Inhaltsverzeichnis

Abbildungsverzeichnis

Tabellenverzeichnis

1 Einleitung

Im Gegensatz zu den vergangenen Jahren war es die Aufgabe dieses Projektseminars Echtzeitsysteme eine Steuerung zu entwickeln, welche einen vorgegebenen Rundkurs durch eine Kamera erkennt und diesen absolvieren kann. In den letzten Jahren wurden Sensoren und die Abstände zu den Wänden dafür genutzt. Durch eine Kamera und verschiedene Sensoren wird die Umgebung analysiert und durch Filterung und Regelung Werte ermittelt, um das Fahrzeug zu steuern. Die finale Implementation lässt das Fahrzeug autonom den Rundkurs bewältigen, ohne dabei über die markierten Linien zu fahren. Als zweite Aufgabe wurde Hinderniserkennung mit Spurwechsel gewählt, hier erkennt das Fahrzeug ein Hindernis und wechselt die Spur, um diesem auszuweichen. Sollte die Strecke komplett blockiert sein oder das Fahrzeug sich auf eine reale Wand zu bewegen, wird dies erkannt und das Fahrzeug hält an.
In den folgenden Kapiteln werden die Grundlage der Hardware und Software, die Organisation des Teams, die Implementierung der Aufgaben und die Probleme beschrieben. Es wird auf die einzelnen Bestandteile der Regelung und der Bildverarbeitung eingegangen, sowie der Entwicklungsprozess beschrieben.

2 Grundlagen

In diesem Kapitel werden die verschiedenen Grundlagen auf der Hardware- und Softwareseite beschrieben. Dies schließt Informationen über das Modellauto, die verwendeten Kameras, das Metabetriebssystem ROS, OpenCV, die vorgegebenen Packages von PSES und die Gestaltung des Rundkursen in den Veranstaltungsräumen mit ein.

2.1 Hardware

Die benötigte Hardware für das Projektseminar wurde allen Gruppen zur Verfügung gestellt und es gab eine kurze Einführung in die Arbeitsräume und den Umgang mit den einzelnen Teilen. Zur Verfügung standen das Modellfahrzeug, eine Kinect Kamera, eine Weitwinkelkamera, passende Akkus und Ladestationen und einiges an Klebeband. In den folgenden Unterabschnitten werden das Modellfahrzeug und die Kameras genauer beschrieben.

2.1.1 Modellauto

Das zur Verfügung gesellte Fahrzeug ist einem Auto ähnlich. Es wurde speziell für das Projektseminar entwickelt und durch die TU Darmstadt gepflegt. Es besitzt vier Räder einen Motor und ist mit verschiedenen Sensoren, einer Kamera einem Mini-Computer und Mikrocontrollern ausgestattet. Da sich die Aufgabenstellung dieses Seminars auf Bildverarbeitung fokussierte, wird hier nicht näher auf die anderen Sensoren eingegangen. Die Steuerung erfolgt über den Computer, auf welchem ROS läuft (siehe Unterunterabschnitt 2.2.1). Das Fahrzeug besitzt einen maximalen Lenkwinkel und eine maximale Geschwindigkeit, diese wurden durch eine bereitgestellte Fernsteuerung gemessen. Das Mainboard ist mit diversen Anschlüssen ausgestattet, die es erlauben eine Tastatur, Maus und einen Bildschirm anzuschließen. Des Weiteren ist eine Kinect und eine Weitwinkelkamera vorhanden (siehe Unterunterabschnitt 2.1.2). Die Stromversorgung während der Fahrt übernehmen zwei Akkupacks.

Bild Auto

2.1.2 Kameras

Zu Beginn war eine Kinect Kamera auf dem Fahrzeug verbaut. Diese wurde in den frühen Phasen der Entwicklung genutzt, jedoch kurz nach der Zwischenpräsentation durch die Weitwinkelkamera ersetzt. Diese zeichnet sich durch ein 120° Sichtfeld, eine FullHD Auflösung und einem leicht einstellbaren Positionswinkel aus. Grund für den Austausch war eine schlechte Farbwahrnehmung der Kinect, diese wurden übersaturiert dargestellt (näheres siehe Unterabschnitt 5.2). Eine Erkennung der richtigen Farbe durch die bereitgestellten Bibliotheken war dadurch nicht mehr möglich. Mit der Weitwinkelkamera wurde dieses Problem gelöst. Die Farben wirken natürlicher und sind durch die angewendeten Filtermethoden extrahierbar. Da die Halterung der Webcam auf Computerbildschirme ausgerichtet ist, musste eine elegante Lösung gefunden werden. Ein festkleben mit Klebeband hielt für den Anfang, war jedoch zu anfällig für ein Verstellen der Kamera. Deswegen wurde ein 3D-Modell entwickelt und eine passende Halterung

Abbildung 2.1: Halterung der Weitwinkelkamera

im Fablab[1] gedruckt. Mit dieser konnte die Kamera stabil am Fahrzeug befestigt werden und es gab keine Schwankungen mehr durch andere Kamerapositionen (nähere Informationen in Unterabschnitt 5.3).

Bild updaten

2.2 Software

In diesem Abschnitt wird die Software vorgestellt, die auf dem Mini-Computer läuft und während der Programmierung genutzt wird. Das Hauptbetriebssystem stellt Lubuntu[2] dar. Auf diesem ist das Metabetriebssystem ROS installiert, welches die eigentlichen Funktionen übernimmt. Die Programmierung der Nodes geschieht mit dem Programm RoboWare Studio, dort sind alle vorgefertigten Nodes und die selbst erstellten übersichtlich dargestellt. Programmiert wird in der Programmiersprache C++.
Im folgenden werden häufig genutzte Pakete beschrieben und das Meta-Betriebssystem ROS.

2.2.1 ROS - Robot Operating System

ROS ist ein Metabetriebssystem für Roboter, welches auf Linux basiert und in vielen Unternehmen zur Steuerung von Robotern genutzt wird. Es stellt mehrere Pakete zur Verfügung, die einige nützliche Funktionen ermöglichen. Dazu gehört die Verteilung auf mehrere Systeme im Netzwerk, Paketverarbeitung und die Kommunikation zwischen den Nodes [**?**]. Die Kommunikation findet durch ein Publish-Subscribe-System statt. Die einzelnen Nodes publishen zu bestimmten Themen Nachrichten, deren Inhalt

[1] https://www.fablab.tu-darmstadt.de/
[2] https://lubuntu.net/

sich auf das Thema bezieht. So erhält man zum Thema `/uc_bridge/usr` über eine Subscription Nachrichten über die Werte des rechten Abstandssensors. Auch die Steuerung des Motors und des Lenkwinkels geschieht über das puplishen von Nachrichten. Die Eigenschaften der Pakete und Tutorials zu den Funktionen sind auf der Website von ROS zu finden und erleichtern den Einstieg. Durch Open Source lässt sich schnell herausfinden, was wirklich benötigt wird und welche Funktionen im späteren Verlauf hilfreich sein könnten.

2.2.2 OpenCV

Zur Verarbeitung der Kamerabilder wird die freie Bibliothek OpenCV[3] genutzt, diese ist auch durch eine Node in ROS eingebunden und einfach nutzbar. Zur Verwendung wird das von der Kamera erhaltene Objekt in der Datenstruktur von OpenCV als `cv::Mat` gespeichert. Darauf lassen sich alle späteren Bearbeitungsschritte ausführen. Sie stellt viele nützliche Funktionen bereit, die das Erkennen der Fahrspuren und der Hindernisse erleichtern. Dazu gehören zum Beispiel `cv::wrapperperspective`, mit der das Bild in eine Bird-Eye View umgewandelt wird und `cv::imshow` zum Anzeigen des aktuellen Bildes. Interessanterweise funktionierte diese Funktion nur, wenn sie im Zusammenhang mit `cv:waitKey` aufgerufen wurde. Mittels `cv::cvtColor`, `cv::inRange` und `cv::GaussianBlur` lässt sich der Farbraum des Originalbildes transformieren und in diesem anschließend alle Farben herausfiltern, die in einem bestimmten Bereich liegen. In der späteren Verarbeitung war es nötig Farbwerte von einzelnen Pixeln auszuwerten, dies geschah mit Hilfe der Methode `cv::mat.at<uchar>(y,x)`. Auch ein Einzeichnen von neuen Punkten war möglich und für die spätere Kurvenerkennung hilfreich.

2.2.3 PSES Packages

Zur Kommunikation mit den einzelnen Fahrzeugteilen sind von den Veranstaltern einige Nodes vorgegeben, dazu gehören `pses_uc_bridge`, `pses_simulation`, `pses_dashboard` und `pses_odometry`. Von diesen wurde in der finalen Lösung nur `pses_uc_bridge` genutzt. Diese Node muss zuerst gestartet werden, da sie alle Steuerfunktionen übernimmt und die Publish- und Subscribe-Komponenten für die verschiedenen Werte erstellt. Über `pses_dashboard` kann das Fahrzeug manuell gesteuert werden, dies wurde genutzt, um herauszufinden wie groß der maximale Lenkwinkel und die Geschwindigkeit waren, die das Fahrzeug bewältigen kann. Da diese Node gestartet über eine graphische Oberfläche verfügt, war auch ein Ablesen der jeweiligen Sensorwerte möglich. `pses_simulation` stellt eine Simulationsumgebung für das Fahrzeug da, in der der erstellte Code getestet werden konnte. Dies wurde jedoch nicht genutzt, da ein Testen in der realen Umgebung bessere Werte lieferte und in der Simulation kein Kamerabild simuliert werden konnte. Die Node `pses_odometry` dient zur Selbstlokalisierung des Fahrzeuges. Nach dem Start ist es möglich die zurückgelegte Strecke und die einzelnen Positionsdatenveränderungen zu berechnen, um sich ein genaueres Bild über die Position des Fahrzeuges in seiner Umwelt zu machen.

[3] https://opencv.org/

2.3 Rundkurs

Zur Verfügung standen zwei Räume in Gebäude S3|11, dort wurden die Fahrzeuge, die Akkus und alles weiter benötigte Material aufbewahrt. Zusätzlich boten sie Platz zum Programmieren und Basteln und in einem Raum war der Boden mit einem Rundkurs ausgestattet. Dieser war mit farbigem Klebeband auf eine Teichfolie in mitten des Raumes geklebt. Er bot zwei grüne Linien am Rand und eine pinke Mittellinie. Hier fanden sich öfter mehr als eine Gruppe ein, um den zuvor programmierten Code zu testen, die Kameras einzustellen oder die Lenkwinkel zu messen.
Da dieser Rundkurs für die finale Bewertung und den Wettbewerb am Ende genutzt wird, konnte man seine Regelung und Steuerung anpassen und die Lichtverhältnisse konstant halten. Es wurde schnell klar, dass geringe Änderungen des Lichtes eine große Änderung bei der Farbwahrnehmung der Kamera nach sich zogen.

3 Organisation

In diesem Abschnitt wird das Projektmanagement des Teams beschrieben, das zum erfolgreichen Abschluss des Projektes beigetragen hat. Bei einem Team mit fünf Personen ist es wichtig Maßnahmen zur Dateiverwaltung und Organisation zu treffen, um eine gute Zusammenarbeit zu gewährleisten.

3.1 Gruppentreffen und Organisation

Im Folgenden werden die Gruppentreffen und das Team an sich beschrieben.

Gruppentreffen

Bei dem ersten Teamtreffen wurde entschieden welche Versionsverwaltung und Organisationsplattform verwendet wird, diese wurden zusammen eingerichtet.

Im zwei Wochen Takt traf sich vorzugsweise das gesamte Team mit einem Betreuer des Projektes. Mit ihm wurde der aktuelle Stand des Projektes besprochen und gegebenenfalls Probleme im Team dargelegt. Diese Treffen waren organisatorischen Zwecken gewidmet, es wurden keine Fragen direkt zur Implementierung geklärt. Als das Projekt grundlegend funktionierte, wurden Anregungen zur Erweiterung der Aufgabenstellung gegeben.

Zur Besprechung von Regelungs- und Implementierungsfragen, gab es annähernd jede zweite Woche ein Regelungstechniktreffen, bei dem sich zwei Teammitglieder jeder Gruppe einfanden. Die Teilnehmer tauschten sich untereinander über den Stand der jeweiligen Gruppen aus und diskutierten verschiedene Fragen. Für weitere Fragen und Anregungen stand dort ein Betreuer der Regelungstechnik zur Verfügung.

Zusätzlich zu den Beratungsgesprächen wurden wöchentliche Treffen mit allen Gruppenmitgliedern festgelegt. Jeder berichtete von seinem Vorgehen der vergangenen Woche, so dass alle im Team den Überblick über den aktuellen Stand des Projektes hatten. Daraufhin wurden Ideen und Anregungen zusammengetragen und die Planung für die nächste Woche erarbeitet. Mit Hilfe der im Unterabschnitt 3.2 noch erwähnten Organisationsplattform Trello wurden die Aufgaben erfasst und den interessierten Gruppenmitgliedern zugeordnet.

Team

Die Zeitplanung wurde dynamisch gehalten, da es schwer einschätzbar ist, wie viel Zeit eine einzelne Aufgabe in Anspruch nehmen wird. Erfahrungen aus anderen Projekten haben gezeigt, dass dieses Vorgehen sinnvoll ist. Um genügend Zeit für die Optimierung und Dokumentation einzuplanen und gegebenenfalls einen ausreichenden Zeitpuffer zur Verfügung zu haben, wurde als Ziel festgelegt, dass nach vier von fünf Monaten der Projektzeit eine funktionierende Version vorhanden sein soll. Jede Woche wurde bei dem Gruppentreffen festgelegt, wer sich welcher neuen Aufgabe annimmt und es wurde erneut abgeschätzt, wie viel Zeit eine angefangene Aufgabe noch beanspruchen wird.

Nachdem sich alle in die Dokumentation eingelesen hatten, wurden erste Grundlagen zusammen erarbeitet, damit jeder ein Grundwissen über das Modellauto hatte.

Es bildeten sich Aufgabengruppen für Organisation, Regelung und Bildverarbeitung, diese wurden weitestgehend parallel behandelt. Die Verantwortung für diese Teilgruppen wurde auf die Mitwirkenden aufgeteilt. Jedes Gruppenmitglied konnte bei jeder Aufgabengruppe helfen, allerdings ist es wichtig Verantwortliche festzulegen, um den Überblick zu bewahren und Zielführend zu agieren. Als die Tests der Bildverarbeitung und Regelung die aufgestellten Bedingungen erfüllten, wurden diese Aufgabenbereiche zusammengeschlossen und es wurde dadurch das Fahren durch den Rundkurs erlangt. Neben der Optimierung des Rundkurses wandte sich eine Subgruppe dem zweiten Thema, Hinderniserkennung mit Spurwechsel, zu.

Es gab viele Aufgaben, die direkt am Fahrzeug getestet werden mussten. Da das gleichzeitige Testen mehrerer Aufgaben meist nicht möglich war, wurden die anderen Teammitgliedern beim Zusammenkommen über die Aufgabenverteilung hinaus unterstützt. Jeder teilte sich seine Zeit jede Woche selbst ein und kommunizierte, welche Aufgabe wann bearbeitet wird, damit Interessengruppen zusammenfinden konnten.

3.2 Aufgabenverwaltung

Für die Planung der Aufgaben wurde die web-basierte Projektmanagementsoftware Trello[4] verwendet. Durch Visualisierung im Karteikastensystem wurde der Überblick über die bevorstehenden Aufgaben und die dazugehörenden Verantwortlichen bewahrt. Zusätzlich gab es Farblabel, um die zusammengehörenden Aufgaben visuell zu markieren. Das Board wurde in Kategorien, wie *ToDo, Im Gange, Testen, Fertig und Zurückgestellt/Gescheitert*, aufgeteilt. Die einzelnen Aufgaben wurden je nach Aufgabenstatus in die zutreffende Kategorie verschoben.

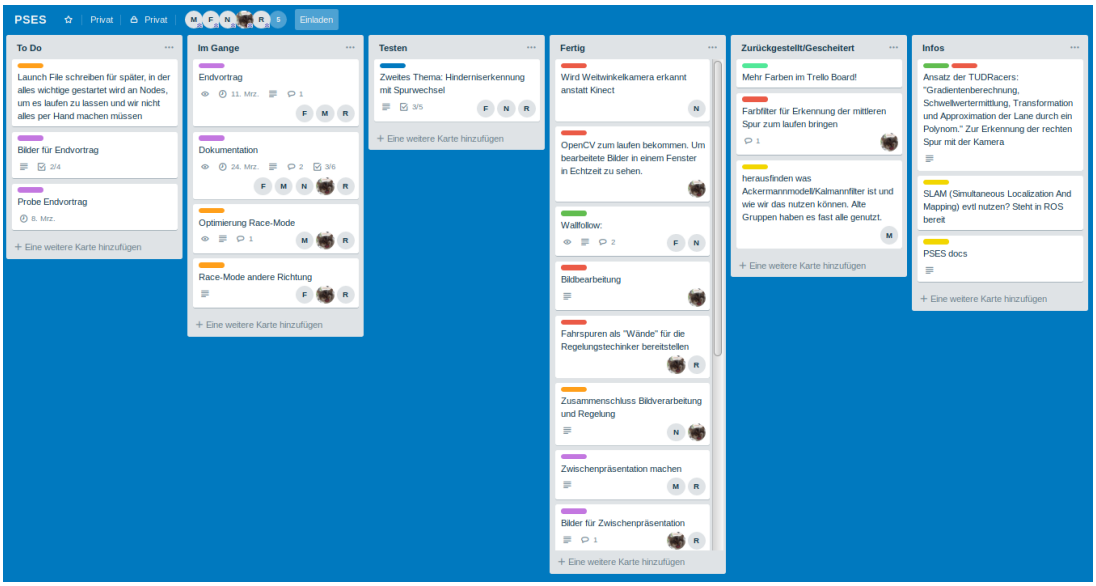


Abbildung 3.1: Trello-Board von Team AUDO zum Projektstand: Optimierung der einzelnen Fahrmodi

[4] https://trello.com/

3.3 Versionsverwaltung

Angesichts guter Erfahrungen stand schnell fest, dass GitHub[5] als Versionsverwaltung verwenden wird. Die meisten Teammitglieder verfügten bereits über Kenntnisse mit GitHub und mussten sich nicht in neue Verwaltungsstrukturen einarbeiten. Der Vorteil von Versionsverwaltungen ist, dass alte Programmierzustände gesichert sind und man diese wieder herstellen kann. Durch die Versionierung des Quellcodes mit GitHub wurde das fehlerfreie Programmieren im Team sichergestellt. Ohne Verwaltung des Quellcodes ist es fehleranfällig mit mehreren Leuten an dem gleichen Programmiercode zu arbeiten. Über GitHub wurden zusätzlich zu dem Quellcode die Vortragsfolien und die Dokumentation verwaltet.

[5] https://github.com/

4 Implementierung

Im folgenden Kapitel wird die finale Implementierung und die Schritte dahin beschrieben. Es wird auf die Bildverarbeitung, die Erkennung von Fahrspuren und Hindernissen und die darin enthaltenen Regelungsansätze eingegangen. Zur Übersicht zeigt Abbildung 4 die Nodes und verwendeten Topics eines Fahrmodi. Es wird dargestellt wie die Bilder von `/cv_camera` an `/line_recognition_node` weitergegeben und dort verarbeitet werden. Die so entstandenen inhaltlich wichtigen Punkte werden an die `/drive_node` gesendet, welche die Steuerung und Kommunikation mit der `/uc_bridge` übernimmt.

schauen wegen Nummerierung der Abbildungen am Ende, da es irgendwie zu Problemem mit dem ersten kommt

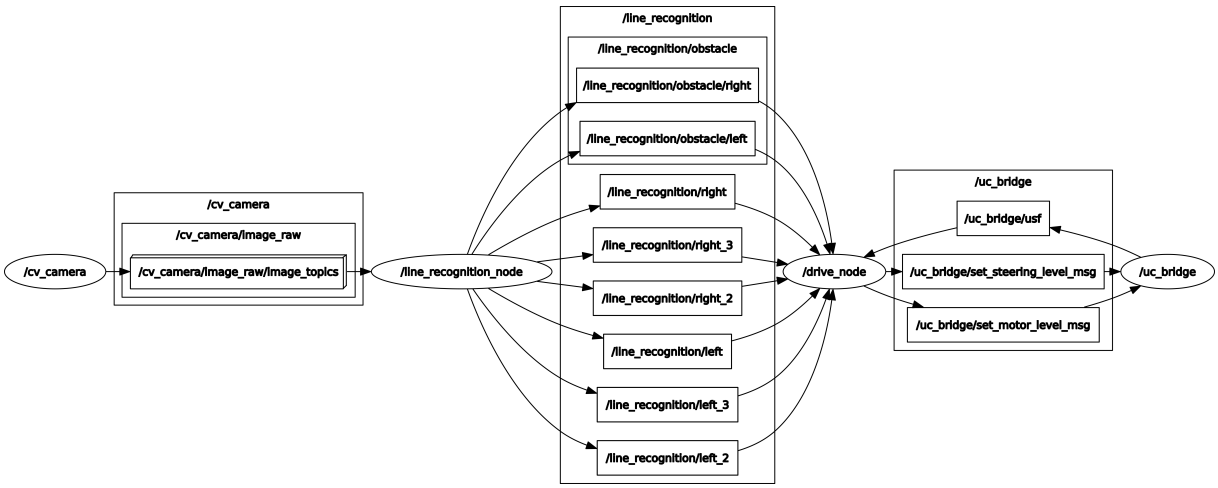


Abbildung 4.1: Nodes und Topics der Implementierung

4.1 Bildverarbeitung und Erkennung von Fahrspuren

Zur Bilderzeugung konnte zwischen zwei Kamerasystemen gewählt werden: Einer Kinect Kamera sowie einer Weitwinkel Webcam. Die Entscheidung fiel auf die Weitwinkel Webcam, da sich diese aufgrund ihrer Weitwinkel- und Farbdarstellungseigenschaft und den in Abschnitt 5 genannten Problemen der Kinect Kamera sich als besser geeignet herausstellte.

Die mittels OpenCV eingelesenen Kamerabilder werden durch die Natur der Kameraposition in Zentralperspektive aufgezeichnet. Diese Art der Projektion eignet sich aufgrund des vorhandenen Fluchtpunktes nicht für die benötigte Bildverarbeitung, weshalb eine Transformation in Vogelperspektive durchgeführt wird, welche die Problematik des Fluchtpunktes beseitigt.

Zur Erkennung der Fahrspuren wurden die Farbwerte der grünen äußeren Fahrbahnmarkierungen ermittelt. Mithilfe dieser Farbwerte kann ein Graustufenbild erzeugt werden, welches die Übereinstimmung mit dem vorgegebenen Farbwert repräsentiert. Je mehr der Farbpixel des Originalbildes mit dem verglichenen Farbwert übereinstimmt, desto heller erscheint der Pixel an seiner Stelle im Graustufenbild. Um die Orientierung

in diesem sicherzustellen, werden drei Marker in äquidistanter Höhe auf die rechte und linke Linie gesetzt (siehe Abbildung 4.1). Der Algorithmus zur Positionierung der Marker sucht dabei zeilenweise im Bild nach aufeinanderfolgende helle Pixel und addiert die Helligkeit dieser Pixel auf. Übersteigt diese Summe einen Schwellwert, so wird diese Stelle mit einem Marker versehen.

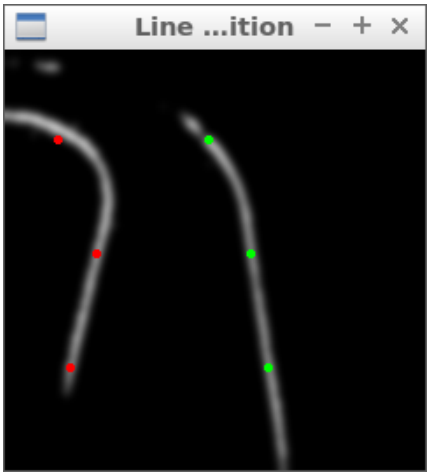


Abbildung 4.2: Fahrspur mit Markern

4.2 Erkennung von Kurven und Geraden

Um das Fahrverhalten besser anpassen zu können, fiel die Entscheidung darauf Fahrmodi zu erstellen, welche unterschiedliche Reglerparameter besitzen. Um diese Fahrzustände korrekt erkennen zu können ist es von Nöten zwischen Geraden, Kurven, Kurveneinund Kurvenausfahrten zu unterscheiden.

Um eine Kurve zu erkennen, werden die drei gesetzten Marker jener Seite betrachtet, an welcher sich das Fahrzeug gerade orientiert. Da sich diese in gleichem vertikalen Abstand zu einander befinden wird lediglich der horizontale Abstand betrachtet. Zuerst wird hierfür der vertikale Abstand zwischen dem Unteren sowie dem Mittleren Marker ermittelt. Dieses Verfahren wird mit dem Oberen und dem Mittleren wiederholt. Übersteigt die Differenz der beiden ermittelten Abstandswerte einen Schwellwert, so liegt eine Kurve vor. Abbildung 4.2 zeigt eine solche Situation.

Um nun auch Kurvenein- und Kurvenausfahrten erkennen zu können, wurde ein Puffer implementiert, welcher aus vorherigem und aktuellem Kurvenerkennungswert ermitteln kann in welcher Umgebungssituation sich das Fahrzeug befindet. Des Weiteren dient der Puffer als eine Hysterese, welche ein Flackern der Fahrzustände vermeidet.

4.3 Erkennung von Hindernissen

Wie auch die Fahrspurerkennung, basiert die Hinderniserkennung auf der Filterung der Kamerabilder auf eine spezielle Farbe. Aus diesem Grund wurden die Hinderniswürfel, um sie vom Rest der Kulisse unterscheiden zu können mit orangefarbenen Klebeband versehen. Um sicher zu gehen, dass nicht fälschlicherweise ein Hindernis erkannt wird,

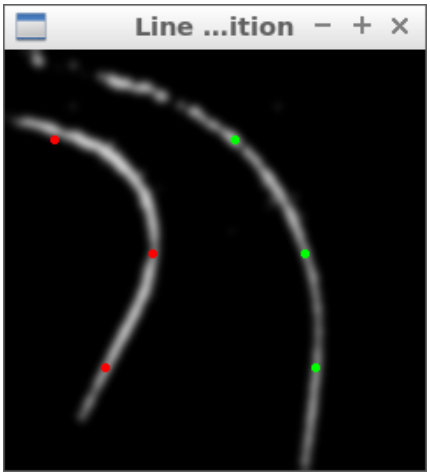


Abbildung 4.3: Kurveneinfahrt mit Makern

wird ein ähnlicher Algorithmus wie bei der Fahrspurerkennung angewandt. Des Weiteren wird ein Hindernis nur als solches erkannt, wenn zwei vertikale Linien nebeneinander erkannt werden, wie es bei den Hinderniswürfeln der Fall ist. Prinzipiell wäre es somit auch möglich andere Gegenstände in die Fahrspur zu stellen, solange diese zwei vertikale orangefarbene Streifen besitzen. Dies wurde mit einem auf einer Spur fahrenden Fahrzeug einer anderen Gruppe erfolgreich getestet. Abbildung 4.4 zeigt die verschiedenen Phasen der Hinderniserkennung.

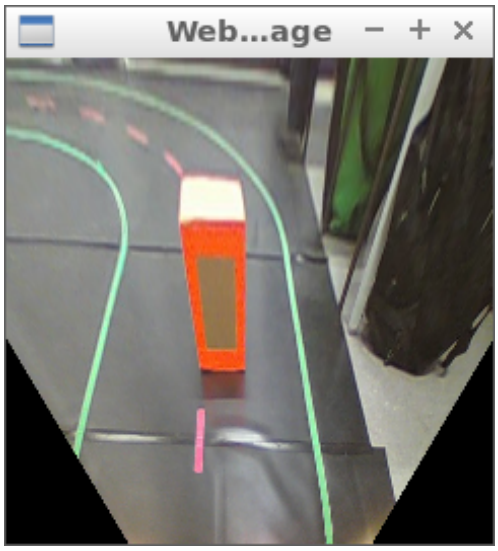
4.3.1 Kollisionsvermeidung

Um in allen Fahrsituationen eine Beschädigung des Fahrzeugs zu vermeiden, wird eine Kollisionsvermeidung eingesetzt. Diese verhindert unabhängig von Fahrspuren oder erkannten Hindernissen, dass es im Fahrbetrieb zu einem Zusammenstoß mit anderen Gegenständen oder der Wand kommt. Hierzu werden die Daten des vorderen Ultraschallsensors ausgewertet, der einen ungefähren Abstandswert zum nächstgelegenen Gegenstand liefert.

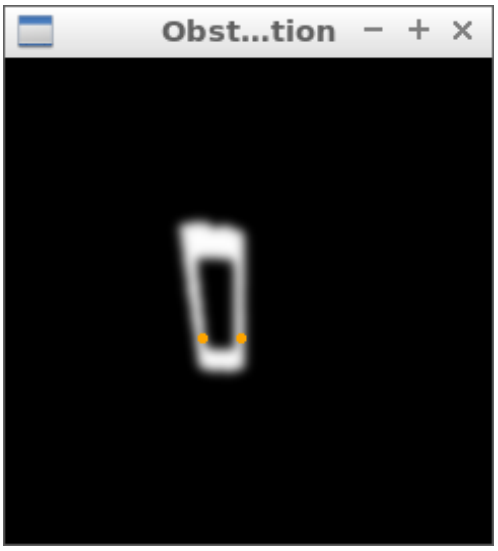
Um aus dem fehlerbehafteten Signal verwertbare Daten zu erhalten werden die Werte des Sensors geglättet. Dies geschieht indem die letzten 20 Sensorwerte gemittelt werden. Liegt der Durchschnitt unter einem Schwellwert, wird der Motor angehalten. Als praxistauglicher Wert hat sich 0.35 erwiesen, also ein Abstand von 35 Zentimetern.

Ein besonderes Problem ergibt sich aus der Tatsache, dass die Werte des Sensors in unregelmäßigen Abständen kurzzeitig auf den Wert Null springen. Dies entspräche einem Gegenstand direkt vor dem Sensor, was schon alleine aufgrund der Fahrzeuggeometrie nicht plausibel ist. Diese Werte müssen daher von der Auswertung ausgenommen werden.

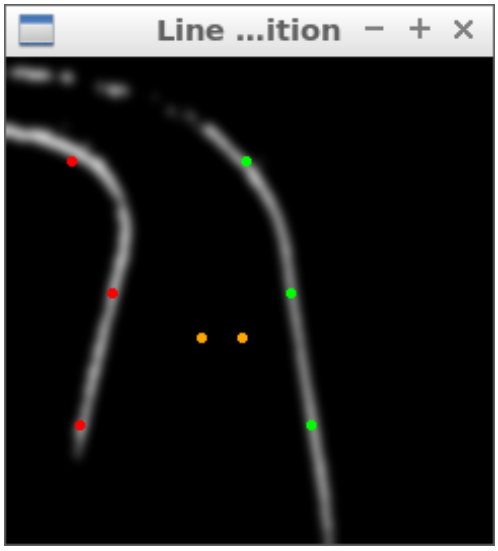
```
1   collision_protection, no collision with objects
2   double range from usf value
3   return true, if there is an object
4 */
5 bool collision_protection(double range)
```



(a) Transformiertes Farbbild



(b) Hindernis nach Filterung



(c) Markiertes Hindernis

Abbildung 4.4: Fahrspur mit Hindernis

```
{
  // initialization
  if(usf_flag)
  {
    for(int i = 0; i < RANGE_OF_USF_AVERAGE; i++)
    {
      usf_history[i] = 0;
    }
    usf_flag = false;
    return true;
  }
  // refresh array
  if(range > 0)
  {
    for(int i = 0; i < RANGE_OF_USF_AVERAGE - 1; i++)
    {
      usf_history[i + 1] = usf_history[i];
    }
      usf_history[0] = range;
  }
  // average
  double average = 0;
  for(int i = 0; i < RANGE_OF_USF_AVERAGE; i++)
  {
    average += usf_history[i];
  }
  average = average / RANGE_OF_USF_AVERAGE;
  if(average < 0.35)
  {
    return true;
  }
  else
```

Listing 4.1: Codeausschnitt zu Collision Detection

Bild des Ultraschallsensors

4.3.2 Regelungskonzept und Spurhaltung

Nach umfangreichen Tests zu Beginn des Projekts wurde ein strukturvariabler Regelungsansatz mit PD-Reglern gewählt. Eine strukturvariable Regelung lässt sich vor Allem dann sinnvoll einsetzen, wenn zwischen wenigen bekannten Arbeitspunkten umgeschaltet werden muss. Die Rennstrecke des Projektseminars Echtzeitsysteme lässt sich in Kurven- und Geradenabschnitte aufteilen. Aufgrund der hohen Nichtlinearität der Lenkungsmechanik (insbesondere Lose, Haftreibung, Gleitreibung) lässt sich mit einem

einzigen PD-Regler kein zufriedenstellendes Regelverhalten in allen Fahrsituationen erzielen. Eine stationäre Genauigkeit ist in diesem Fall zur Spurhaltung im Übrigen nicht erforderlich, solange die bleibende Regelabweichung in der Praxis so klein ist, dass alle Fahrsituationen der Rennstrecke ohne Linienüberschreitung gefahren werden können. Mit mehreren gut eingestellten PD-Reglern lässt sich diese Anforderung erfüllen.

Die Bildverarbeitung liefert die Positionsdaten aller Markierungslinien, die im Bildausschnitt der Kamera erfasst werden. Da die Position der Kamera auf dem Fahrzeug sich nicht ändert, kann ein fester Sollwert für den Abstand zu diesen Linien vorgegeben werden. Es muss zur Berechnung der Regelabweichung lediglich bekannt sein, ob das Fahrzeug sich an der rechten oder an der linken Fahrbahnbegrenzung orientieren soll. Näheres hierzu in Unterunterabschnitt 4.3.4.

4.3.3 Implementierung des PD-Reglers

Die Berechnung der Lenkwinkelregelung erfolgt in mehreren Schritten. Zunächst wird aus den situationsabhängigen Reglerparametern sowie Führungsgröße und aktueller Position eine Stellgröße berechnet. Der differentielle Anteil des PD-Reglers greift hierbei auf den aktuellen und vorherigen Wert der Regelabweichung, sowie die verstrichene Zeitdauer seit der letzten Berechnung zu.

```
  collision_protection, no collision with objects
  double range from usf value
  return true, if there is an object
*/
bool collision_protection(double range)
{
  // initialization
  if(usf_flag)
  {
    for(int i = 0; i < RANGE_OF_USF_AVERAGE; i++)
    {
      usf_history[i] = 0;
    }
    usf_flag = false;
    return true;
  }
  // refresh array
  if(range > 0)
  {
    for(int i = 0; i < RANGE_OF_USF_AVERAGE - 1; i++)
    {
      usf_history[i + 1] = usf_history[i];
    }
      usf_history[0] = range;
  }
  // average
```

```
27    double average = 0;
28    for(int i = 0; i < RANGE_OF_USF_AVERAGE; i++)
29    {
30      average += usf_history[i];
31    }
32    average = average / RANGE_OF_USF_AVERAGE;
33    if(average < 0.35)
34    {
35      return true;
36    }
37    else
```

Listing 4.2: Codeausschnitt zu Collision Detection

Codeausschnitt: PD Regler

Anschließend erfolgt eine Begrenzung der Stellgröße. Die Hardware erlaubt Lenkwinkel im Bereich -800....+800. Um Beschädigungen zu vermeiden wird der maximale Lenkwinkel softwareseitig auf -700....+700 begrenzt. Bei höheren Werten kommt es zu einem Blockieren des Lenkgetriebes.

Um ein ruhiges Lenkverhalten zu erreichen, werden die Ausgangswerte des Reglers zum Schluss geglättet. Dazu wird ein gewichteter Mittelwert der letzten Stellgrößen gebildet. Die Gewichtung erfolgt in Abhängigkeit des zeitlichen Verlaufs. Dabei hat der zuletzt berechnete Wert das größte Gewicht, ein 0,3 Sekunden zurückliegender Wert hingegen nur das halbe Gewicht. Der gewichtete Mittelwert erzeugt ein ruhiges Regelverhalten, ohne jedoch die Regelung zu stark zu verzögern.

```
1   collision_protection, no collision with objects
2   double range from usf value
3   return true, if there is an object
4 */
5 bool collision_protection(double range)
6 {
7   // initialization
8   if(usf_flag)
9   {
10    for(int i = 0; i < RANGE_OF_USF_AVERAGE; i++)
11    {
12      usf_history[i] = 0;
13    }
14    usf_flag = false;
15    return true;
16  }
17  // refresh array
18  if(range > 0)
19  {
```

```
20      for(int i = 0; i < RANGE_OF_USF_AVERAGE - 1; i++)
21      {
22        usf_history[i + 1] = usf_history[i];
23      }
24        usf_history[0] = range;
25    }
26    // average
27    double average = 0;
28    for(int i = 0; i < RANGE_OF_USF_AVERAGE; i++)
29    {
30      average += usf_history[i];
31    }
32    average = average / RANGE_OF_USF_AVERAGE;
33    if(average < 0.35)
34    {
35      return true;
36    }
37    else
```

Listing 4.3: Codeausschnitt zu Collision Detection

Codeausschnitt: Regler Glättung

4.3.4 Implementierung verschiedener Fahrsituationen

Das Konzept sieht die Aufteilung der zu fahrenden Strecke in vier wiederkehrende Fahrsituationen vor:

- Geradeausfahrt
- Kurvenfahrt
- Übergang von Geraden- zu Kurvenfahrt
- Übergang von Kurven- zu Geradenfahrt

Je nach Situation wird zwischen einem Regler für Geradeausfahrt und einem Regler für Kurvenfahrt umgeschaltet. Die Übergangszustände dienen lediglich der stufenweisen Geschwindigkeitsanpassung beim Wechsel zwischen den beiden Streckenabschnitten. Die Umschaltung zwischen den verschiedenen Reglern erfolgt in Abhängigkeit vom gewählten Fahrmodus (siehe Unterunterabschnitt 4.3.6).

Der Regler für die Geradeausfahrt zeichnet sich durch aus, während der Regler für die Kurvenabschnitte . (.....) (.....)

4.3.5 Einleitung eines Spurwechsels

Die Implementierung eines Spurwechsels eröffnet vielfältige Möglichkeiten. Während ein Spurwechsel auf geraden Abschnitten der Strecke unproblematisch ist, kann es im

Zusammenhang mit Kurven zu Problemen kommen. Wird eine Kurve auf der äußeren Spur durchfahren, so ist während der Kurvenfahrt grundsätzlich kein Wechsel auf die innere Spur möglich. Die konstruktive Begrenzung des Lenkwinkels wird durch die Kurvenfahrt schon komplett ausgenutzt, sodass kein Spielraum für ein Ausweichen nach innen mehr bleibt. Weniger problematisch ist das Ausweichen von der inneren auf die äußere Spur während einer Kurvenfahrt.

Auf der Rennstrecke des Projektseminars gibt es nur zwei Varianten eines Spurwechsels: von der rechten auf die linke Spur, und von der linken auf die rechte Spur. Grundsätzlich können diese Wechsel vorgenommen werden, indem der Regler-Sollwert auf den jeweils anderen Fahrbahnrand gesetzt wird. Dazu muss jedoch zunächst sichergestellt werden, dass sich überhaupt beide Fahrbahnmarkierungen im Sichtfeld der Kamera befinden. Daher wird (....)

4.3.6 Fahrmodi und Wettbewerb

Im Rahmen dieses Projektseminars sind zwei Aufgaben zu erfüllen: das Absolvieren einer kompletten Runde in möglichst geringer Zeit, und die Hinderniserkennung mit Spurwechsel bei möglichst wenigen Linienübertritten. Um beide Aufgaben möglichst gut zu lösen werden zwei verschiedene Fahrmodi implementiert, die sich durch die Fahrgeschwindigkeit und die Kriterien zur Wahl der Fahrspur unterscheiden.

Fahrmodus 1: Rundenzeit

Im ersten Fahrmodus liegt die Herausforderung in der Minimierung der Rundenzeiten. In diesem Modus wird ohne Hindernisse auf der Strecke gefahren, daher ist die Hinderniserkennung abgeschaltet. TODO

Fahrmodus 2: Hinderniserkennung und Spurwechsel

In diesem Modus spielt die Fahrgeschwindigkeit eine untergeordnete Rolle, weshalb sie auf einen konstanten und relativ niedrigen Wert gesetzt wird. Zunächst orientiert sich das Fahrzeug am rechten oder linken Fahrbahnrand und folgt der Spur. Solange sich kein Hindernis auf der Strecke befindet, wird dieser Zustand beibehalten.

Ein Spurwechsel setzt voraus, dass dem Fahrzeug die eigene Position auf der Fahrbahn bekannt ist. Da dem Fahrzeug vorgegeben wird auf welcher Fahrspur es fahren soll, und die Regler ein schnelles Regelverhalten besitzen, kann im Allgemeinen davon ausgegangen werden, dass sich das Fahrzeug auf der vorgegebenen Fahrspur befindet.

Sobald ein Hindernis in das Sichtfeld der Kamera eintritt liefert die Bildverarbeitung dessen Koordinaten. Da auch die Koordinaten der beiden Fahrspuren bekannt sind, kann aus den Daten ermittelt werden, ob sich das Hindernis auf der aktuellen Fahrspur des Autos befindet. Befindet sich das Hindernis auf der eigenen Fahrspur, wird rechtzeitig ein Spurwechsel veranlasst. Andernfalls wird das Hindernis ignoriert und die Fahrt fortgesetzt.

Der Spurwechsel wird durch die Erkennung eines Hindernisses eingeleitet. Die Bildverarbeitung liefert dabei die Positionen des rechten und linken Randes eines Hindernisses, sodass neben der Position auch dessen Breite bekannt ist (siehe Unterabschnitt 4.3). Für den Spurwechsel entscheidend ist die Frage, ob sich das erkannte Hindernis über-

haupt auf der gleichen Fahrspur wie das Fahrzeug befindet. Ist das nicht der Fall, stellt es kein Hindernis dar und kann ignoriert werden.

```
  collision_protection, no collision with objects
  double range from usf value
  return true, if there is an object
*/
bool collision_protection(double range)
{
  // initialization
  if(usf_flag)
  {
    for(int i = 0; i < RANGE_OF_USF_AVERAGE; i++)
    {
      usf_history[i] = 0;
    }
    usf_flag = false;
    return true;
  }
  // refresh array
  if(range > 0)
  {
    for(int i = 0; i < RANGE_OF_USF_AVERAGE - 1; i++)
    {
      usf_history[i + 1] = usf_history[i];
    }
      usf_history[0] = range;
  }
  // average
  double average = 0;
  for(int i = 0; i < RANGE_OF_USF_AVERAGE; i++)
  {
    average += usf_history[i];
  }
  average = average / RANGE_OF_USF_AVERAGE;
  if(average < 0.35)
  {
    return true;
  }
  else
```

Listing 4.4: Codeausschnitt zu Collision Detection

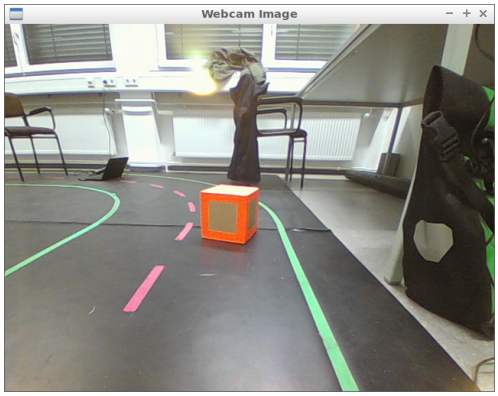
Codeausschnitt: Hinderniserkennung

5 Probleme

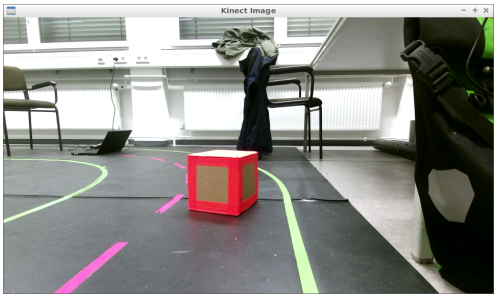
Das folgende Kapitel beschäftigt sich mit den Problemen, die während der Implementationsphase aufgetreten sind und wie diese gelöst wurden. Hauptsächlich waren dies Probleme mit der Kinect, der Halterung der Weitwinkelkamera, der Belichtungswerte und des Fahrwerks.

5.1 Kinect - unnatürliche Farben

Die Kamera der Kinect hat eine unnatürliche Farbdarstellung. Es sind starke Unterschiede zwischen Kamerabild und der Wirklichkeit zu erkennen.



(a) Webcam Original



(b) Kinect Original

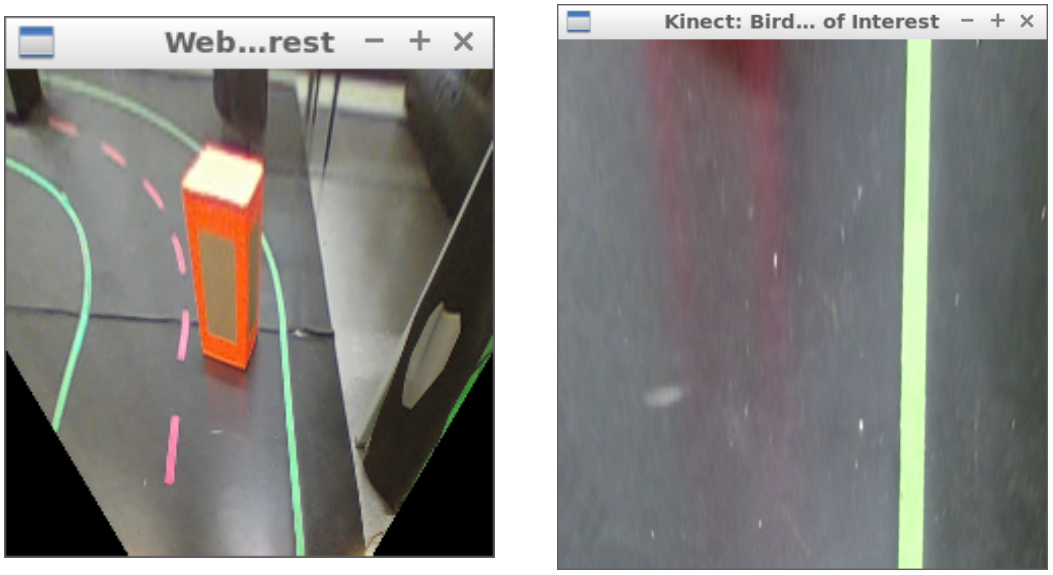
Abbildung 5.1: Farbdarstellung von Webcam und Kinect

Die Farben der Kinect sind übersaturiert und haben teilweise nicht einmal den richtigen Farbton.

5.2 Kinect - kleines Sichtfeld

Das Sichtfeld der Kinect Kamera fällt relativ klein aus. Das hat auch damit zu tun, dass die Kinect in einem Winkel auf dem Fahrzeug angebracht ist, mit dem das Sichtfeld erst circa 30cm vor dem Fahrzeug beginnt. Dadurch kann der Abstand des Fahrzeuges zur Fahrbahnbegrenzung zum aktuellen Zeitpunkt nicht ohne weiteres bestimmt werden. Zudem ist die Kamera in der Kinect am rechten Rand platziert. Demzufolge ist die linke Fahrbahnbegrenzung häufig nicht im Sichtfeld. Dieser Effekt wird noch durch die Transformation zum Bird-Eye View verstärkt.

Um eine robuste Fahrbahnerkennung zu implementieren, sind anwendungsoptimierte Eingangsdaten von großer Wichtigkeit. In unserem Fall benötigen wir eine natürliche Farbdarstellung, ein ausreichend großes Sichtfeld, das möglichst nah am Fahrzeug beginnt und eine flüssige Bildrate. Diese Anforderungen werden alle von der Webcam erfüllt. Die Webcam kann zudem sehr flexibel auf dem Fahrzeug angebracht werden.
Die Halterung der Webcam verursacht allerdings ein weiteres Problem, da eine geringfügige Änderung des Kamerawinkels deutliche Auswirkungen auf die Transformation zum Bird-Eye View hat. Wenn der Bird-Eye View nicht mehr korrekt berechnet wird, funktioniert die Erkennung der Fahrsituation nicht mehr gut. Genauer wird nicht mehr



(a) Webcam Bird-Eye View und Bildausschnitt

(b) Kinect Bird-Eye View und Bildausschnitt

Abbildung 5.2: Sichtfeld von Webcam und Kinect

erkannt, wann sich das Fahrzeug auf einer Geraden befindet. Da die Regelung abhängig von der Fahrsituation ist, kann das Fahrverhalten dadurch erheblich schlechter werden. Eine stabile Befestigung der Webcam ist daher wichtig. Näheres dazu im folgenden Abschnitt.

5.3 Befestigung der Weitwinkelkamera

Während die Kinect fest mit dem Fahrzeug verschraubt ist, stellt sich bei Verwendung der Weitwinkelkamera die Frage nach deren Positionierung und Befestigung. Der mitgelieferte Standfuß kann nicht sinnvoll auf dem Fahrzeug fixiert werden, da er nur für stationären Betrieb geeignet ist. Zudem kommt es durch die Vibrationen im Fahrbetrieb zu einer schleichenden Veränderung des Kamerawinkels mit den in Unterabschnitt 5.2 beschriebenen Auswirkungen. Um eine stabile und auch optisch ansprechende Lösung zu finden, wurde mittels CAD-Software eine Kamerahalterung entworfen und mit einem 3D-Drucker ausgedruckt. Dies bietet gleich mehrere Vorteile:

- Die Kameraposition kann frei gewählt werden
- Die Halterung ist stabil, aber sehr leicht
- Aufgrund des modularen Aufbaus der Halterung ist ein späterer Umbau sehr einfach möglich
- Die Kinect Kamera kann demontiert werden, dies entlastet die Vorderachse

Die Halterung besteht aus einem unteren Teil zur Befestigung am Fahrzeug, einem Mittelteil der das Emblem der Gruppe zeigt, sowie einem oberen Teil, auf dem die

Webcam montiert wird. Abbildung 5.3 zeigt das CAD-Modell, der reale Aufbau ist in Abbildung 2.1 zu sehen.

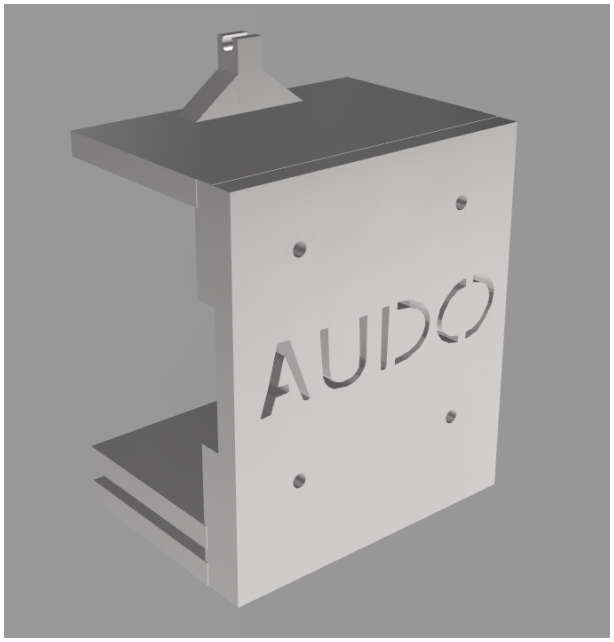


Abbildung 5.3: 3D-Modell der Kamerahalterung

5.4 Weitwinkelkamera verändert Belichtung zur Laufzeit

Die Weitwinkelkamera passt ihre Belichtung an die aktuellen Lichtverhältnisse zur Laufzeit an. Das führt dazu, dass selbst kleine Veränderungen der Lichtverhältnisse die Belichtung ändern, wie zum Beispiel Lichtspiegelungen auf dem Boden. Wenn sich die Belichtung ändert, ändern sich auch schlagartig die Farbwerte. Um keine komplizierte Anpassung der Filterparameter während der Laufzeit vornehmen zu müssen, muss die Belichtung konstant gehalten werden. Das ist bei dieser Weitwinkelkamera nicht in der Software einstellbar. Deshalb wurde ein pragmatischer Ansatz gewählt, um dieses Problem zu lösen. Die Weitwinkelkamera wird mit einer LED Leiste geblendet, welche über das 12V-Boardnetz gespeißt wird. Damit nimmt die Weitwinkelkamera die Umgebung immer sehr hell wahr. Leichte Veränderungen der tatsächlichen Lichtverhältnisse werden durch das Blenden einfach herausgefiltert. Das Ergebnis ist eine konstante Belichtung und somit auch konstante Farbwerte.

5.5 Fahrwerk

Die Räder kollidieren bei zu hohem Lenkwinkel mit dem Gehäuse, deswegen wird der maximale Lenkwinkel auf einen ausgemessenen Wert begrenzt.
Die Lenkmechanik des Fahrzeugs ist aus Plastik und nicht sonderlich genau gefertigt. Das hat zur Folge, dass die Lenkung ein merkliches Spiel hat. Dies bedeutet, dass kleine Veränderungen am Lenkwinkel keine realen Auswirken haben. Außerdem liegt relativ viel Last auf der Vorderachse, weil der Akku und die Kinect vorne im Fahrzeug platziert

sind. Dadurch wird die Lenkmechanik noch zusätzlich belastet. Bei höheren Geschwindigkeiten sollte sich dieser Effekt allerdings abschwächen. Durch diese Effekte kann sich nicht darauf verlassen werden, dass nach dem Einstellen eines Lenkwinkels, dieser auch tatsächlich anliegt. In der Praxis wird allerdings immer mindestens so schnell gefahren, dass dieser Effekt keine allzu großen Auswirkungen hat.
Ein weiteres Problem besteht darin, dass der Lenkwert der an die `UC_Bridge` gesendet wird, sich nicht linear in einen Lenkwinkel umrechnen lässt. Die Beziehung zwischen Lenkwert und Lenkwinkel auszumessen gestaltet sich schwierig. Da diese Beziehung stark von der Geschwindigkeit und der Last auf die Lenkmechanik abhängt. Wenn der tatsächliche Lenkwinkel durch passende Sensoren regelmäßig gemessen werden würde, könnte eine Regelung für den Lenkwinkel konzipiert werden. Dieses Problem wurde jedoch nicht weiter beachtet, da sich auf eine möglichst robuste Regelung zur Fahrspurhaltung konzentriert wurde und diese das Problem teilweise ausgleichen kann. Um die Lenkmechanik nicht unnötig zu belasten, wurde eine möglichst ruhige Regelung implementiert.

Liste of todos entfernen ganz am Ende

Todo list

A Anhang