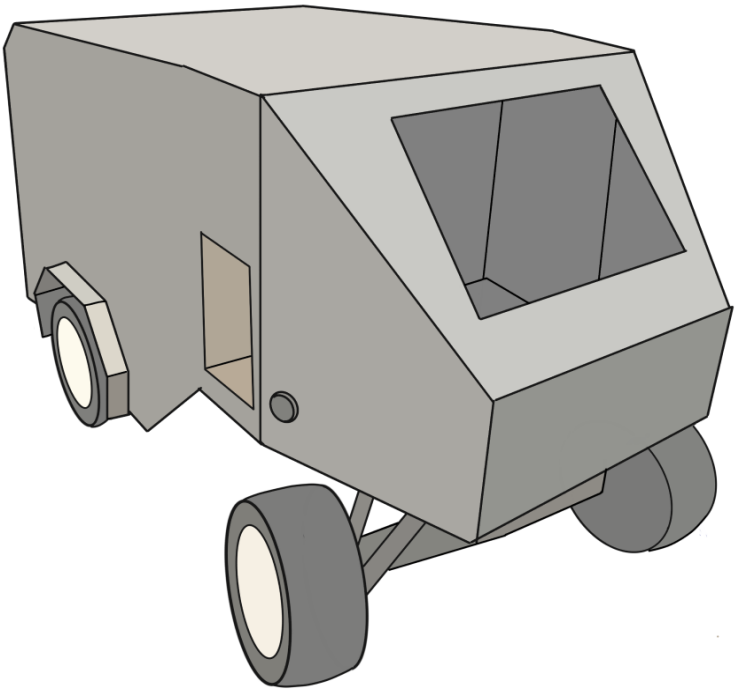
Pizza Roboter: der autonome Lieferroboter

Ein Projekt von Manuel Selch (17 J.)
Erarbeitungsort: privat zu Hause und teilweise in der Schule
Projektbetreuer: Dr. Andreas Stingl (Lehrer der Robotik AG)
JugendForscht 2023 Oberfranken

Fachgebiet Informatik: Ein selbst gebauter und programmierter Roboter mit autonomer Navigation durch Routengenerierung, Fahrspur-Assistenten, Ampelerkennung und Hindernisvermeidung



Kurzfassung

Grundidee des Projektes ist, verschiedene Methoden zur autonomen Navigation zu testen, um am Ende autonom auf dem Verkehrsübungsplatz Bayreuth fahren zu können.

Dafür wird ein Lieferroboter entwickelt, der durch eine LiDAR Kamera und unterschiedliche Sensoren einer Straße folgen kann, Kreuzungen erkennt, an diesen korrekt abbiegt um zur Zielposition zu gelangen, vor Hindernissen bremst, sowie rote Ampeln erkennt um stehen zu bleiben.
Hierzu werden verschiedene Methoden entwickelt und getestet. Als Unterstützung dient eine Simulation, um verschiedene Szenarien leicht analysieren zu können.
Der Zentrale Kern der Arbeit ist hierbei die Entwicklung einer möglichst genauen Selbst-Lokalisierung.

Inhaltsverzeichnis

Abbildungsverzeichnis

1. Einleitung

1.1 Ideenfindung

Zur Zeit gibt es immer wieder News über neue Meilensteine im Bereich autonomes Fahren, beispielsweise erste autonome Shuttlebusse in Kronach und Hof [1].
Dies ist ein hochaktuelles und zugleich interessantes Thema. Viele moderne Autos haben auch bereits erste Assistenz-Systeme verbaut, welche den Fahrer unterstützen.
Damit ein autonomes Fahrzeug selbstständig navigieren kann, muss sich dieses zuerst einmal lokalisieren. Je präziser diese Lokalisierung ist, desto leichter fällt auch das anschließende Navigieren. Daher liegt der Fokus dieser Arbeit auf der Untersuchung verschiedener Methoden zur Lokalisierung eines autonomen Autos.
Es soll die Frage geklärt werden, welche Möglichkeiten und Sensoren es hierzu gibt und wie diese zu einer höheren Genauigkeit der Lokalisierung führen.
Als Prototyp wird hierzu ein Lieferroboter entwickelt, der mit verschiedenen Sensoren ausgestattet ist, um seine Umgebung wahrnehmen zu können.
Zusätzlich wird eine Simulation genutzt, um die verschiedenen Techniken der Lokalisierung leichter vergleichen zu können.

1.2 Wie kann so eine Lösung realisiert werden?

Ein autonomes Auto muss mit einer Vielzahl verschiedener Sensoren ausgestattet sein, die gleichzeitig die Umgebung wahrnehmen und so die Position des Autos bestimmen können. Durch den Kalman Filter können die Eigenschaften dieser Sensoren optimal miteinander verbunden werden.
Die Lokalisierung wird an einem eigenen autonomen Lieferroboter getestet, wobei schnell die Komplexität dieses Projektes klar geworden ist, und daher zusätzlich eine Simulation herangezogen wurde, in der die verschiedenen Sensoren systematisch getestet und ausgewertet werden können.

2. Theorie

2.1 Grundlegendes Verständnis der Lokalisierung

Je genauer ein Roboter seine globale Position bestimmen kann, desto leichter ist auch dessen Navigation. Denn ohne eine genaue Position zu kennen, ist es schwierig, einem vorher berechneten Pfad zu folgen. Für eine möglichst genaue Lokalisierung reicht es daher nicht, sich auf lediglich einen Sensor zu verlassen. Ein GPS Modul ist beispielsweise praktisch, um eine grobe globale Position zu bestimmen, reicht aber nicht aus, um einer geraden Straße mit einer Genauigkeit von wenigen Zentimetern folgen zu können. Hierzu werden genauere Sensoren benötigt, deren Vor- und Nachteile im Folgenden untersucht werden sollen.
Grundlegend werden mit einem Odometrie Sensor die Radumdrehungen gemessen. Bei bekanntem Umfang der Räder lässt sich so die zurückgelegte Strecke messen. Für eine optimale Genauigkeit müsste hier zusätzlich noch der Reifenwechsel zwischen Sommer- und Winterreifen berücksichtigt werden. Mittels einem Gyroskop-Sensor, der Winkel und Beschleunigung misst, lässt es sich relativ gut geradeaus fahren. Für zusätzliche Genauigkeit wird noch eine Kamera genutzt, die ihr Live-Bild mit den Trainingsdaten des Teach-Repeat Systems (vgl. *Kapitel 2.2.3 Teach-Repeat System)*, abgleicht, sowie per LiDAR präzise Abstände zu den einzelnen Objekten misst. Eine weitere Idee ist das Berücksichtigen der Ackermann Geometrie des Roboters. Da der Roboter sich nicht auf der Stelle drehen kann, sondern durch den Lenkeinschlag stattdessen einen Kreis beschreibt, kann mit etwas Trigonometrie aus der Schule anhand des Lenkeinschlages und der Umdrehungen der Räder die relative Position auf diesem Kreis berechnet werden (vgl. *Kapitel 2.2.1 Ackermann Geometrie*).
Am wichtigsten ist jedoch die Fusion der verschiedenen Sensorwerte. Verschiedene Sensoren haben

verschiedene Fehlerraten. Doch durch das Vereinen verschiedener Sensoren wird der Gesamtfehler reduziert. Dies wird durch den Kalman Filter erreicht (vgl. *Kapitel 2.2.3 Sensor Fusion*).

2.1 Lokalisierung durch Kalman Filter

2.1.1 Aufbau des Kalman Filters

Nun soll geklärt werden, wie mithilfe des Kalman Filters [3] der Roboter durch verschiedene Sensorwerte getrackt werden kann. Das Problem von Sensoren ist die Unsicherheit und Fehlerquote, welche diese mit sich bringen. Aber durch den Kalman-Filter kann diese Unsicherheit gemindert werden, indem die gemessenen Informationen durch eine optimierte Kombination ein zuversichtliches Ergebnis liefern. Der Kalman-Filter bestimmt dabei, welchen Informationen aktuell am meisten vertraut werden sollen und passt so die Kombination der Informationen an.
Grundsätzlich besteht der Kalman Filter aus drei Schritten: Zuerst wird eine **Vorhersage** basierend auf dem Wissen der letzten Roboter Position dessen Odometrie getroffen. Danach folgt die **Messung**. Es werden die Sensorwerte abgerufen und mit der Vorhersage verglichen. Als letztes folgt das **Update**. Hier wird das Wissen über die Roboterposition basierend auf der Vorhersage und der Sensorwerte aktualisiert. Diese drei Schritte werden anschließend immer wieder wiederholt. Dadurch sollte die Lokalisierung mit der Zeit immer genauer werden.

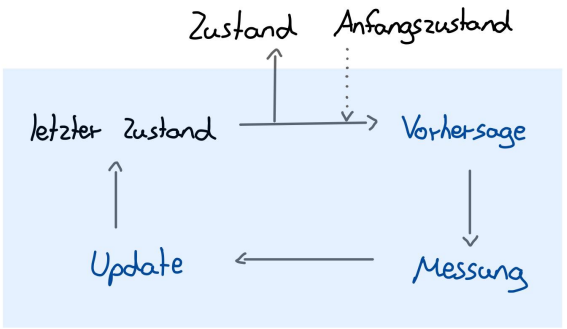


Abbildung 01: Schaubild zum Kalman Filter

Abbildung 03 stellt ein Schaubild dar, welches die grundlegende Funktionsweise des Kalman-Filters zeigt: Zuerst muss ein Anfangszustand definiert werden. Dieser enthält eine ungefähre Position des Roboters. In der Vorhersage wird anschließend der neue Zustand mit Hilfe der Odometrie und der Lenkung berechnet (vgl. *Kapitel 2.2.1 Ackermann Geometrie)*. Denn diese Methode ist kurzzeitig sehr exakt, die Fehlerquote erhöht sich jedoch mit der Zeit.

Um die langfristige Fehlerquote zu mindern, werden nun die Messwerte der weiteren Sensoren abgerufen, um die Vorhersage entsprechend zu aktualisieren. Der Sinn dieser Sensoren ist, langfristig eine höhere Genauigkeit zu liefern, diese Sensoren dienen somit zur Korrektur der Position.
Zur Aktualisierung wird zunächst das “Kalman Gain” berechnet, welches den Fehler der Vorhersage mit dem Fehler in der neuen Messung vergleicht. Wenn das “Kalman Gain” groß ist, wird der neuen Messung mehr vertraut, da dann der Fehler der Messung kleiner ist als der Fehler in der Vorhersage. Wenn das „Kalman Gain” stattdessen klein ist, wird der Messung weniger vertraut, da dann der Fehler in der Messung größer ist als der Fehler in der Vorhersage.
Dadurch kann bestimmt werden, wie stark die Vorhersage von der Messung beeinflusst werden soll. Mit jeder Iteration wird der Fehler in der Schätzung im Vergleich zum Fehler in der Messung kleiner und mit der Zeit weniger Wert auf die Messwerte gelegt. Dadurch verfeinert sich die Lokalisierung mit jeder Iteration des Kalman-Filters.
Im Folgenden werden die drei Schritte des Kalman Filters am Beispiel des autonomen Roboters genauer erläutert.

2.1.2 Vorhersage

In der Vorhersage wird eine Funktion benötigt, welche die Veränderung des Zustands seit dem letzten Vorhersage-Schritt beschreibt. Hierzu wird zunächst die normale Bewegungsgleichung genutzt, wobei in Kapitel 2.2.1 “Ackermann Geometrie” noch eine bessere Methode besprochen wird.
Um den neuen Zustand vorhersagen zu können, muss zunächst definiert werden, was der Zustand des Roboters ist.
Der Zustand beschreibt alle Informationen, die für eine vollständige Beschreibung der Eigenschaften eines Systems nötig sind. In unserem Fall handelt es sich um ein autonomes Fahrzeug, welches sich in X- und in Y- Richtung bewegen und um die eigene Achse (α) rotieren kann.
Die Zustandsmatrix X könnte somit wie folgt aussehen:

$$X = \begin{bmatrix} x \\ y \\ \alpha \end{bmatrix} \quad (1)$$

Nun ist dieser Zustand aber niemals korrekt, sondern lediglich eine Schätzung. Der Fehler dieser Vorhersage wird als Kovarianz-Matrix mit dem Parameter P abgebildet:

$$P_{k|k-1} = P_{k-1} + Q_{k-1} \quad (2)$$

Q ist hier eine weitere Kovarianz-Matrix für das Rauschen des Zustands, damit P nie null werden kann. Denn der Kalman Filter lässt mit jeder Iteration P immer kleiner werden, denn der Vorhersage soll mit der Zeit immer mehr vertraut werden, wobei diese dennoch immer ein wenig von der Messung korrigiert werden soll.

2.1.3 Messung

Als nächstes soll die Vorhersage durch weitere Sensordaten korrigiert werden. Der Roboter besitzt hierzu einen GPS Sensor. Es können aber auch andere Sensoren genutzt werden (vgl. Kapitel 2.2 Optimierung der Lokalisierung), wichtig ist nur die Eigenschaft dieser Sensoren: Zur Korrektur der Vorhersage sind nur Sensoren geeignet, die keinen Drift aufweisen.
Der GPS Sensor gibt den aktuellen Längen- und Breitengrad zurück, wodurch die globale Position bekannt ist. Doch diese Angaben müssen noch in ein einheitliches Koordinatensystem transformiert werden. Als einheitliches Koordinatensystem wird eine 2D Karte mit den Achsen X und Y in Metern gewählt. Den Ecken dieser Karte werden jeweils die entsprechenden GPS Punkte zugewiesen, um GPS Messungen in dieses Koordinatensystem zu transformieren:

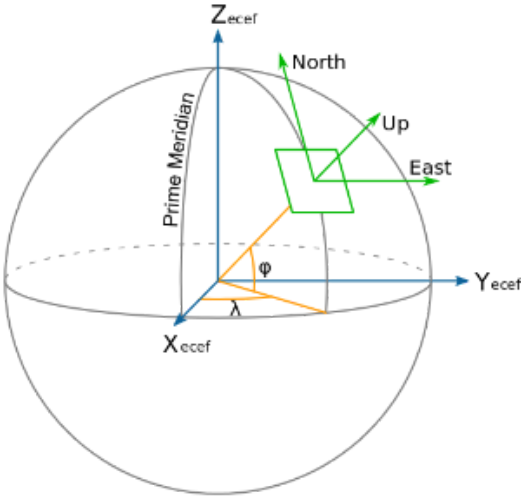


Abbildung 02: Konvertieren zwischen GPS Position und einem lokalen Koordinatensystem [7]

Abbildung 02 zeigt, wie das GPS-Signal des Sensors in ein einheitliches Koordinatensystem transformiert wird, um leichter mit Sensor Fusion arbeiten zu können. Das neue Koordinatensystem besitzt nun Achsen mit Einheit in Metern statt Längen- und Breitengrad.

2.1.3 Update

Die eigentliche Mathematik des Kalman-Filters steckt im letzten Schritt: dem Update. Hier soll die bisherige Vorhersage durch die Messung korrigiert werden. Das Problem der Messung liegt in dem hohen Rauschen. Es darf sich somit nicht voll auf die Messung verlassen werden, sondern die Messung soll die Vorhersage nur zu einer bestimmten Gewichtung beeinflussen. Die Gewichtung wird durch das “Kalman Gain” bestimmt. Die Formel hierzu lautet:

$$K_k = \frac{P_{k|k-1}}{P_{k|k-1} + R_k} \quad (3)$$

P steht hier weiterhin für die Unsicherheit des Zustandes bzw. der Vorhersage, während *R* die Unsicherheit der Messung darstellt. Je größer der erwartete Fehler der Messung, desto größer ist *R*, desto kleiner ist somit das Kalman Gain, denn es soll sich weniger auf die Messung verlassen werden.
Der Zustand X kann nun abhängig vom Kalman Gain *K* aktualisiert werden:

$$X_k = X_{k|k-1} + K_k * (Y_k - X_{k|k-1}) \quad (4)$$

Nun muss noch die neue Kovarianz-Matrix des Zustandes berechnet werden.
Diese soll mit jeder Iteration immer kleiner werden, um der Vorhersage mehr zu vertrauen.

$$P_k = (1 - K_k) * P_{k|k-1} \quad (5)$$

Denn anfangs ist das Kalman Gain groß, dadurch wird die Kovarianz P umso kleiner und somit wird wiederum das Kalman Gain immer kleiner.

2.2 Optimierung der Lokalisierung

2.2.1 Ackermann Geometrie

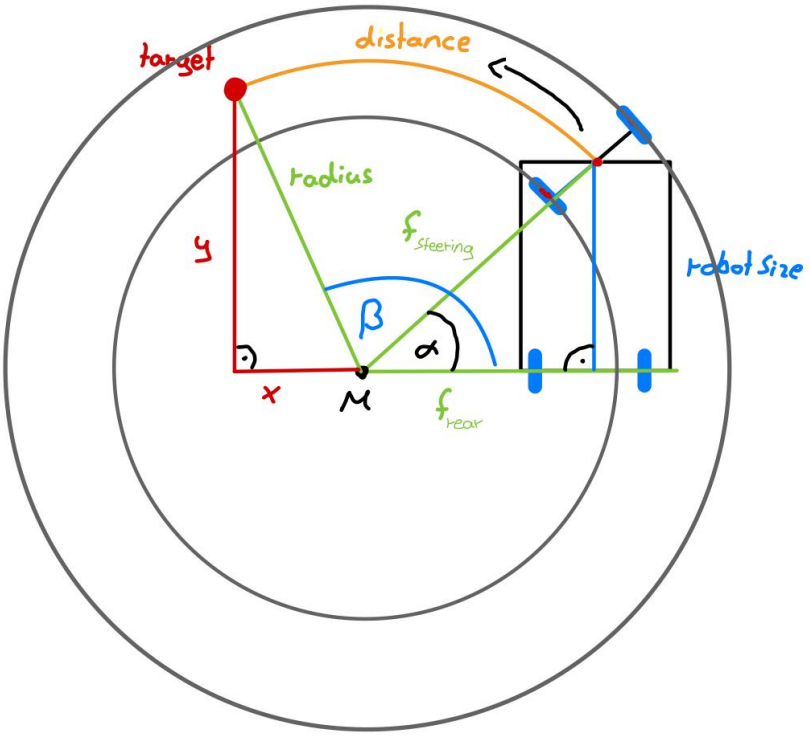


Abbildung 03: Skizze zur Visualisierung der Ackermann Geometrie

Wie in *Abbildung 03* zu erkennen ist, kann sich der Roboter durch die Ackermann Geometrie nicht um die eigene Achse drehen, sondern dessen Kurve beschreibt einen Kreis. Die Berücksichtigung dieser Information führt zu einer höheren Genauigkeit der Lokalisierung, als die Position lediglich durch die Bewegungsgleichung (3) anhand der Geschwindigkeit und Beschleunigung zu berechnen:

$$s = s_0 + vt + \frac{1}{2}at^2 \quad (6)$$

Um nun die relative Position auf diesem Kreis zu berechnen sind einige Schritte nötig:
Zuerst muss der Mittelpunkt des Kreises berechnet werden. Dies ist der Schnittpunkt der Geraden $\mathbf{f}_{\mathbf{steering}}$ und $\mathbf{f}_{\mathbf{rear}}$.
Die Gerade $\mathbf{f}_{\mathbf{steering}}$ verläuft durch den Lenkeinschlag der Vorderräder, die Gerade $\mathbf{f}_{\mathbf{rear}}$ durch die Hinterachse.

Die Steigung der Geraden $\mathbf{f}_{\mathbf{steering}}$ entspricht dem Tangens von dessen Winkel, welcher die Summe des aktuellen Roboter-Winkels und des Lenkeinschlages ist:

$$m_{steering} = tan(steering + currentRobotAngle) \quad (7)$$

Die Gerade muss durch den Drehpunkt der Vorderräder ($\mathbf{P}_{\mathbf{steering}}$) verlaufen. Dadurch lässt sich anschließend der Y-Achsenabschnitt dieser Geraden berechnen:

$$Psteering.x = currentPos.x - sin(currentRobotAngle) * robotSize \quad (8)$$
$$Psteering.y = currentPos.y + cos(currentRobotAngle) * robotSize \quad (9)$$

$$y = m * x + t \quad (10)$$
$$t = y - m * x \quad (11)$$
$$t_{steering} = Psteering.y - m_{steering} * Psteering.x \quad (12)$$

Dasselbe Schema wird zur Berechnung der Geraden $\mathbf{f_{rear}}$ angewendet. Da der Ursprung des lokalen Koordinatensystems des Roboters auf der Hinterachse liegt, entspricht der Punkt $\mathbf{P_{rear}}$ der aktuellen Roboterposition:

$$m_{rear} = tan(currentRobotAngle) \quad (13)$$
$$P_{rear} = currentPos \quad (14)$$
$$t_{rear} = P_{rear}.y - m_{rear} * P_{rear}.x \quad (15)$$

Nun sind beide Geraden bekannt und es kann der Schnittpunkt durch das Gleichsetzen der Funktionen gebildet werden, welcher dem Mittelpunkt **M** des Kreises entspricht. Die Y-Koordinate des Schnittpunktes erhält man durch einsetzen der errechneten x-Koordinate in einer der beiden Funktionen.

$$f_{steering} = f_{rear} \quad (16)$$
$$m_{steering} * x + t_{steering} = m_{rear} * x + t_{rear} \quad (17)$$
$$x = \frac{t_{steering} - t_{rear}}{m_{rear} - m_{steering}} \quad (18)$$
$$y = m_{rear} * x + t_{rear} \quad (19)$$
$$M = [x, y] \quad (20)$$

Nun, da der Mittelpunkt **M** bekannt ist, kann der Radius **r** des Kreises durch den Satz des Pythagoras berechnet werden:

$$\Delta x = M.x - currentPos.x \quad (21)$$
$$\Delta y = M.y - currentPos.y \quad (22)$$
$$r^2 = \Delta x^2 + \Delta y^2 \quad (23)$$
$$r = \sqrt{\Delta x^2 + \Delta y^2} \quad (24)$$

Wenn nun der Roboter diesem Kreis einer Strecke **distance** entlangfährt, entspricht dies dem gefahrenen Kreisbogen und da der Radius des Kreises bekannt ist, kann daraus der neue Winkel **alpha** bestimmt werden, wobei der aktuelle Winkel **currentRobotAngle** noch addiert werden muss:

$$U = 2\pi * r \quad (25)$$
$$distance = U * alpha / 360° \quad (26)$$
$$alpha = \frac{distance}{U} * 360° + currentRobotAngle \quad (27)$$

Der letzte Schritt ist die X- und Y-Koordinate des neuen Winkels durch Trigonometrie zu bestimmen:

$$x = cos(alpha) * r + M.x \quad (28)$$
$$y = sin(alpha) * r + M.y \quad (29)$$

Sind die Odometrie-Messwerte und die Angaben des Lenkeinschlages exakt, lässt sich damit schon einmal ziemlich gut die Position bestimmen. Interessant an diesem Ansatz ist, dass statt eines Gyroskop-Sensors lediglich der Lenkeinschlag benötigt wird. Aber selbst in der Simulation wird der

Fehler mit zunehmender Strecke schnell größer. Daher müssen noch weitere Sensorwerte benutzt werden, um diesen Fehler zu reduzieren.

2.2.2 Fahrspuren-Assistent

Nun, wenn die Position des Roboters berechnet werden kann und eine globale Route des Roboters zum nächsten Ziel bekannt ist, beginnt das Navigieren zu den einzelnen Wegpunkten.
Zum Erreichen der einzelnen Wegpunkte ist zunächst die Idee, der Straße über einen Fahrspur Assistenten zu folgen, bis der nächste Wegpunkt erreicht ist, dort entsprechend der Route abzubiegen und wieder der Straße bis zum nächsten Wegpunkt zu folgen. Der Roboter besitzt hierzu zwei Kameras. Die erste Kamera beobachtet das Umfeld vor dem Roboter. Damit lässt sich der weitere Verlauf der Straße auswerten, wodurch sich schneller dynamische Hindernisse oder Abbiegungen erkennen lassen. Eine weitere Kamera die direkte Straße vor dem Roboter und lässt sich zur Erkennung von Fahrspuren nutzen. Werden über diese Kamera die beiden Fahrspur-Linien erkannt, kann so der Roboter entsprechend gegensteuern und auch bei Kurven sicher lenken. Damit der Roboter auch rückwärts fahren könnte, wären noch weitere Sensoren nötig, die das Umfeld hinter dem Roboter auswerten.
Die Fahrspuren-Erkennung ist mit Python und OpenCV [4] umgesetzt, einer Bibliothek mit Algorithmen zur Bildverarbeitung. Zuerst wird ein aktueller Frame der Kamera ausgelesen. Bevor Fahrspuren erkannt werden können, werden per „Binary Thresholding“ weiße Pixel gefiltert. Dabei definiert man einen Schwellwert und erhält ein Schwarz/Weiß-Bild. Pixel, die den Schwellwert überschreiten, sind weiß, alle anderen Pixel schwarz. Dadurch können weiße Linien erkannt werden, auch wenn sie leicht gräulich sind. Um den Rechenaufwand gering zu halten, wird eine „region of interest“ (roi) definiert. Das heißt, es wird nur ein spezieller Bereich untersucht, wobei zur Erkennung von zwei Fahrspuren auch zwei Bereiche untersucht werden müssen. Hierzu dienen zwei Rechtecke, jeweils links bzw. rechts im Frame, wie in *Abbildung 04* zu erkennen.
In beiden Rechtecken werden alle weißen Pixel, die vertikal übereinander liegen, addiert und der Durchschnitt gebildet. Dadurch erhält man eine Liste mit dem Durchschnitt der weißen Pixel von links nach rechts. Nun wird der Index mit dem höchsten Durchschnitt gesucht, um herauszufinden, wo in diesem Rechteck die weiße Linie liegt.
Die Differenz zur Mitte des Rechtecks entspricht dem relativen Versatz des Roboters zur Mitte der Fahrspur.
In einer vorher gespeicherten Karte wird die nächste Position des Roboters zur Straße gesucht und über Trigonometrie der Versatz zur Mitte addiert. Dadurch erhält man eine neue Schätzung der Position des Roboters. Dieser Schätzung wird dabei aber auch nicht blind vertraut, sondern als weitere Sensor-Messung dem Kalman Filter zugeführt.

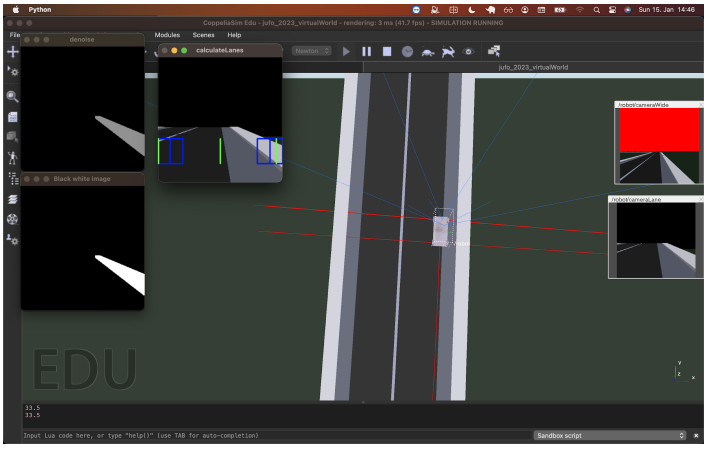
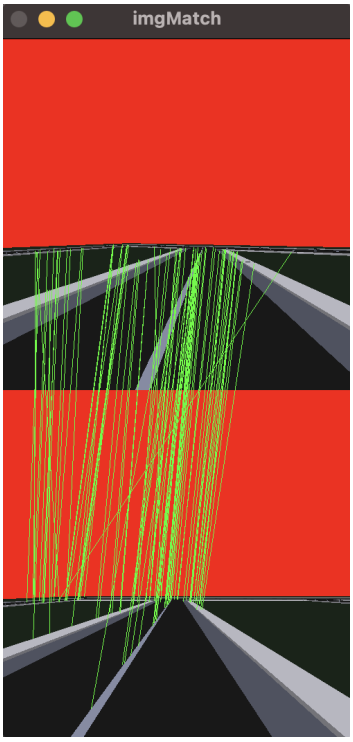


Abbildung 04: Fahrspuren-Assistent in der Simulation

2.2.3 Teach-Repeat System

Die grundlegende autonome Lokalisierung und Navigation ist nun umgesetzt und untersucht. Doch dabei lassen sich noch viele Fehler und Ungenauigkeiten erkennen. Vor allem das Abbiegen ist noch sehr ungenau. Bisher dreht sich der Roboter im Grunde nur um 90 Grad oder fährt geradeaus weiter. Doch schon bei kleinen Kreuzungen ist dies zu ungenau und muss optimiert werden.
Nun sollen dem Roboter einzelne Strecken beigebracht werden, damit er diese später exakter und somit sicherer befahren kann. Dadurch kann sowohl die Navigation durch den gespeicherten Pfad verbessert werden, als auch die Lokalisierung durch die gespeicherten Kamera Bilder der Umgebung. Während der "Teaching"-Phase wird der Roboter ferngesteuert und dieser speichert dabei die einzelnen Wegpunkte der gefahrenen Route ab. Jeder Wegpunkt beinhaltet die Position des Roboters und ein aktuelles Bild sowie Tiefeninformationen der LiDAR Kamera. Später in der “Repeating”-Phase fährt der Roboter die Strecke autonom ab. Hierbei hilft ein Pfad-Tracking-System. Der Vorteil der beigebrachten Route liegt in den gespeicherten Kamerabildern. Denn an jedem Wegpunkt kann, siehe *Abbildung 05*, das gespeicherte Kamerabild mit dem Livebild verglichen werden. Merkmale werden auf den beiden zu vergleichenden Bildern erkannt und dadurch kann überprüft werden, ob sich der Roboter rechts oder links vom eigentlichen Weg befindet und kann entsprechend gegensteuern.
Das heißt, wenn der Roboter annimmt an einer gewissen Position zu sein, die Kamera aber zeigt, dass sich der Roboter zu weit links bzw. rechts von dieser Position befindet, kann somit die Position korrigiert und die Lokalisierung somit optimiert werden.



Es lässt sich somit erkennen, wie weit die Position des Roboters nach links und nach rechts versetzt ist, aber noch nicht, ob er sich vor oder schon hinter dem Wegpunkt befindet. Hierzu war zuerst gedacht, die Größe von einzelnen Objekten zu vergleichen, da weit entfernte Objekte durch die Perspektive kleiner sind. Dies funktioniert jedoch nur bei sehr großen Unterschieden, wenn sich der Roboter bspw. mehrere Meter verfahren hätte, wobei auch hier die Kamera nur minimale Änderungen in der Größe der Objekte feststellen kann. Mittlerweile besitzt der Roboter eine LiDAR Kamera. Mit dieser ist es möglich, die Entfernung von Objekten mit einer höheren Genauigkeit zu erkennen. Hierzu werden die bereits auf beiden Bildern erkannten Merkmale genutzt. Zwischen jedem Merkmal auf dem Trainings- und Livebild wird die Differenz der Entfernung berechnet und daraus der Durchschnitt gebildet.

Abbildung 05: Identifizierung gleicher Merkmale in zwei Frames

3. Implementierung

3.1 Simulation

Um verschiedene Szenarien mit unterschiedlichen Lokalisierungs-Methoden vergleichen zu können, wird die Simulation CoppeliaSim [2] verwendet. Hierin wurden der Verkehrsübungsplatz Bayreuth sowie der Roboter rekonstruiert, um ähnliche Verhältnisse zu schaffen wie in der Realität. Somit lassen sich die verschiedenen Methoden in der virtuellen Welt einzeln oder zusammen testen und optimieren.
Die Simulation lässt sich leicht mit Python steuern. Ein Python-Script greift das aktuelle Kamerabild ab, nimmt Berechnungen vor und gibt anschließend der Simulation Befehle für Motorgeschwindigkeit und -Lenkung. Dies ist insofern praktisch, da der Roboter ebenfalls mittels Python programmiert wird und somit lediglich eine Schnittstelle zur Simulation und zur Auswertung der simulierten Sensoren implementiert werden muss.

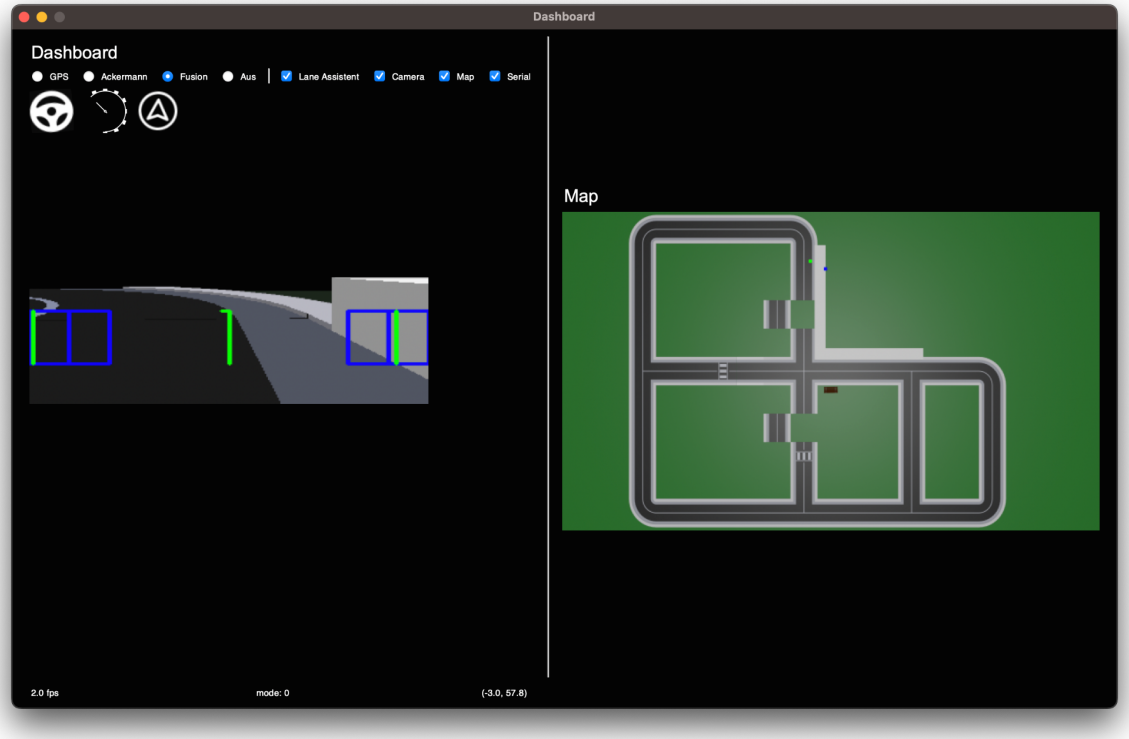


Abbildung 06: Dashboard in der Simulation

Abbildung 06 zeigt die laufende Simulation und die Auswertung der Sensoren mittels Python. In der rechten Hälfte der Abbildung sieht man eine Karte der Simulation, welche sowohl die echte Position (kleiner grüner Punkt), als auch die geschätzte Position (kleiner blauer Punkt) darstellt.

3.2 Echter Prototyp

Gleichzeitig wurde die Lokalisierung an einem eigenen autonomen Roboter in der realen Welt getestet. Dieser wurde im Rahmen der Jugend Forscht Arbeit 2021/22 selbst konstruiert und umgesetzt [8]. Im Folgenden kann nun an diesem Roboter die Lokalisierung getestet werden.

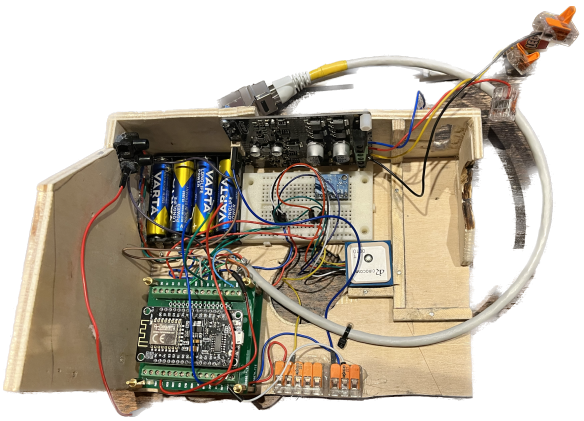


Abbildung 07: Zentrale des Roboter

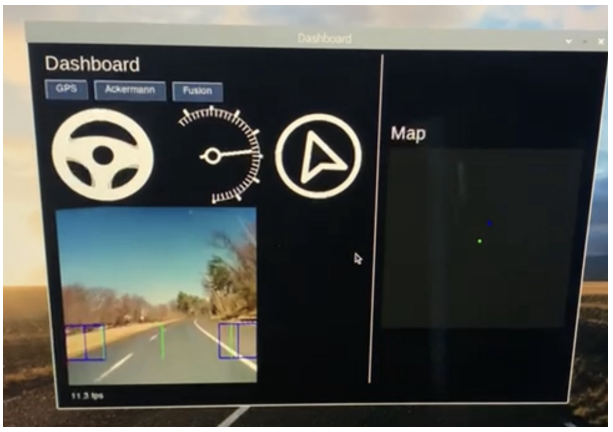


Abbildung 08: Dashboard in der realen Welt

Abbildung 07 zeigt einen Teil des Roboters, welcher für die Auswertung und Interpretation der Sensoren zuständig ist. Auf einem Raspberry Pi (*in Abbildung 07 nicht zu sehen*) läuft die komplette, wobei dieser die Sensordaten von einem Microcontroller bekommt. Der Microcontroller schickt erst die aktuellen Sensorwerte, sobald sich diese verändert haben, um die Performance zu erhöhen. Auf dem Raspberry Pi wird zudem ein Dashboard (*Abbildung 08)* mit allen wichtigen Daten für die Lokalisierung dargestellt.

4. Versuche

4.1 Versuch Lokalisierung

Im folgenden Versuch sollen die Methoden zur Lokalisierung miteinander verglichen werden, sowie der Einfluss der verschiedenen Parameter des Kalman Filters gezeigt werden.

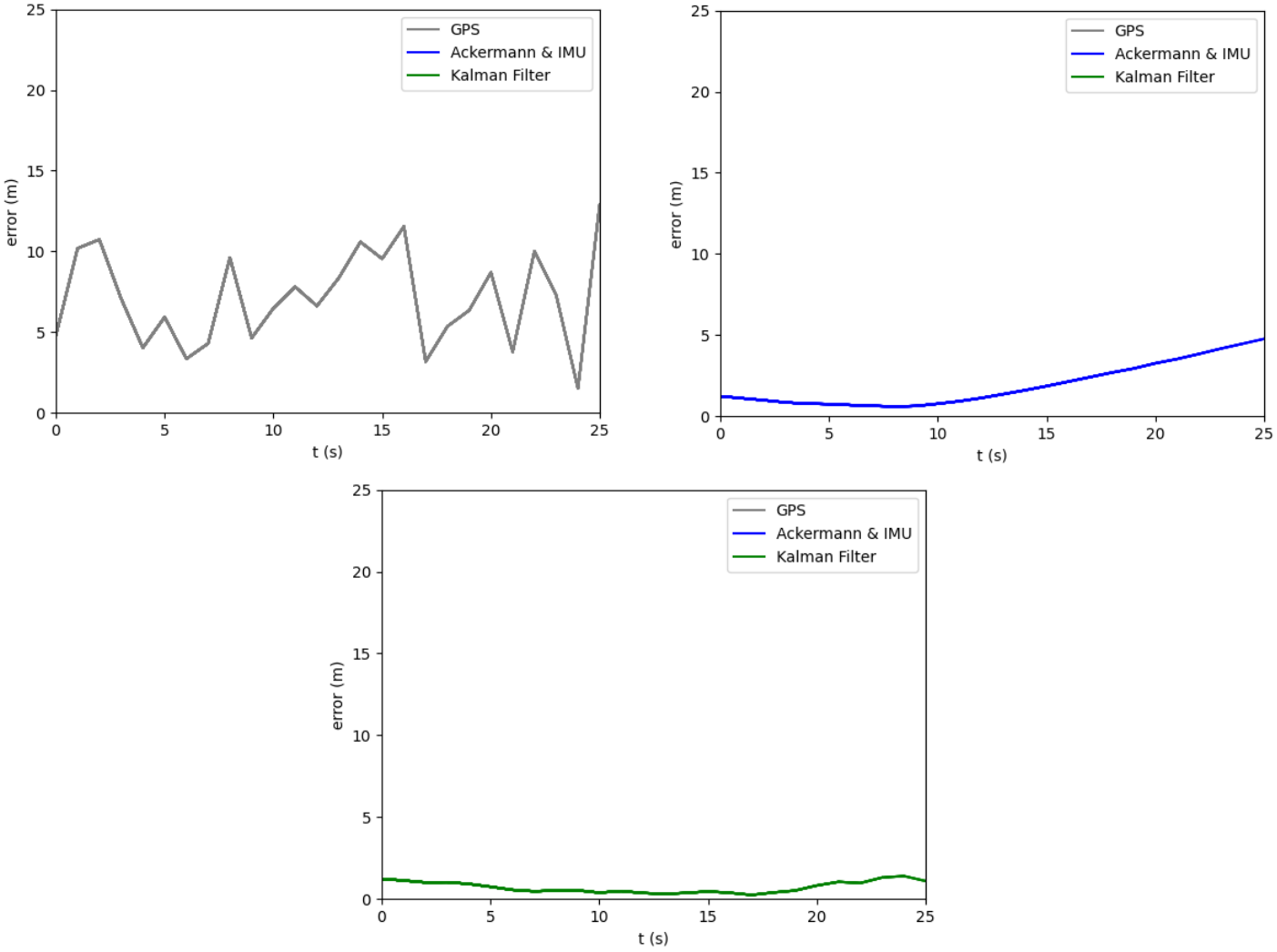


Abbildung 09: Vergleich der Methoden zur Lokalisierung

Abbildung 09 vergleicht drei verschiedene Methoden, die Position des Roboters zu bestimmen.
Links zeigt den ersten Ansatz mittels GPS. Die gemessenen Koordinaten springen dabei wild umher, aber befinden sich dabei immer ungefähr bei der Position des Roboters.
Rechts wird die Position nur anhand der Odometrie und des Lenkeinschlages bestimmt. Der Gyro-Sensor wurde hierbei weggelassen, um das Problem besser zu veranschaulichen: Auch mit einem Gyro würde die Lokalisierung langfristig driften.
Unten ist das Zusammenwirken der beiden Sensoren dargestellt. Hier werden die Eigenschaften der beiden Sensoren miteinander kombiniert und man erhält somit ein wesentlich besseres Gesamtergebnis. Als Hauptsensor wird der Odometrie-Sensor zusammen mit dem Lenkeinschlag genutzt. Der GPS-Sensor dient anschließend zur Korrektur der Position, um dem Driften vorzubeugen. Durch das gleichzeitige Berücksichtigen der Ungenauigkeit des GPS-Sensors, werden dabei auch die Sprünge dieses Sensors reduziert.

4.2 Versuch Rausch- und Driftverhalten

Sensoren sind nie perfekt und beinhalten sowohl systematische als auch zufällige Messfehler.
Der letzte Versuch (vgl. *Kapitel 4.1 Versuch Lokalisierung*) hat bereits gezeigt, dass der Kalman Filter durch die Fusion verschiedener Sensoren zu einem wesentlich besseren Ergebnis führt, als die einzelnen Sensoren alleine.
Im Folgenden wird der Einfluss der Parameter, die das Rauschverhalten beschreiben, untersucht.

Der Parameter *Q* steht für das Rauschen, das bei der Vorhersage des Kalman-Filters entsteht, da nie alle Einflüsse abgebildet werden können und durch das Driften des Gyroskop Sensors die Genauigkeit eingeschränkt wird.
Gleichzeitig entsteht auch eine Ungenauigkeit in der Messung des GPS Sensors, welche im Kalman-Filter mit dem Parameter *R* abgebildet wird.
Sind *Q* oder *R* zu klein, kann ein Teil des Rauschens nicht herausgefiltert werden. Auf der anderen Seite werden durch ein zu großes *Q* oder *R* Veränderungen des Systems erst mit einer Zeitverzögerung modelliert. Umso wichtiger ist es somit, eine gute Wahl von *Q* und *R* zu treffen.
Q beschreibt den Wert, den die einzelnen Elemente einer Matrix auf ihrer Diagonale annehmen.
Mit dem Ausdruck $Q_{Rausch} = 0.1$ bezeichne ich im Folgenden die Identitätsmatrix $Q = I_3 * Q_{Rausch}$
Die folgenden Parameter wurden experimentell bestimmt und haben sich als interessant herausgestellt:

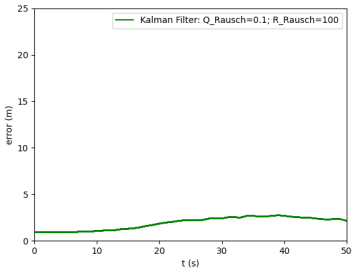


Abb. 10: $Q_{Rausch} = 0.1;\ R_{Rausch} = 100$

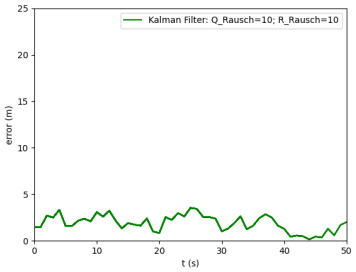


Abb. 11: $Q_{Rausch} = 10; R_{Rausch} = 10$

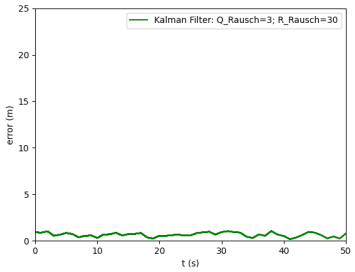


Abb. 12: $Q_{Rausch} = 3; R_{Rausch} = 30$

In Abbildung 10 ist das Prozessrauschen $Q_{Rausch} = 0.1$ und damit relativ klein. Dies bedeutet, dass der Vorhersage und damit dem Gyroskop und der Odometrie stark vertraut wird. Dagegen wird der Messung mit einem sehr hohen Messrauschen $R_{Rausch} = 100$ wesentlich weniger vertraut. Dies hat zur Folge, dass über die Zeit die Schätzung des Zustands leicht driftet, da der Vorhersage zu stark vertraut wird.
In *Abbildung 11* wird das Prozessrauschen $Q_{Rausch} = 10$ daher wesentlich erhöht und das Messrauschen $R_{Rausch} = 10$ wesentlich reduziert. Doch dadurch wird ein Teil des Messrauschens nicht komplett gefiltert und es kommt zu Sprüngen. Dennoch ist mit diesem Parametersatz das Rauschen im Vergleich zu den rohen GPS Daten (vgl. *Kapitel 4.1 Versuch Lokalisierung*) bereits um die Hälfte reduziert worden.
Abbildung 12 zeigt einen Verlauf, der deutlich mehr dem Ziel, die Vorteile von Gyroskop, Odometrie und GPS zu vereinen, näher kommt. Der Kalman Filter reagiert hier schnell auf Veränderungen und kann dennoch den Drift des Gyros und der Odometrie kompensieren.
Im ersten Versuch wurden Parameter gewählt, die zu stark der Vorhersage folgen und daher den Drift dieser Sensoren nicht unterdrücken. Im zweiten Versuch wurde zu stark dem GPS vertraut, wodurch die Unsicherheit des GPS Signals zu stark auftritt. Mit dem dritten Datensatz konnte ein zufriedenstellendes Ergebnis geliefert werden. Es ist dabei aber nicht möglich, allgemeingültige Parameter für ein optimales Ergebnis zu definieren. Die Parameter hängen sehr stark von der konkreten Anwendung und deren Sensoren ab.

5. Zusammenfassung

In dieser Arbeit wurde gezeigt, wie die Position und Orientierung des Roboters in Echtzeit abgeschätzt werden kann. Dazu wurden die Grundlagen für einen Kalman-Filter vorgestellt.
Der Kalman-Filter arbeitet mit Daten aus dem GPS-Sensor, dem Gyroskop sowie der Odometrie, mit dem Ziel, die Vorteile aller Sensoren zu vereinen. Der Vorteil des Gyroskops und der Odometrie ist das geringe Rauschen, wobei diese langfristig driften. GPS dagegen weist keinen Drift auf, ist jedoch sehr ungenau. Durch eine optimale Wahl der Parameter für das Rauschverhalten kann durch den Kalman Filter eine sehr genaue Position geschätzt werden.
Doch die bisherige Idee, die Kamera zusätzlich zur Lokalisierung zu nutzen (vgl. *Kapitel 2.3.6*) ist zwar gut, jedoch ist die aktuelle Methode noch nicht in allen Situationen praktikabel. Beispielsweise wird noch vorausgesetzt, dass die aktuelle Position bereits ungefähr korrekt ist und nur noch minimal korrigiert werden muss. Außerdem müssen die Kamera Bilder zum Vergleichen erst gespeichert werden, bzw. jede Route dem Roboter erst beigebracht werden.
Eine mögliche Alternative, welche in der Zukunft getestet werden soll, ist die Lokalisierung aus der Vogelperspektive [5]. Dies hätte den Vorteil, dass eine einzige Karte aus der Vogelperspektive genutzt werden kann. Bisherige Karten von Google Maps sind vermutlich zu ungenau für diesen Zweck und es müsste eine eigene Karte über eine Drohne erstellt werden. Dadurch lässt sich dann die Kamera zur Lokalisierung auch nutzen, wenn noch keine Position des Roboters bekannt ist. Wenn der Winkel der Kamera am Roboter bekannt ist, kann dieses Livebild zu einer virtuellen Ansicht von oben transformiert werden [6].

6. Quellen

[1] Shuttle Modellregion Oberfranken: Autonomes Fahren
https://www.shuttle-modellregion-oberfranken.de/, Stand 01.03.2023

[2] Coppelia Robotics: Simulation https://www.coppeliarobotics.com/, Stand 01.03.2023

[3] ilectureonline.com: Kalman Filter
http://www.ilectureonline.com/lectures/subject/SPECIAL%20TOPICS/26/190, Stand 01.03.2023

[4] OpenCV Team: OpenCV-Home https://opencv.org/, Stand 01.01.2023

[5] Toriya, Hisatoshi, Itaru Kitahara, and Yuichi Ohta. "Mobile camera localization using aerial-view images." Information and Media Technologies 9.4 (2014)

[6] MathsWork: Birds-Eye-View https://de.mathworks.com/help/visionhdl/ref/birdseyeview.html, Stand 01.01.2023

[7] NPS Wiki: Lokales Koordinatensystem
https://wiki.nps.edu/display/RC/Local+Coordinate+Frames, Stand 01.03.2023

[8] Manuel Selch, Marco Selch, Leon Wolf:
Autonomer Rucksack: Der Lieferroboter als intelligente Unterstützung, Jugend Forscht 2021/22

7. Danksagung

Dank gilt meinem Lehrer, Herrn Dr. Andreas Stingl, der mich während des Robotikunterrichtes zu der Idee dieses JuFo Projekt inspiriert und uns immer wieder ermutigt hat, dieses Projekt voranzutreiben und fertigzustellen. Außerdem danke ich meiner Familie für die Bereitstellung der Technik und die Unterstützung bei der Entwicklung und Fertigstellung des Projektes zu Hause. Gleichzeitig bedanke ich mich auch bei meinen Freunden und Klassenkameraden, die mir ebenfalls Werkzeuge für die Konstruktion der Prototypen zur Verfügung gestellt haben.