet.

4.1.4 4. Phase: Einleiten des Parkmanövers

Das Auto wird hier angehalten und die Parklückenerkennung wird blockiert. Abschließend wird mit der 1. Phase des Einparkens weiter gemacht.

4.2 Einparken

Das Einparken ist vom Grundaufbau dem Überholen recht ähnlich und verwendet daher auch die gleichen Ansätze. Zunächst werden die Teilprobleme wieder in Phasen aufgeteilt.

4.2.1 1. Phase: Einfahren in die Lücke

Sobald der Wagen die Lücke gefunden hat, fährt er noch ein Stück weiter geradeaus,führt dann einen Spurwechsel im selben Schema vom Überholmanöver aus, nur dieses Mal rückwärts.

4.2.2 2. Phase: Erreichen der Parkposition

Sobald der Wagen dann in der Parklücke ist, fährt er so lange rückwärts, bis der hintere ToF-Sensor anschlägt, dann bleibt der Wagen kurz stehen und geht in die nächste Phase.

4.2.3 3. Phase: Ausparken

Hier fährt der Wagen dann einfach wieder mit einem einfachen Spurwechsel vorwärts aus der Parklücke zurück in die Fahrbahn. Es wird noch ein Timer von 20s aktiviert, der dann auch wieder den Parkplatz-Scan freigibt, das braucht es, da der Wagen ansonsten beim Rausfahren die zweite Box erkennen würde und wieder im Parkplatzsuchenmodus wäre.

5 Geschwindikeitsregler

Der Geschwindigkeitsregler für Ansatz A ist sehr simpel gehalten und ist nur außerhalb von den Manövern aktiviert. In den Manövern selbst sind feste Geschwindigkeiten vorgegeben. Die Geschwindigkeitsbestimmung basiert auf dem aktuellen Lenkwinkel, ist dieser unter einem Schwellwert, wird beschleunigt. Nach einer Wartezeit wird der Schwellenwert erneut geprüft und ggf. weiter beschleunigt, das Ganze wiederholt sich bis entweder die maximale Geschwindigkeit erreicht wurde oder der Schwellenwert überschritten ist. Ist der Schwellenwert überschritten, wird die Geschwindigkeit auf einen minimalen Wert gesetzt. Er sollte mittels hardgecodeter Zeitabstände zum richtigen Zeitpunkt auf der Geraden die Geschwindigkeit erhöhen und dann mittels weiterer Zeitabstände vor den Kurven wieder reduzieren.

6 Fazit

6.1 Zusammenfassung

Abschließen können wir sagen, dass wir ein autonomes Fahrzeug haben, welches die gestellten Aufgaben und Hindernisse lösen und überwinden kann. Das Fahrzeug kann selbstständig die Spur halten und dem Verlauf der Strecke folgen, ohne die Verkehrslinien ungewollt zu kreuzen. Es kann eine Parklücke auf der rechten Seite erkennen und dort ein- und wieder ausparken. Bei einem Hindernis auf der Fahrbahn erkennt das Fahrzeug dies und führt ein Überholmanöver durch. Dem Fahrzeug ist es möglich, ohne weitere Probleme eine Kreuzung zu überqueren. Eine selbstständige Regelung der Geschwindigkeit ist vorhanden.

6.2 Reflexion & Bewertung der Aufgabenstellung

Im Großen und Ganzen sind wir mit uns unserer Lösung des Projektes zufrieden und haben auch im Laufe des Projektes nicht annähernd damit gerechnet so weit zukommen und eine so solide Abgabe zu haben. Dennoch sind wir unseren Fehlern und Problemen bewusst. Angefangen bei der Region-of-Interest aus Ansatz A, die noch sehr starr und undynamisch ist. Besser wäre es, wenn diese sich den Kurven anpassen würde und auch dynamisch die Vorausschau anpassen könnte, um so besser über Kreuzungen zu kommen. Der rudimentäre Geschwindigkeitsregler müsste auch noch optimiert werden. Das Problem bei dem aktuellen Geschwindigkeitsregler ist, dass dieser unter anderem noch sehr sprunghaft ist und die Entscheidung mittels des aktuellen Lenkwinkels eigentlich schon zu spät ist, besser wäre vielleicht eine Vorausschau, die den späteren Lenkwinkel bestimmt. Die internen Eigenschaften der Software sind auch verbesserungswürdig. Aus designtechnischer Sicht ist es eher unschön, das Blockieren der einzelnen Knoten so dezentral zu machen, besser wäre ein richtige Statemaschine die diese Regelung übernimmt. Die Statemaschine könnte sich dann auch um die Regelung der einzelnen Phasen innerhalb der Manöver kümmern, was den Code deutlich wartbarer machen würde. Ansonsten sind uns nach Abgabe noch einige Redundanzen innerhalb des Codes aufgefallen, die man hätte vermeiden können. Allgemein kann man sagen, war unser Hauptproblem die Zeit. Wir haben während des Projektes zu viel für den Papierkorb entwickelt und zu viel herumexperimentiert bevor wir etwas Brauchbares hatten, was uns letztendlich zu viel Zeit gekostet hat.

6.3 Ausblick & das Kamera-Problem

Aus unserer Sicht würde es am meisten Sinn ergeben, mit Ansatz B weiterzuarbeiten. Dieser liefert ein deutlich robusteres und flüssigeres Endresultat, welches auch auf komplexere Anwendungen anpassbar wäre. An Stelle die Kamera jedoch über dem Fahrzeug fliegen zu lassen, ließe sich die gleiche Kameraperspektive auch durch vier Kameras erzeugen, welche in Kombination einen 360° Rundumblick generieren. Daraus ließe sich dann die Top-Down-View im selben Stile wie wir sie verwenden generieren. In der realen Welt kommt dieses System bei den meisten modernen Fahrzeugen in Form einer Einparkhilfe bereits zum Einsatz und wäre entsprechend leicht umsetzbar. Im Rahmen des cITIcars wäre es jedoch ein eher aufwendiger und platzintensiver Ansatz.

Anhang

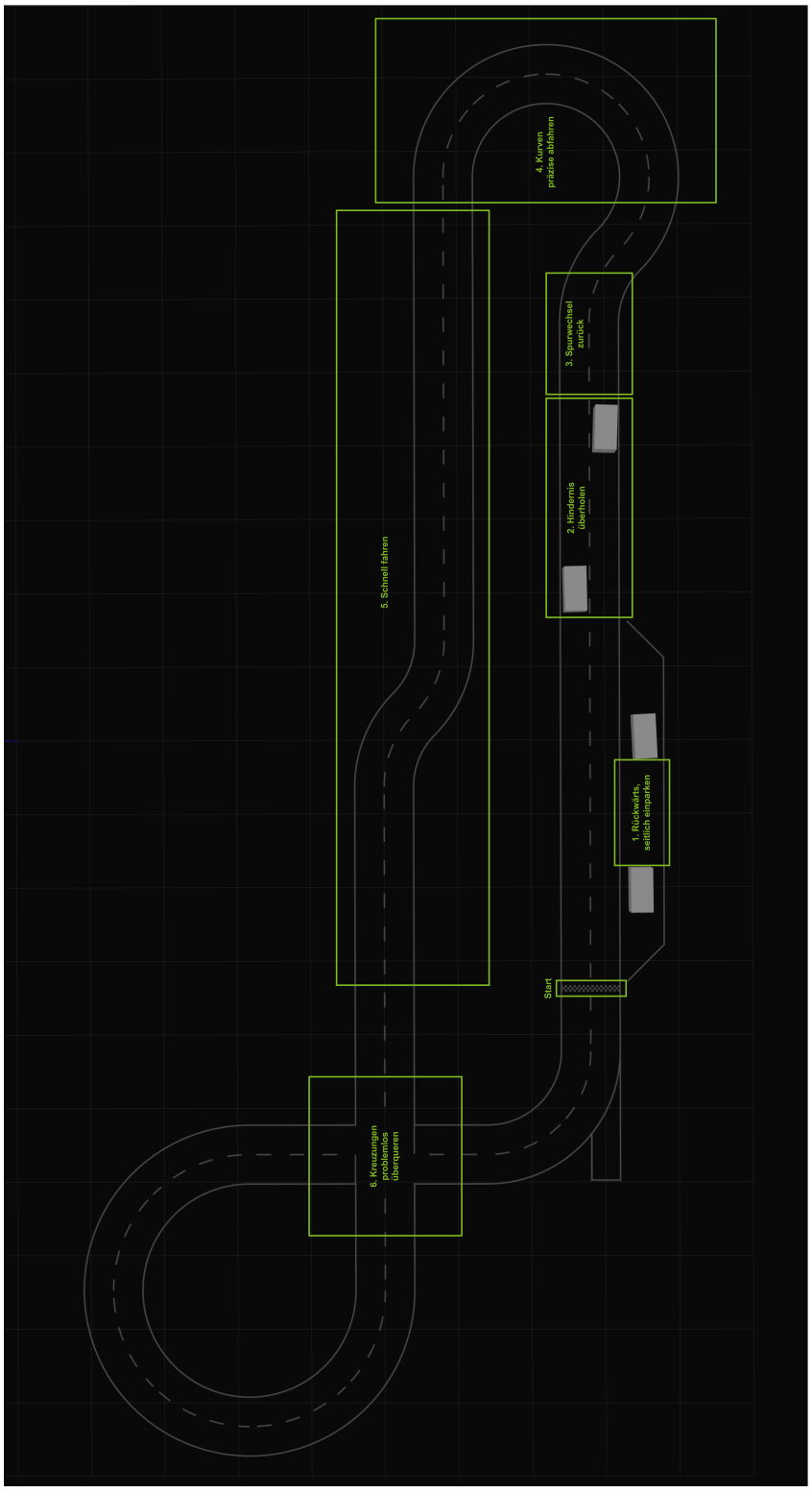

Abbildung 12: zu überwindende Teststrecke

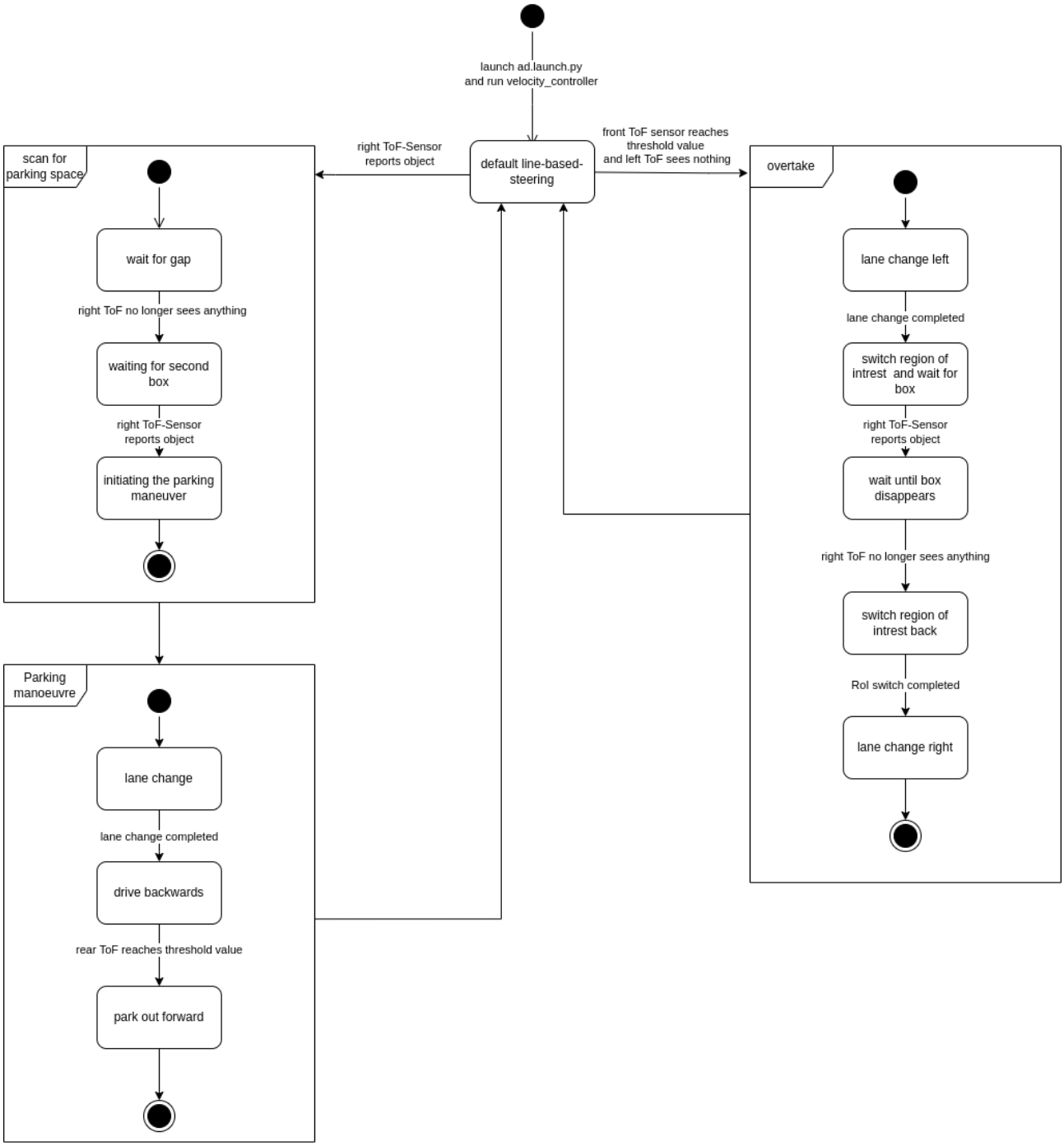

Abbildung 13: Zustandsmaschine unserer Lösung

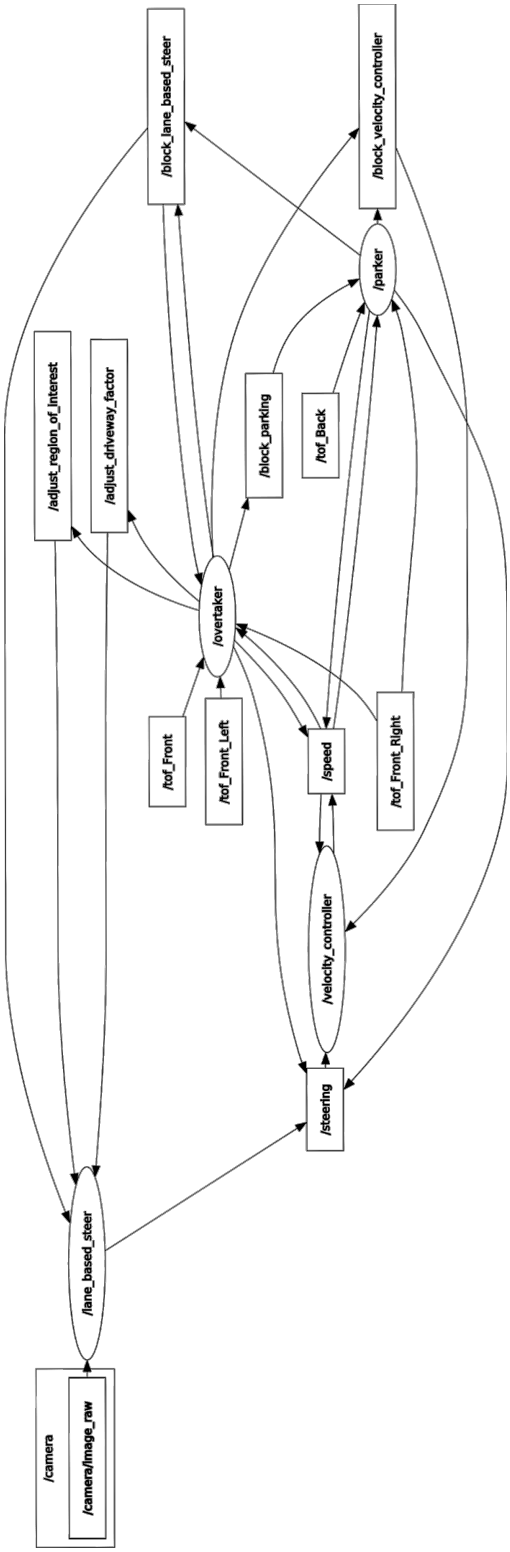

Abbildung 14: Übersicht über Knoten und Topics

Ansatz	**Verzeichnis**	**starten des autonomen Fahrzeugs**	**starten des Geschwindigkeitsreglers**
Ansatz A	../autonomous_driving	`ros2 launch autonomous_driving ad.launch.py`	`ros2 run autonomous_driving velocity_controller`
Ansatz B	../autonomous_driving_v2	`ros2 launch autonomous_driving_v2 ad.launch.py`	`ros2 run autonomous_driving_v2 velocity_controller`

Tabelle 1: Übersicht über Lösungsansätze

Abbildungsverzeichnis

Tabellenverzeichnis

Literatur

Verwendete Tutorials:
https://www.youtube.com/watch?v=KEYzUP7-kkU[1]
https://www.youtube.com/watch?v=yvfI4p6Wyvk[1]
https://www.youtube.com/watch?v=eLTLtUVuuy4[1]
https://www.youtube.com/watch?v=j4el1XARYSo[1] [2]

[1] zuletzt am 05.08.2022 geprüft
[2] nicht in Finallösung enthalten