

DIPLOMARBEIT

Gesamtprojekt

Camera Controlled Swarm Robots**Positionerkennung**

Mottl Mario 5BHEL Betreuer: Dipl.-Ing Josef Reisinger

Swarm Control

Clemens Pruggmayer 5BHEL Betreuer: Dipl.-Ing Josef Reisinger

Visualisierung und Simulation

Michael Reim 5BHEL Betreuer: Dipl.-Ing Josef Reisinger

Schuljahr 2020/21

Abgabevermerk:

Datum: TT.MM.JJJJ

übernommen von:



**Höhere Technische Bundeslehranstalt Hollabrunn
Höhere Lehranstalt für Elektronik und Technische Informatik**

EIDESSTATTLICHE ERKLÄRUNG

Ich erkläre an Eides statt, dass ich die vorliegende Diplomarbeit selbständig und ohne fremde Hilfe verfasst, andere als die angegebenen Quellen und Hilfsmittel nicht benutzt und die den benutzten Quellen wörtlich und inhaltlich entnommenen Stellen als solche erkenntlich gemacht habe.

Mario Mottl

Clemens Pruggmayer

Michael Reim

Hollabrunn, am 5. April 2019

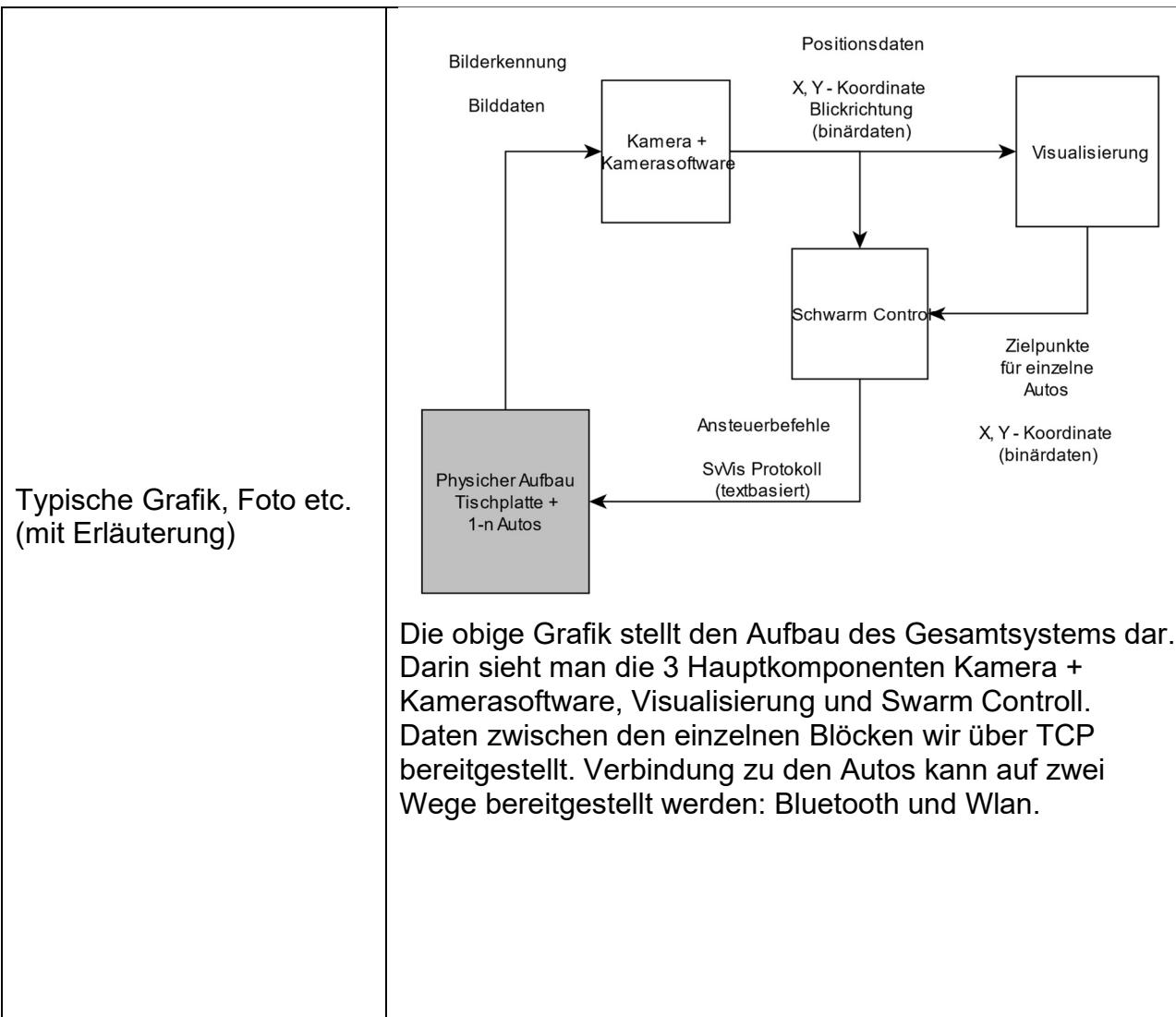
**DIPLOMARBEIT
DOKUMENTATION**

Namen der Verfasser/innen	Michael Reim, Clemens Pruggmayer, Mario Mottl
Jahrgang Schuljahr	5BHEL
Thema der Diplomarbeit	Camera Controlled Swarm Robots
Kooperationspartner	-

Aufgabenstellung	Es sollen mehrere 1 bis n viele autonome Fahrzeuge (STM32F107RB + MDDS Board) über eine 1,5m erhöhten Kamera erfasst werden. Positionen der Fahrzeuge sollen in (x/y) Koordinaten verwandelt werden. Diese Daten sollen an die Visualisierung und Simulation geschickt. Diese erzeugt über selbst gezeichnete Bilder einen Weg für die Fahrzeuge. Der Weg soll an das Swarm Controll weitergeschickt werden welche die Positionsdaten mithilfe der Kamera in Fahrkommandos umwandelt.
------------------	---

Realisierung	Zur Erfassung der Autos wurde eine „DFK 33UX273“ von ImagingSource verwendet. Diese wird per USB an einen Laptop/Computer angeschlossen. Für die Visualisierung und Simulation wurde eine 3D Visualisierungssoftware in C++ geschrieben. Welche die Autos in Echtzeit am Bildschirm anzeigt. Swarm Controll wurde mithilfe des SvVis Protokoll realisiert. Auf den Autos(Cortex M4) läuft eine selbstgeschriebene RTOS-Software, die die Kommandos des SVIS Protokoll in Bewegungen umwandelt.
--------------	---

Ergebnisse	Eine alte Version des SvVis Protokolls wurde abgeändert und verbessert. Eine Teststrecke wurde aufgebaut und mit einer Halterung für die Kamera erweitert. Positionserkennung wurde mithilfe von Python realisiert. Die Autosteuerungssoftware wurde selbstständig mit RTOS realisiert. Visualisierung wurde von Grund auf in C++ + OpenGL geschrieben für den speziellen Anwendungsfall.
------------	--



Die obige Grafik stellt den Aufbau des Gesamtsystems dar. Darin sieht man die 3 Hauptkomponenten Kamera + Kamerasoftware, Visualisierung und Swarm Controll. Daten zwischen den einzelnen Blöcken werden über TCP bereitgestellt. Verbindung zu den Autos kann auf zwei Wege bereitgestellt werden: Bluetooth und Wlan.

Teilnahme an Wettbewerben, Auszeichnungen	
---	--

Möglichkeiten der Einsichtnahme in die Arbeit	HTL Hollabrunn Anton Ehrenfriedstraße 10 2020 Hollabrunn
---	--

Approbation (Datum / Unterschrift)	Prüfer/Prüferin	Direktor/Direktorin Abteilungsvorstand/Abteilungs vorständin
---------------------------------------	-----------------	--

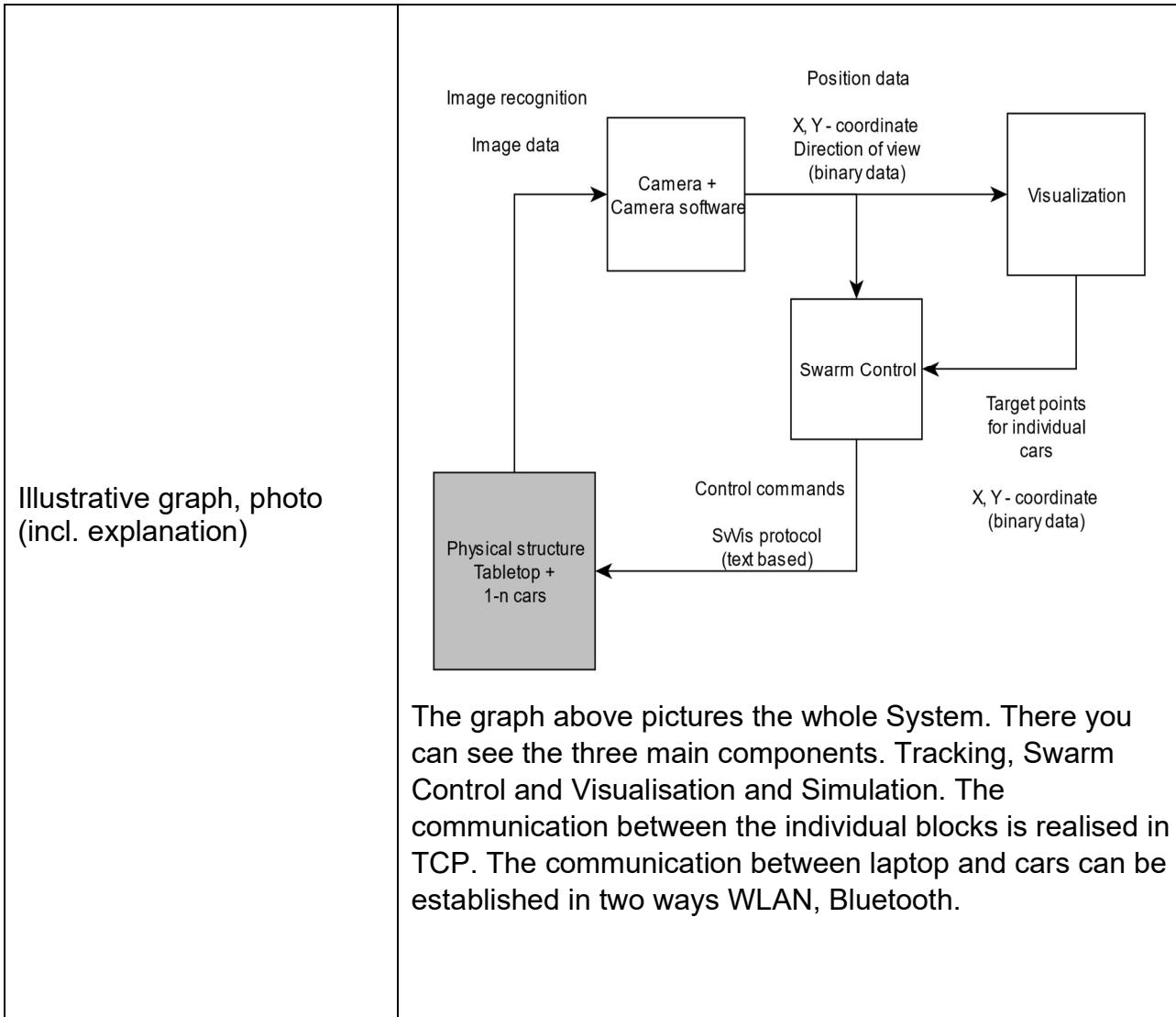
**DIPLOMA THESIS
Documentation**

Author(s)	Michael Reim, Clemens Pruggmayer, Mario Mottl
Form Academic year	5BHEL
Topic	Camera Controlled Swarm Robots
Co-operation partners	-

Assignment of tasks	One or more autonomous vehicle (powered by an STM32F107RB + MDDS Board) should get detected by a camera that is attached 1,5 metres above the table. The positions of the vehicles should be converted into an x-y coordinate grid. The produced data should then be sent to our visualisation and simulation for correction purposes. A picture of a path should be drawn. The generated path should then be sent off to the Swarm Control. Where the path will be transformed into vehicle commandos.
---------------------	---

Realisation	For proper detection, an “DFK 33UX273” from Imaging Source was used. It is connected via USB to a Laptop. The visualisation and simulation were programmed in “C++” + “OpenGL” which draws the cars onto the screen in Realtime. Swarm Control uses a technology called “SvVis”. The vehicles run on a self-implemented RTOS-Software (Real Time Operating System) which transforms the vehicle commandos into engine movement.
-------------	---

Results	An older version of the “SvVis” was used and altered to fit our purposes. A test track with a bracket for the camera was built. The tracking software was programmed in python. Vehicle Control software was written with RTOS. The visualisation and simulation were programmed in “C++” + “OpenGL” for our specific purpose.
---------	--



Participation in competitions	
Awards	-

Accessibility of final project thesis	HTL Hollabrunn Anton Ehrenfriedstraße 10 2020 Hollabrunn
---------------------------------------	--

Approval (Date / Signature)	Examiner/s	Head of Department / College
--------------------------------	------------	------------------------------

DA Antrag und unterschriebene Erklärung aus der Diplomarbeitsdatenbank einfügen

Inhaltsverzeichnis

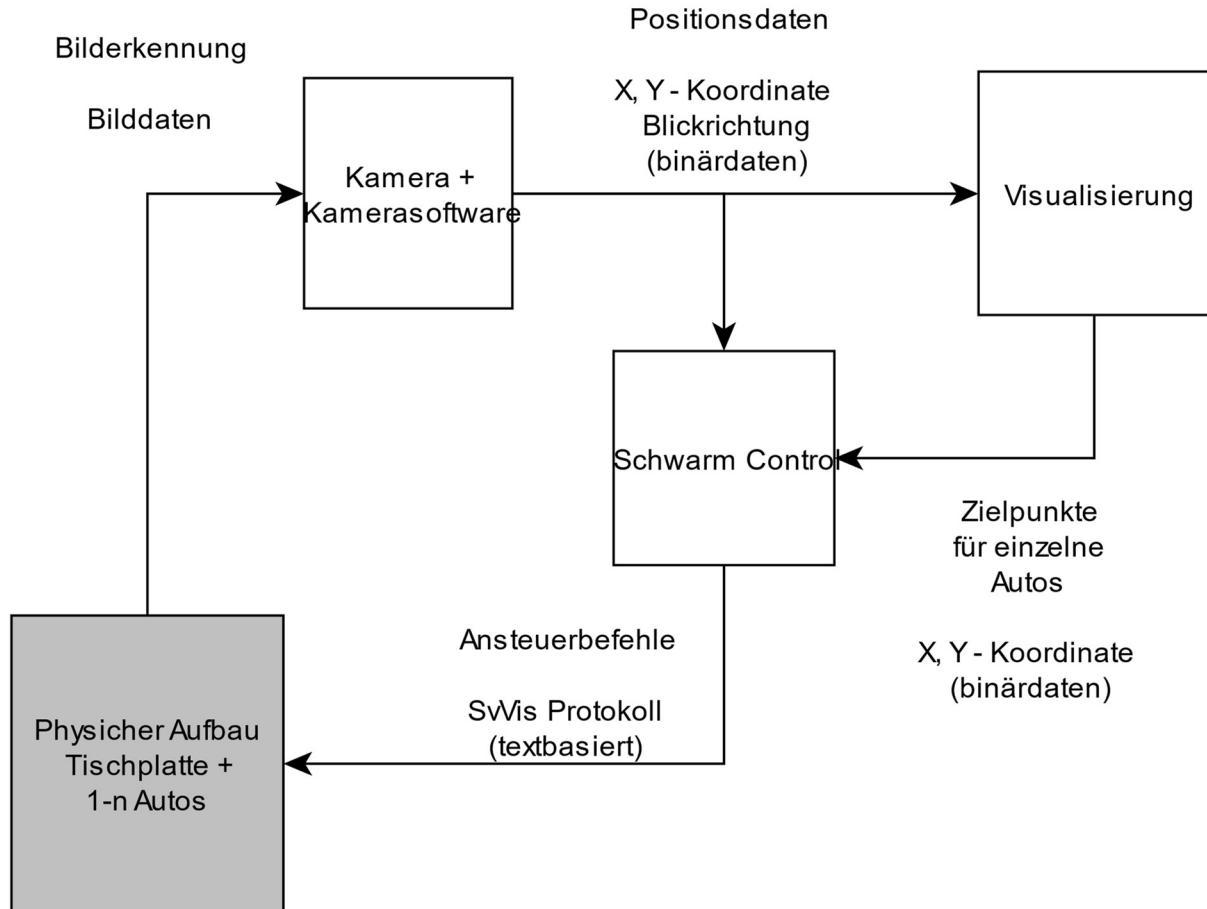
1	Camera Controlled Swarm Robots	11
1.1	Blockschaltbild	11
1.2	Funktionsbeschreibung	11
2	Positionserkennung	12
2.1	Übersicht	12
2.2	Aufbau	13
2.3	Kamera	14
2.3.1	Korrekt Platziere der Kamera	14
2.3.2	Kalibrierung der Kamera (GUI)	15
2.3.3	Kalibrierung der Kamera (Code / Python)	16
2.4	Erkennung Positions LEDs	17
2.4.1	Algorithmus zur Erkennung	18
2.4.1.1	Blob Detection	18
2.4.1.1.1	Beispielparameter	18
2.4.1.1.2	Filtern von Blobs nach Farbe, Größe, Form	19
2.4.1.2	Kalibrierung der Erkennung	20
2.4.2.1	Feature	20
2.4.2.2	Wiederherstellen	21
2.4.2.3	21
2.4.3	Berechnung der Position	22
2.5	Zusammensetzung von Bildpunkte zu Autos	24
2.5.1	Optimierung	25
2.6	Auswertung der Bilddaten	26
2.7	Kommunikation mit Simulation / Visualisierung	27
3	Visualisierung und Simulation	28
3.1	Übersicht Softwarearchitektur	28
3.1.1	Blockschaltbild	28
3.1.2	GUI	28
3.1.3	Engine	28
3.1.4	Client	28
3.1.5	Modelle, Texturen, Shader	29
3.1.6	Path Generierung	29
3.2	Graphical User Interface	30
3.2.1	Aufbau	30
3.2.2	Befehle	30
3.2.2.1	Befehle für die Simulation	30
3.2.2.2	Befehle für den realen Aufbau	31
3.2.3	Realisierung	32
3.2.3.1	Erstellung und Initialisierung vom GUI	32
3.2.3.2	Erstellung der Events für das GUI	33
3.3	Aufbau der Engine	34

3.3.1 Blockschaltbild.....	34
3.3.2 Graphisches Rendering	34
3.3.2.1 Texturen, Modelle und Shader	34
3.3.2.2 Grafik Renderer mit OpenGL	37
3.3.3 Fahrzeug AI	40
3.3.3.1 Generierung des Weges	40
3.3.3.2 Aufbereitung der Goals	45
3.3.3.3 Senden der Goals zur Visualisierung	46
3.3.3.4 Verarbeitung der Goals in der Visualisierung	48
3.4 Erstellung von 3D-Modellen.....	48
3.4.1.1 Tisch.....	48
3.4.1.2 Fahrzeug	50
3.4.1.3 Komplette 3D-Szene	51
3.5 Bewegungen im Dreidimensionalen Raum.....	52
3.5.1.1 Kamerarotation.....	52
3.5.1.2 Kamerabewegung	53
3.6 Auswertung von erhaltenen Positionsdaten	55
3.7 Übertragung zu Fahrzeug Software.....	59
4 Fahrzeug Software	60
4.1 Übersicht der Architektur	60
4.2 Kommunikation mit Swarm Controll.....	60
4.2.1 Kommunikationsprotokoll	60
4.2.1.1 Übersicht Software-Architektur der SvVis-API	61
4.2.1.2 Decodierung	62
4.2.1.3 Encodierung	63
4.3 Hardware Ansteuerung.....	64
4.3.1 LED-Ansteuerung	64
4.3.2 Kommunikationsmodule	64
4.3.2.1 Allgemein	64
4.3.2.2 DAP Link	64
4.3.2.3 Bluetooth	64
4.3.2.4 WLAN	65
4.3.3 Motor Ansteuerung	65
5 Swarm Control.....	66
6 SvVis – Visualisierung - crashkurs	67
6.1 Verbindung aufbauen	67
6.1.1 COM-Port	67
6.1.2 TCP/IP – Verbindung.....	68
6.2 Daten senden / Empfangen	69
7 Ergebnisse	71
7.1 Funktionalität Positionserkennung.....	71
7.1.1 Erkennung der Fahrzeuge	71
7.1.2 Tracking.....	73

7.1.2.1 Position 1	73
7.1.2.2 Position 2	73
7.2 Steuerung der Fahrzeuge.....	74
7.3 Simulationstest mit Pseudodaten	74
7.4 Steuerung der Fahrzeuge.....	76
7.5 Steuersoftware Funktionalitätstest.....	76
8 Anhang	78
8.1 Inbetriebnahme (f. 4Klasse TdoT)	78
8.1.1 Inbetriebnahme Fahrzeug Software	80
8.1.2 Inbetriebnahme Kamera Software	80
8.1.3 Inbetriebnahme Visualisierung / Simulation.....	81
8.2 Projektmanagement	82
8.2.1 Projektplan	82
8.2.1.1 Mottl	82
8.2.1.2 Reim.....	83
8.2.1.3 Pruggmayer.....	83
8.2.2 Projekttagebuch.....	84
8.2.2.1 Mottl	84
8.2.2.2 Reim.....	86
8.2.2.3 Pruggmayer.....	91
8.2.3 Projektkosten.....	93
9 Quellenverzeichnis	93
9.1 Bücher	93
9.2 Onlinemedien	93
9.2.1 [1] Wikipedia HSV.....	93
9.2.2 Dokumentation CMSIS-RTOS	93
9.3 Zeitschriften.....	93

1 Camera Controlled Swarm Robots

1.1 Blockschaltbild

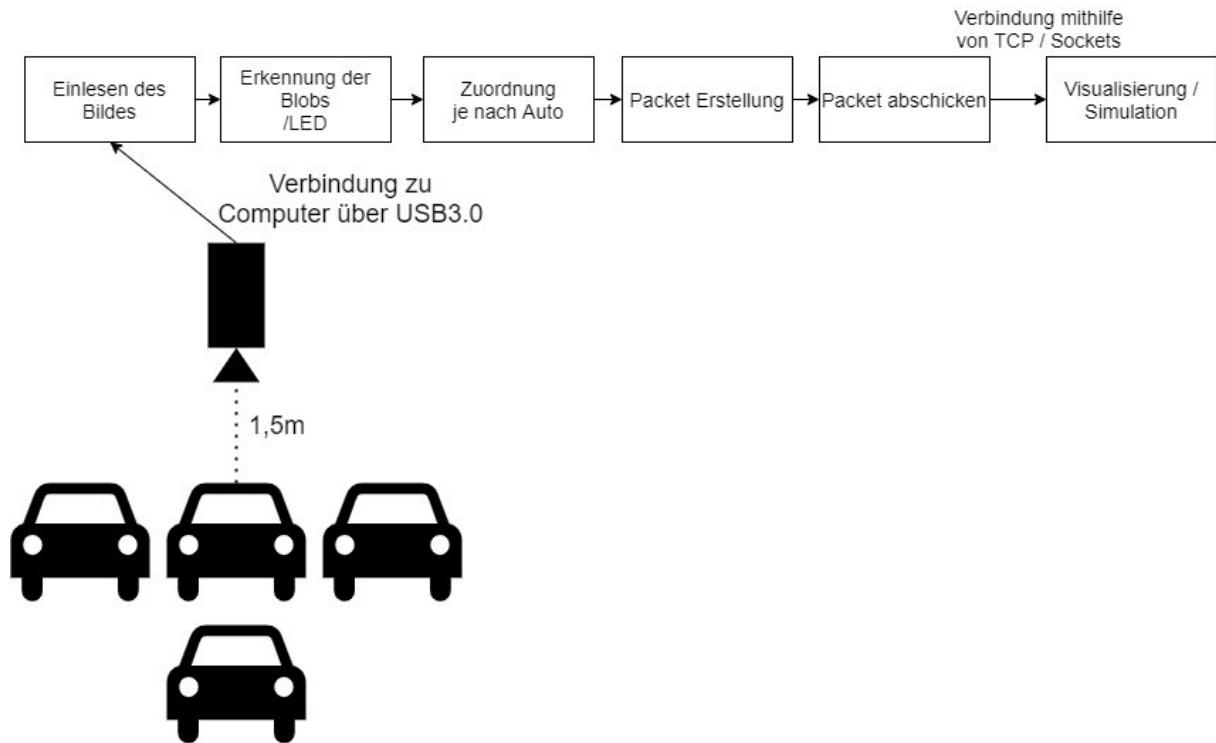


1.2 Funktionsbeschreibung

Es sollen 1 bis n viele autonome Fahrzeuge (STM32F107RB + MDDS Board) über eine 1,5m erhöhte Kamera erfasst werden. Positionen der Fahrzeuge sollen in (x/y) Koordinaten verwandelt werden. Diese Daten sollen an die Visualisierung und Simulation geschickt werden. Diese erzeugt über selbst gezeichnete Bilder einen Weg für die Fahrzeuge. Der Weg soll an das Swarm Control weitergeschickt werden, welche die Positionsdaten mithilfe der Kamera in Fahrkommandos umwandelt. Die Swarm Control sendet die Fahrkommandos über das SvVis-Protokoll an die Autos, welche entsprechend der Fahrkommandos auf der Tischplatte kontrolliert fahren.

2 Positionserkennung

2.1 Übersicht

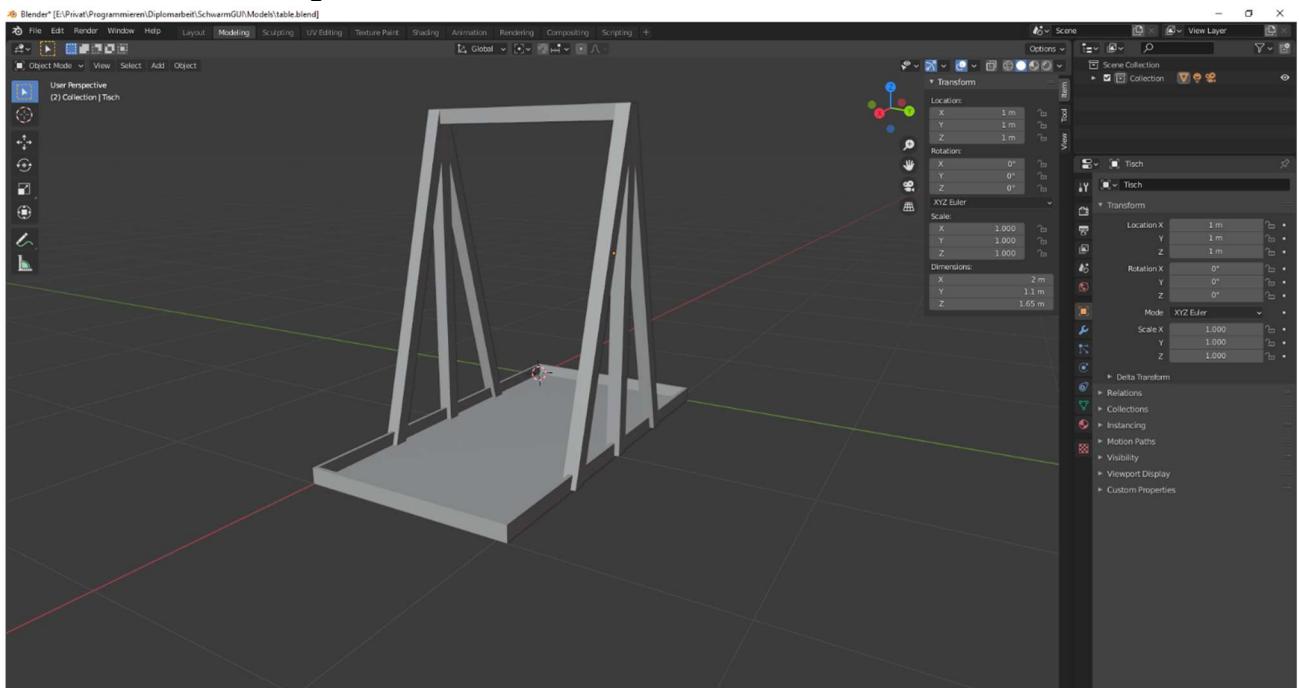


Es werden mehrere Autos mithilfe einer Kamera die sich 1,5m über ihnen befindet. erfassen. Diese Autos sind jeweils mit mehreren Positions LEDs ausgestattet. Die Kamera sendet die Bilddaten über ein USB 3.0 Kabel an einen Computer. Die Software wertet die Bilddaten aus und erkennt die LEDs (Blobs) diese werden danach zugewiesen welche Punkte zu welchem Auto gehören. Ist dieses erfolgt werden einzelne Pakete erstellt diese beinhalten jeweils die Position des Autos, dessen Identifikationsnummer und dessen Orientation. Diese Pakete werden danach einzeln an die Visualisierung / Simulation gesendet.

Die Software wurde sowohl in Python als auch in C++ programmiert.

2.2 Aufbau

Der Aufbau wurde nach einer 3D Modellierung von Reim Michael nachgebaut mit kleinen Veränderungen. Die Kamera sitzt oben und blickt herab auf die Tischfläche in einem Abstand von ~1,5m. Jedoch wird nicht die gesamte Tischfläche genutzt. Die Originalmaße des Tisches betragen 2m*1,05m. Die Kamera kann aber nicht die ganzen zwei Meter Länge.



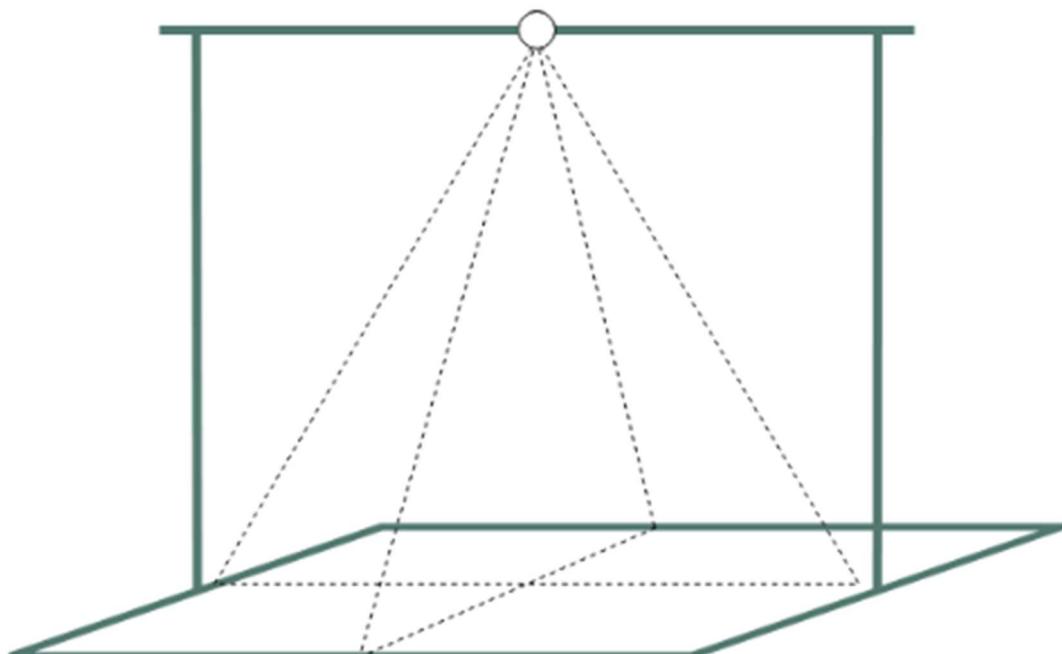
2.3 Kamera

Bei der verwendeten Kamera handelt es sich um eine „DFK 33UX273 USB 3.0 Farb-Industriekamera“. Diese ist mit einem $1/29\text{ inch}$ Sony CMOS Pregius Sensor (IMX273) diese kann 1,6MP (1440x1080 Pixel) auflösen und hat eine Aktualisierungsrate von bis zu 238 Hz (238 fps) Diese ist über ein USB 3.0 Kabel mit einem Computer verbunden.

Es wird eine speziell angefertigte Linse benutzt um die Brennweite der Linse auf 1,5m zu ändern.



2.3.1 Korrekter Platzierung der Kamera



Es müssen vor Beginn die Grenzen des Sichtfeldes der Kamera abgesteckt werden. Fahren Autos außerhalb dieser Grenzen kann die Kamera sie nicht mehr erkennen. Darüber hinaus muss die Kamera auf die Lichtverhältnisse im Raum angepasst werden.

2.3.2 Kalibrierung der Kamera (GUI)

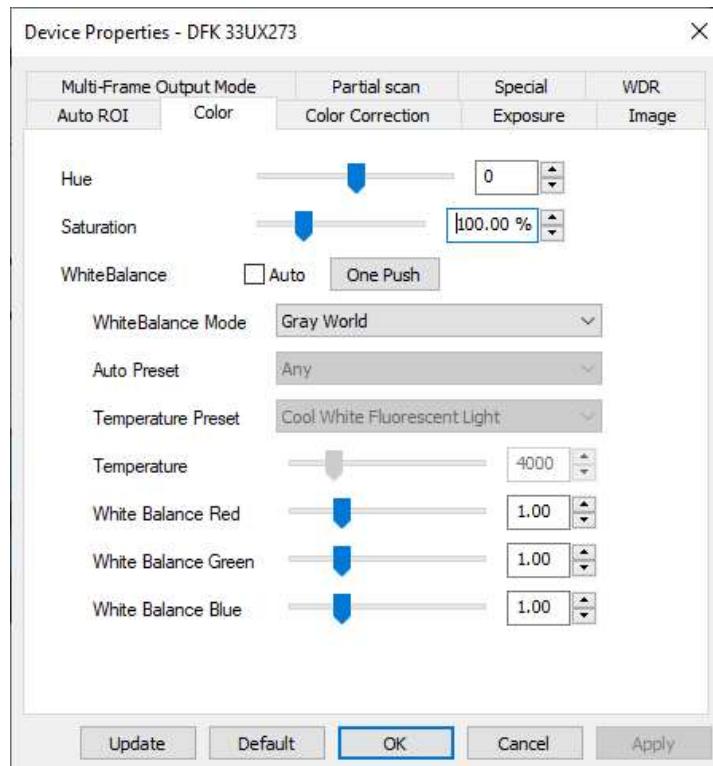
Die Kamera kann auf mehreren Wegen kalibriert werden. Im Programmcode als Befehlszeile oder in einer GUI (Graphical User Interface)

Dieses GUI erkennt automatisch die angeschlossene Kamera liest die Serien Nummer aus.

Video Format und Frame Rate (FPS) wie andere Parameter können hier auch gleich geändert werden.



Unter „Properties...“ kann man dann die erweiterten Eigenschaften der Kamera ändern. Wie zum Beispiel die Farbe/Temperatur etc.



Diese Werte können auch während dem Benutzen der Kamera „Live“ angepasst werden. Falls die Einstellungen nicht passen, kann man „Default“ benutzen, um auf die Werkseinstellungen zurückzusetzen. Die Einstellungen werden gespeichert und müssen nicht bei jedem Neustart neu gesetzt werden.

2.3.3 Kalibrierung der Kamera (Code / Python)

Wie schon erwähnt kann die Kamera auch im Programmcode eingestellt werden.

Man benötigt also keine GUI wenn man diese nicht haben möchte.

Es muss immer ein Kamera Objekt erstellt werden.

```
23 Camera = IC.TIS_CAM()
```

Mithilfe einer for schleife kann man über alle angeschlossenen Geräte iterieren, um sein gewünschtes Gerät zu finden.

```
26 Devices = Camera.GetDevices()
27 for i in range(len( Devices )):
28     print( str(i) + " : " + str(Devices[i]))
```

Man kann auch direkt die Kamera öffnen wenn man den Namen kennt. In diesem Fall der Name der Verwendeten Kamera.

```
31 Camera.open("DFK 33UX290 18810256")
```

Um das Videoformat und die FPS einzustellen verwendet man die eingebauten Methoden der Klasse.

```
41 Camera.SetVideoFormat("RGB32 (640x480)")
44 Camera.SetFrameRate( 30.0 )
```

Um einzelne Parameter von der Kamera zu ändern wie z.B. die Exposure Zeit zu ändern verwendet man die SetPropertyAbsoluteValue Funktion.

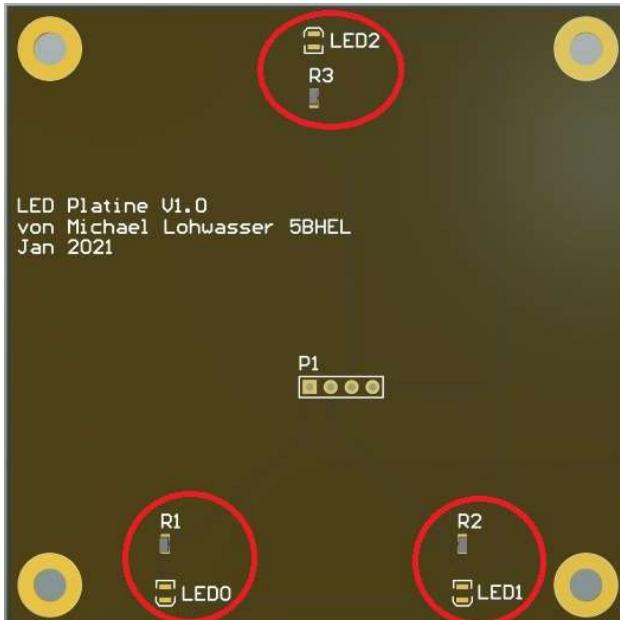
```
70 Camera SetPropertyAbsoluteValue("Exposure","Value",0.0303)
```

Um den Wert von den Einzelnen Parametern der Kamera auszulesen verwendet man nun die GetPropertyAbsoluteValue Funktion.

```
65 Camera.GetPropertyAbsoluteValue("Exposure","Value",ExposureTime)
```

2.4 Erkennung Positions LEDs

Für die Erkennung der Positions LEDs wird ein simpler Algorithmus verwendet. Durch die spezielle Anordnung der LEDs in einem gleichschenkeligen Dreieck kann man die ungefähre Position der anderen LEDs mit dem Satz des Pythagoras berechnen.



Um sicherzustellen, dass die Positions LEDs zu einem Auto und nicht zu einem anderen gehören. Werden immer nur zwei LEDs mit der Kamera erkannt und die Position der letzten LED wird berechnet.

Um die Erkennung möglichst genau zu machen muss am Anfang die Erkennungssoftware kalibriert werden. Dort werden die Abstände zwischen den LEDs in Pixel erkannt und als Referenzwerte für die weitere Erkennung der Autos benutzt.

Sobald die Kalibrierung abgeschlossen ist, wird über die Erkannten LEDs iteriert und sobald drei Bildpunkte auf die zuvor gemessenen Werte passen, werden diese als ein Auto zusammengefasst.

2.4.1 Algorithmus zur Erkennung

Für die Extrahierung der wichtigen Bildpunkte wird eine Klasse von OpenCV verwendet (SimpleBlobDetection). Diese Klasse ist besonders gut geeignet um blobs (zusammenhängende Bildpunkte mit derselben Farbe und Intensität) zu erkennen. Diese Klasse implementiert einfache Algorithmen, um diese zu extrahieren.

Konvertierung des Quellbildes in Binärbilder, indem man einen Schwellenwert mit mehreren

Schwellenwerten von minThreshold (einschließlich) bis maxThreshold (exklusiv) mit einem Abstand Threshholdstep zwischen benachbarten Schwellenwerten anwendet.

Extrahieren von verbundenen Komponenten aus jenem Binärbild mit findContours und berechnen von deren Zentren.

Gruppierung von Zentren aus mehreren Binärbildern anhand ihrer Koordinaten. Nahe Zentren bilden eine Gruppe, die einem Blob entspricht, der vom Parameter minDistBetweenBlobs gesteuert wird.

Schätzen aus den Gruppen die endgültigen Zentren der Blobs und ihre Radien und geben diese als Positionen und Größen der Schlüsselpunkte zurück.

2.4.1.1 Blob Detection

Es wird ein eher simpler Algorithmus benutzt, der durch die verschiedenen Parametern gesteuert wird.

2.4.1.1.1 Beispielparameter

```
// Setup SimpleBlobDetector parameters.  
SimpleBlobDetector::Params params;  
  
// Change thresholds  
params.minThreshold = 10;  
params.maxThreshold = 200;  
  
// Filter by Area.  
params.filterByArea = true;  
params.minArea = 1500;  
  
// Filter by Circularity  
params.filterByCircularity = true;  
params.minCircularity = 0.1;  
  
// Filter by Convexity  
params.filterByConvexity = true;  
params.minConvexity = 0.87;  
  
// Filter by Inertia  
params.filterByInertia = true;  
params.minInertiaRatio = 0.01;
```

- Threshold = Konvertierung des Quellenbildes in mehrere binäre Bilder. Dieses funktioniert, indem man das Bild in mehrere Verschiedene Bilder mit einem sogenannten Threshholdstep pro Bild aufteilt.
- Gruppieren = In jedem Binären Bild werden die weißen Pixel zusammen gruppiert. Diese werden auch Binäre Blobs genannt.
- Zusammenfügen = Der Mittelpunkt jedes Binären Blobs wird errechnet sofern sie näher als der minDistBetweenBlobs(Minimale erlaubte Distanz zwischen zwei Blobs) sind werden sie zusammengefügt.

2.4.1.2 Filtern von Blobs nach Farbe, Größe, Form

- **Farbe**

Um dieses Feature zu verwenden muss man **filterByColor = 1** setzen. Um die Blobfarbe zu ändern setzt man **blobColor** auf den gewünschten Wert zwischen 0 - 255

- **Zirkularität (Circularity)**

Dieses Feature misst wie nah ein Blob einen Kreis ähneln soll.

Die Zirkularität wird mit $\frac{4\pi F \text{ äche}}{\text{perimeter}^2}$ beschrieben.

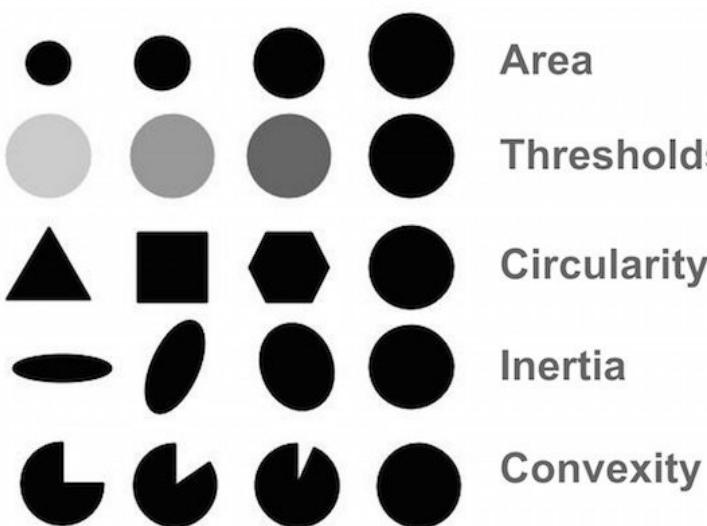
- **Konvexität (Convexity)**

Konvexität ist definiert als $\frac{\text{Fläche des Blobs}}{\text{Fläche des Convex Hull des Blobs}}$. Convex Hull ist die engste Konvexe Form, die die Form komplett umschließt.

- **Trägheitsverhältnis (Inertia)**

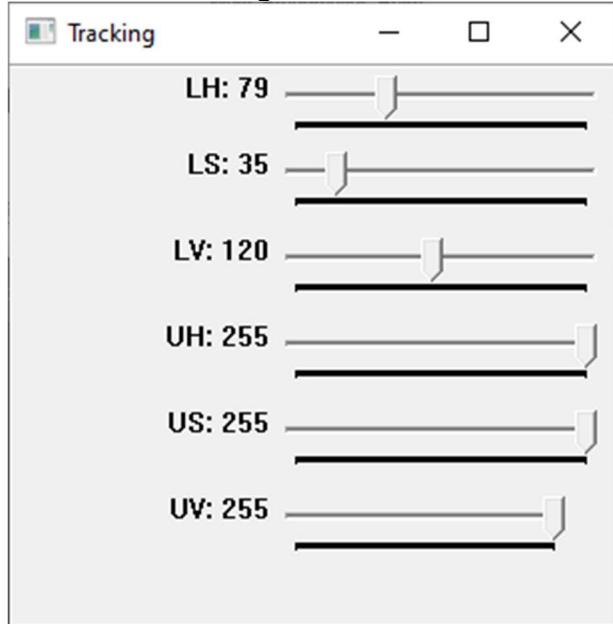
Dieser Parameter misst wie langgezogen die Form ist z.B. wie sehr ein Kreis verzogen ist und einer Ellipse ähnelt.

Hier sieht man eine bildliche Darstellung der Wirkung der einzelnen Parameter.



2.4.2 Kalibrierung der Erkennung

Um die Erkennung zu Kalibrieren verwendet man das eingebaute GUI.



Hier kann man die unteren (LH, LS, LV) und oberen (UH, US, UV) setzen.

Es sind die Koordinaten des HSV-Farbraumes [1]. Diese Farbraum ist definiert durch der Hilfe drei Koordinaten.

Farbwert (englisch **Hue**): Farbwinkel auf dem Farbkreis. (0° = Rot, 120° Grün, 240° für Blau)

Farbsättigung (**Saturation**): (0% Neutralgrau, 50%wenig gesättigte Farbe, 100% gesättigte reine Farbe)

Hellwert (**Value**): (0% keine Helligkeit, 100% volle Helligkeit)

Diese sollten immer so eingestellt werden damit das Referenzauto in allen möglichen Blickwinkeln erkannt werden kann.

2.4.2.1 Feature

Um es für den Benutzer besonders einfach zu machen werden die eingestellten Parameter für die Erkennung automatisch gespeichert und beim nächsten Start des Programms automatisch geladen. Somit können mehrere Tests schneller nacheinander getätigkt werden.

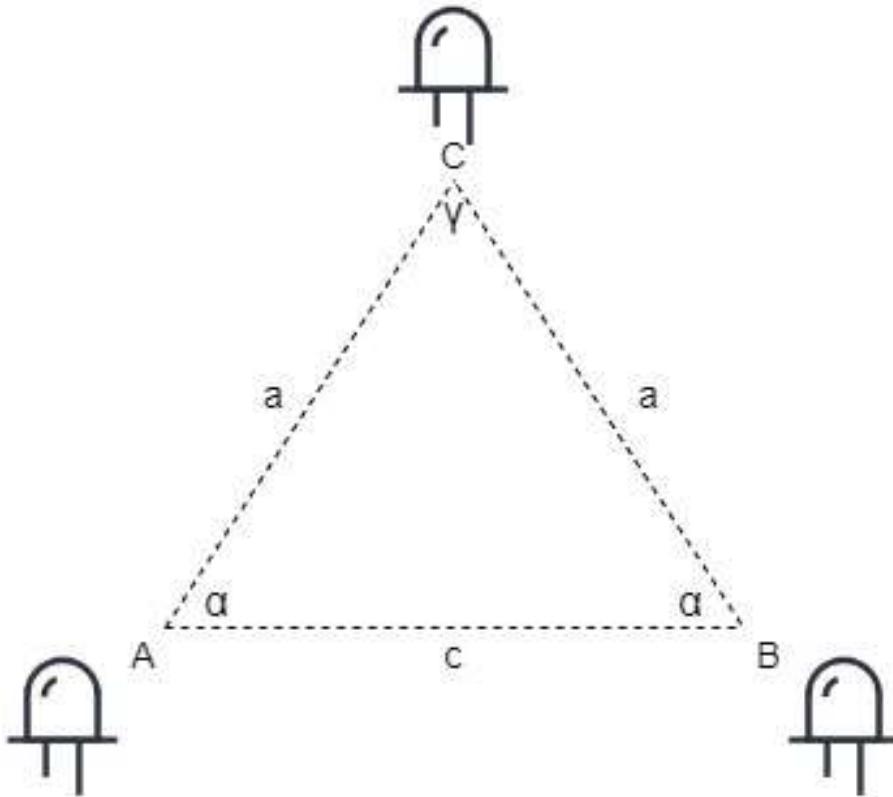
2.4.2.2 Wiederherstellen

```
SwarmDetection::SwarmDetection()
{
    //reads out the last values the user used
    std::ifstream f("settings.cfg");
    if (f.is_open())
    {
        f >> hvalues.l_h >> hvalues.l_s >> hvalues.l_v >> hvalues.u_h >> hvalues.u_s >
> hvalues.u_v;
        f.close();
    }
    //if none are found sets them to zero
    else
    {
        hvalues.l_h = hvalues.l_s = hvalues.l_v = hvalues.u_h = hvalues.u_s = hvalues.
u_v = 0;
    }
}
```

2.4.2.3

```
SwarmDetection::~SwarmDetection()
{
    //saves the last settings the user put in
    std::ofstream f;
    f.open("settings.cfg");
    f << hvalues.l_h << "\n" << hvalues.l_s << "\n" << hvalues.l_v << "\n" << hvalues.u_
h << "\n" << hvalues.u_s << "\n" << hvalues.u_v << "\n";
    f.close();
}
```

2.4.3 Berechnung der Position



Um eine möglichst fehlerfreie Erkennung zu gewährleisten werden Referenzabstände zwischen den drei LEDs des Referenzauto gemessen. Und anschließend als Referenz für alle anderen Autos verwendet.

$P_n \dots$ Koordinaten des Punkt n

$$\vec{AB} = P_B - P_A$$

$$\vec{BC} = P_C - P_B$$

$$\vec{AC} = P_C - P_A$$

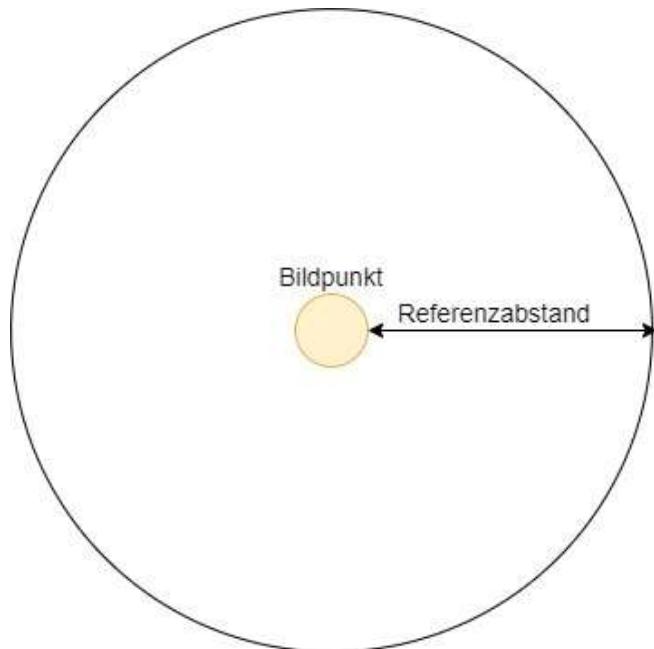
```
float SwarmDetection::getDistance(cv::KeyPoint p1, cv::KeyPoint p2)
{
    //Returns the distance between the two provided keypoints
    float a, b;
    a = p2.pt.x - p1.pt.x;
    b = p2.pt.y - p1.pt.y;
    return sqrt(a*a+b*b);
}
```

```
    std::cout << "[DEBUG]ABv = " << getDistance(keyPoints[0], keyPoints[1]) <<  
"Pixel" << std::endl;  
    std::cout << "[DEBUG]BCv = " << getDistance(keyPoints[1], keyPoints[2]) <<  
"Pixel" << std::endl;  
    std::cout << "[DEBUG]ACv = " << getDistance(keyPoints[0], keyPoints[2]) <<  
"Pixel" << std::endl;
```

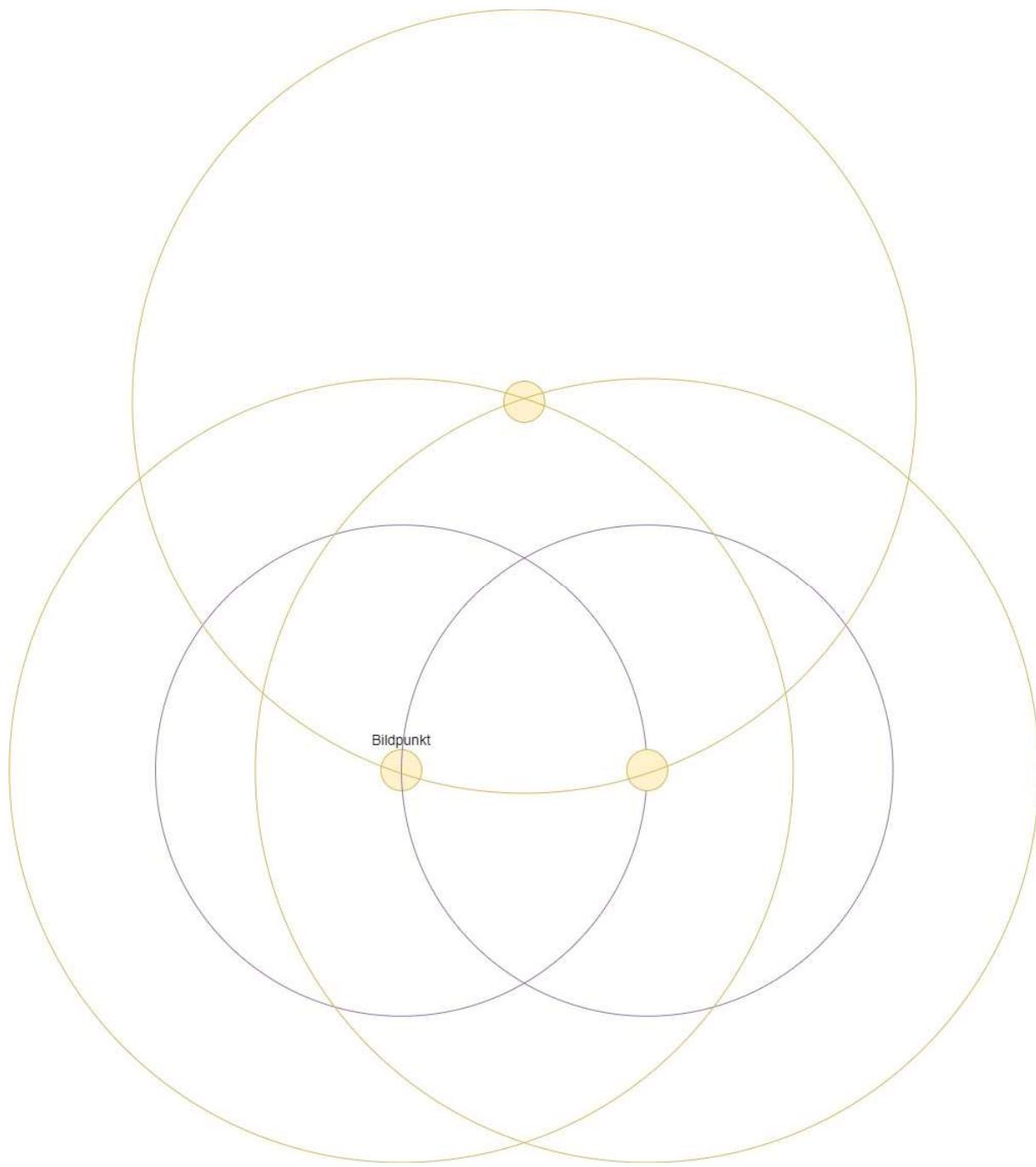
Aus der Symmetrie der Anordnung der LEDs erkennt man, dass $\overrightarrow{AC} \& \overrightarrow{BC}$ gleich groß seien müssen. Um Toleranzen und eventuellen Fehlerkennungen entgegenzuwirken wird bei jeder zuvor gemessenen eine Toleranz hinzugefügt. Aus Ergebnissen von mehreren ausführlichen Tests ist eine Toleranz von +10Pixel gewählt.

Durch diese Referenzwerte kann man bei jedem Bildpunkt berechnen in welchen Radius um den Bildpunkt sich eine andere LEDs / Bildpunkt befinden musst. Dieses wird rekursiv durchgesucht und sofern es drei richtige Anschläge gibt, ist ein Auto gefunden.

2.5 Zusammensetzung von Bildpunkte zu Autos



Dieser Kreis wird über alle Bildpunkte gelegt. Jeder Bildpunkt hat zwei Kreise mit den verschiedenen Abständen. Jedoch müssen bei einem Bildpunkt (C) der kleinere Kreis nicht berücksichtigt werden. Diese Erkennung ermöglicht eine fast perfekte Erkennung der Autos.



So ist sichergestellt, dass Bildpunkte immer dem richtigen Auto zugewiesen werden. Diese Berechnung ist sehr CPU aufwendig und verlangsamt das System daher gibt es Optimierungen, um die Last auf der CPU zu verringern.

2.5.1 Optimierung

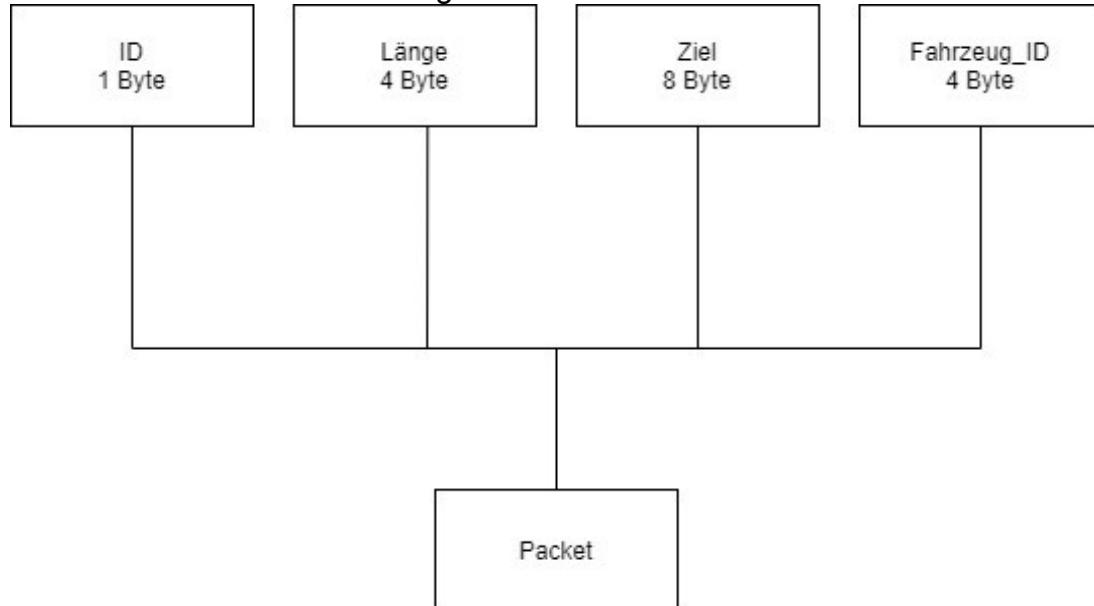
Um unnötigen CPU-Last zu vermeiden wird diese Berechnung nur jeden n-ten Frame verwendet. Für die dauerhafte Erkennung der Autos wird ein simpler Algorithmus verwendet. Dieser nimmt die Distanz zwischen zwei Blobs und vergleicht sie mit einer Hardcoded Value die festgelegt wurde. Diese Werte wurden aus den Live Tests entnommen und werden mit einer Toleranz von +- 20 Pixel bei einer Auflösung von (1440 * 1080 Pixel) implementiert. Durch diese Optimierung verliert man zwar an Genauigkeit, jedoch gewinnt man an Schnelligkeit. Es können dadurch mehrere Kamera Fenster geöffnet sein, ohne irgendwelche Hänger zu erkennen.

2.6 Auswertung der Bilddaten

Um die Bilddaten in für die Visualisierung brauchbare Daten umzuwandeln wir eine Klasse verwendet, die sich um dieses kümmert.

```
class GoalPacket : public Packet
{
private:
    float goal_x, goal_y;
    int vehicle_id;
```

Die Datenstruktur sieht wie folgt aus



- **ID** = ist die Identifikation welches Paket angekommen ist. In diesem Fall wird immer die ID = 5 verwendet. Da ein Positionspacket gesendet wird.
- **Länge** = Ist die Länge des gesamten Paktes.
- **Ziel** = sind die Koordinaten des Autos in float (x,y)
- **Fahrzeug_ID** = Ist die Identifikationsnummer des Autos von welchem die Koordinaten gesendet werden.

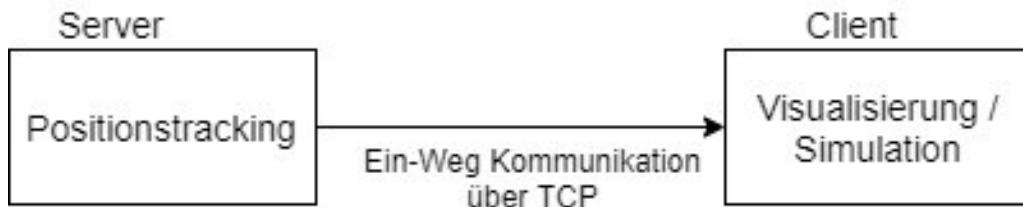
Diese Klasse ermöglicht es ganz einfach die Koordinaten in ein für die Visualisierung verständliches Paket umzuwandeln.

```
Schwarm::GoalPacket packet;
packet.set_goal(x,y);
packet.set_vehicle_id(1);
packet.allocate(packet.min_size());
packet.encode();
```

Mit diesem kleinen Musterprogramm sieht man, dass man ganz leicht die Pakete zusammenbauen kann. Diese werden dann mithilfe von `packet.rawdata()`; über die Socket Verbindung zu der Visualisierung gesendet.

2.7 Kommunikation mit Simulation / Visualisierung

Um die Kommunikation zwischen den beiden Modulen bereitzustellen wird eine von Clemens Pruggmayer geschriebene Library `cppsock` verwendet. Diese kümmert sich zum Großteil um das Error-Handling und threading.



Über diese TCP Verbindung werden die Pakete gesendet. Diese Kommunikation ist einseitig. Das Positionstracking-Modul sendet nur Daten an die Visualisierung diese sendet aber niemals etwas zurück.

Der Server wartet auf eine Verbindung auf Port 10001.

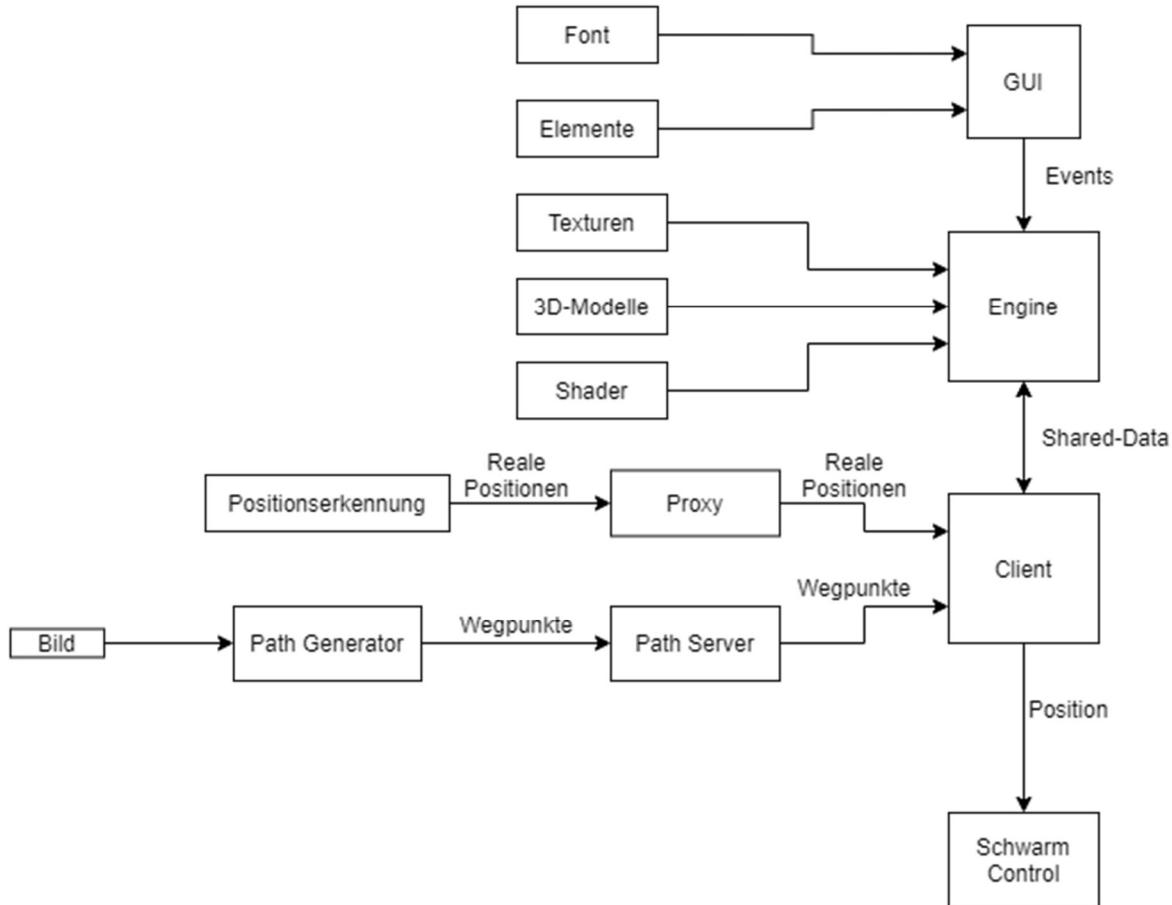
Die Pakete werden wie im Kapitel [2.5 Auswertung der Bilddaten](#) zusammengebaut, encodiert und gesendet.

Diese Pakete können auf der anderen Seite (Visualisierung / Simulation) mit dem **decode** Befehl ganz einfach lesbar gemacht werden.

3 Visualisierung und Simulation

3.1 Übersicht Softwarearchitektur

3.1.1 Blockschaltbild



3.1.2 GUI

Das GUI ist die graphische Oberfläche der Visualisierung. Es enthält Text und verschiedene Elemente wie zum Beispiel eine Eingabezeile. Die Schriftart, die verwendet wird, wird über ein File in das Programm eingelesen. Mehr zu diesem Thema gibt es im Kapitel 3.2 (Graphical User Interface).

3.1.3 Engine

Die Engine ist das Rückgrat der Software. Sie beinhaltet das graphische Rendering, steuert Bewegungen und den Ablauf der Simulation und reagiert auf Events. Dazu wird im Kapitel 3.3 (Aufbau der Engine) näher eingegangen.

3.1.4 Client

Der Client ist für das Empfangen und Senden von Daten über das Netzwerk zuständig. Da die Engine mit dem Client kommunizieren muss, wird ein Shared-memory verwendet. Auf diesen Speicher haben nur die Engine und der Client Zugriff.

3.1.5 Modelle, Texturen, Shader

Modelle beschreiben die Form der Objekte in einer Rendering-Szene. Sie beinhalten Raumpunkte, genannt Vertices, Normalvektoren und Texturkoordinaten.

Texturen werden verwendet, um Modellen Farben zu geben. Diese sind Bild-Files, welche in den Speicher geladen werden, und anschließend am Modell angezeigt werden. Welcher Bereich einer Textur auf einem Modell angezeigt wird, wird über die Texturkoordinaten gesteuert.

Shader sind Programme, die auf der Graphikkarte ausgeführt werden. Sie berechnen mithilfe der Vertices, Texturkoordinaten und Normalvektoren die endgültige Farbe eines Pixels auf dem Bildschirm. Dieser Prozess wird „Shading“ genannt. Genauer erläutert wird das im Kapitel 3.3 (Aufbau der Engine) und 3.4 (Erstellung von 3D-Modellen).

3.1.6 Path Generierung

Die Generierung eines Weges erfolgt aus einem Bild-File. Der Anwender kann mit einem Programm, womit man Bild-Files editieren kann, einen Beliebigen weg auf das Bild zeichnen. Dieses Bild wird anschließend vom „Path Generator“ eingelesen und in Wegpunkte unterteilt, welche die Fahrzeuge abfahren. Diese Wegpunkte und dessen Nummer erhält der „Path Server“. Er ist für die Aufbereitung und für das Senden der Wegpunkte zur Visualisierung zuständig.

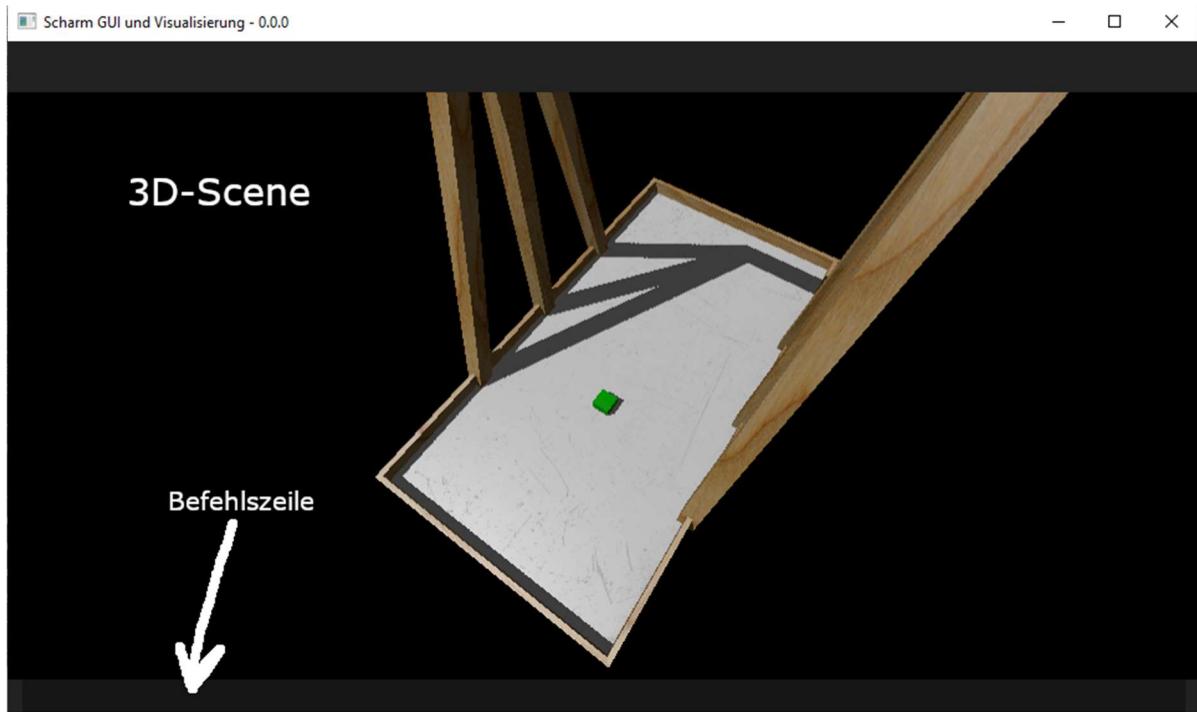
Wenn die realen Fahrzeuge angesteuert werden, werden neben den generierten Wegpunkten, noch zusätzliche Positionen von der Positionserkennung empfangen. Diese Positionen sind die Koordinaten der abgefilmten Fahrzeuge und werden verwendet, um die Bahnen der virtuellen Fahrzeuge in der Visualisierung zu korrigieren.

Da die Positionserkennung Daten in der falschen Byteorder sendet, ist ein Proxy zwischen der Positionserkennung und der Visualisierung, der die Byteorder umdreht und weitersendet.

Genauer wird im Kapitel 3.6 (Auswertung von erhaltenen Positionsdaten) auf dieses Thema eingegangen.

3.2 Graphical User Interface

3.2.1 Aufbau



3D-Szene: In der 3D-Szene wird zum Anzeigen der Visualisierung und Simulation verwendet. In ihr werden die Fahrzeuge und das Modell des Tisches gerendert.

Befehlszeile: Über die Befehlszeile werden die Fahrzeuge im realen Aufbau, sowie in der Simulation, gesteuert.

Anmerkung: Die Befehle zur Ansteuerung des realen Aufbaus wurden aus Zeitgründen noch nicht realisiert!

3.2.2 Befehle

Jeder Befehl, der in die Befehlszeile eingegeben wird, muss mit einem „./“ beginnen.

3.2.2.1 Befehle für die Simulation

Syntax: /simu generate <image-file> <vehicle-ID> <number of goals>

1. generate

Dieses Argument gibt der Simulation die Anweisung, dass ein neuer weg für ein Fahrzeug generiert werden soll.

2. <image-file>

Typ: string

Gibt an welche Bilddatei eingelesen wird. Dabei muss beachtet werden, dass ein Standardverzeichnis angegeben werden kann. Wenn ein Standardverzeichnis verwendet wird, muss der Dateipfad relativ zu dem Standardverzeichnis angegeben werden!

3. <vehicle-ID>

Typ: unsigned integer

Gibt an für welches Fahrzeugt der Weg generiert werden soll. Es ist zu beachten, dass jedes Fahrzeug eine numerische ID, beginnend bei 0 (Typ: unsigned integer), besitzt.

4. <number of goals>

Typ: unsigned integer

Gibt die Auflösung des Weges an. Der Weg wird in einzelne Punkte unterteilt, die vom Fahrzeug nacheinander abgefahren werden. Desto mehr Punkte (Goals) generiert werden, umso genauer wird der weg in der Simulation nachgefahren.

Nach Ausführung dieses Befehls, wird der eingegebenen Fahrzeugnummer, der generierte Weg zugeordnet.

Syntax: /simu start

Nach Ausführung dieses Befehls wird der Simulationsprozess eingeleitet. Der Simulationsprozess beendet sich von allein, wenn alle Fahrzeuge ihren Weg nachgefahren sind.

Syntax: /simu stop

Nach Ausführung dieses Befehls wird der Simulationsprozess unterbrochen.

Syntax: /simu reset

Dieser Befehl löscht bei allen Fahrzeugen den zugehörigen Weg.

Achtung: Dieser Befehl kann nur ausgeführt werden, wenn die Simulation beendet ist. Um die Simulation abzubrechen nutze: „/simu stop“.

3.2.2.2 Befehle für den realen Aufbau

Syntax: /real

Dieser Befehlt wechselt zwischen dem Simulationsmodus und der Ansteuerung des realen Aufbaus. Standardgemäß beim Start der Visualisierung ist der Simulationsmodus eingeschaltet. Wird einmal /real ausgeführt befindet sich die Visualisierung im realen Modus, also sie steuert den realen Aufbau an. Wird /real ein zweites Mal ausgeführt befindet sich die Visualisierung wieder im Simulationsmodus.

3.2.3 Realisierung

3.2.3.1 Erstellung und Initialisierung vom GUI

```
/* CREATE TEXTBOXES */
    std::cout << get_msg("INFO / GUI") << "Creating textboxes..." << std::endl;
    GUI::TextBox cmd_line(main_window_handler.get_width_ptr(), main_window_handler.get_height_ptr(), main_window_handler.get_aspect_ptr());
    cmd_line.set_pos(-0.975f, -1.0f);
    cmd_line.set_size(1.95f, 0.1f);
    cmd_line.set_textinput_color(Main::TEXTBOX_ACTIVE_COLOR);
    cmd_line.set_cursor_color(Main::TEXTBOX_CURSOR_COLOR);
    cmd_line.set_font_size(0.075f);
    cmd_line.set_text_color(Main::TEXTBOX_ACTIVE_TEXT_COLOR);
    std::cout << get_msg("INFO / GUI") << "Textboxes created..." << std::endl;
```

Hier wird die Kommandozeile für die Befehlseingebe erstellt. Es werden einige Parameter, wie Höhe, Breite, Schriftgröße und Textfarbe gesetzt.

```
/*
 * -----
 * INIT ELEMENT HANDLER
 * -----
 */
GUI::ElementHandler element_handler;
element_handler.attach_element(cmd_line);
element_handler.start(std::chrono::milliseconds(5));
std::cout << get_msg("INFO / GUI") << "Started Element-Handler." << std::endl;
```

Der Element-Handler steuert die ganzen Interaktionen der übergebenen GUI-Bausteine, wie Buttons, Textzeilen, usw. Darüber hinaus ruft er Events auf, sodass der Event-Handler darauf reagieren kann. Der Vorteil ist, dass der Event-Handler frei programmierbar ist und der Programmierer bestimmen kann, was passiert, wenn er mit den GUI-Bausteinen interagiert.

```
/*
 * -----
 * INIT ELEMENT RENDERER
 * -----
 */
GUI::ElementRenderer element_renderer(context, cmd_queue);
element_renderer.attach_handler(element_handler);
element_renderer.init();
element_renderer.start(std::chrono::milliseconds(5));
std::cout << get_msg("INFO / GUI") << "Started Element-Renderer." << std::endl;
std::cout << get_msg("INFO / GUI") << "Successfully initialized GUI." << std::endl;
```

Der Element-Renderer generiert die graphischen Daten, auch Meshes genannt, sodass man das Userinterface mit OpenGL rendern kann.

3.2.3.2 Erstellung der Events für das GUI

```
class TextInpListener : public Listener
{
private:
    TextBoxEnterEvent textboxenter;
    inline static Swarm::Client::SharedSimulationMemory* sharedsimumem;

protected:
    virtual void init(void)
    {
        this->register_event(textboxenter, reinterpret_cast<EventFunc>(on_textenter));
        this->set_interval(std::chrono::milliseconds(5));
    }

public:
    TextInpListener(void)
    {
        this->init();
    }

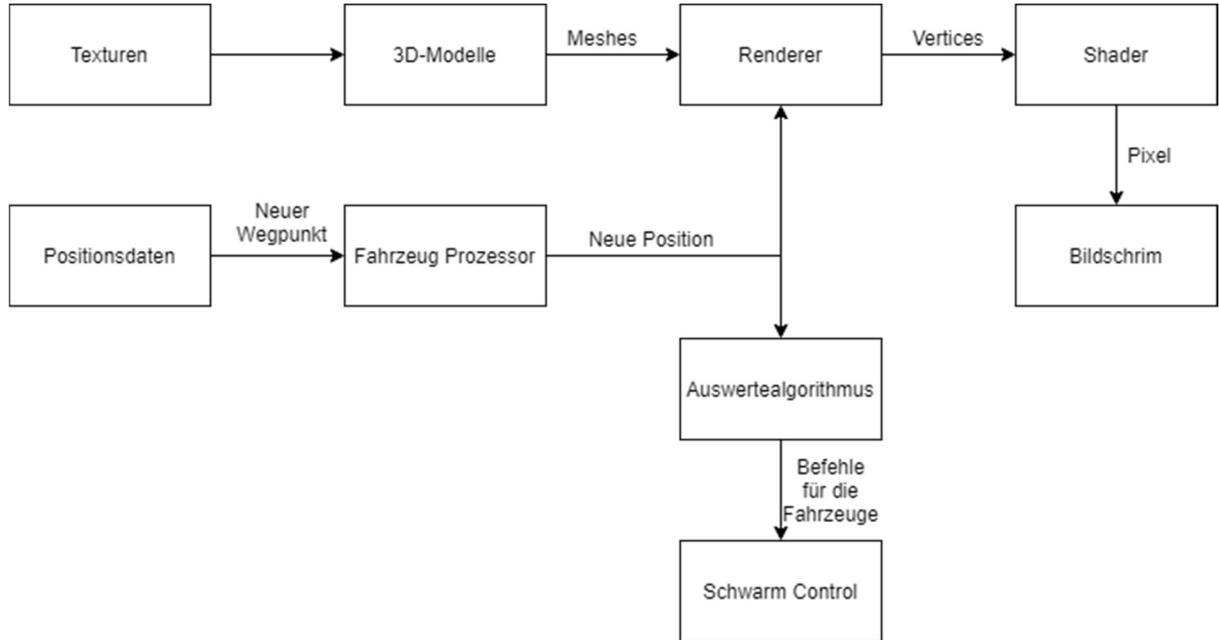
    static void set_sharedsimumem(Swarm::Client::SharedSimulationMemory* mem)
    {
        sharedsimumem = mem;
    }

    static void on_textenter(TextBoxEnterEvent& event)
    {
        std::vector<std::string> args;
        if(Swarm::decode_command(event.get_text_input().get_text_value(), args))
            Swarm::on_command(args, sharedsimumem);
        else
            std::cout << get_msg("ERROR") << "A command has to begin with '/\'. " << std::endl;
        event.get_text_input().set_text_value("");
    }
};
```

Die Methode „on_textenter(TextBoxEnterEvent& event)“ wird aufgerufen, nachdem ein Befehl mit ENTER bestätigt wurde. Hier wird der Befehl mit der Funktion „decode_command“ in seine einzelnen Argumente dekodiert. In der „on_command“ Funktion wird auf die im Punkt **3.2.2** genannten Befehle reagiert und die entsprechende Operation ausgeführt.

3.3 Aufbau der Engine

3.3.1 Blockschaltbild



3.3.2 Graphisches Rendering

Als Graphik API wird OpenGL verwendet.

3.3.2.1 Texturen, Modelle und Shader

```
gl::Model table;
gl::model_error_t model_error = table.load("Models/", "table.obj", gl::Model::MESH_COMBINED | gl::Model::INVERT_T);
gl::Model vehicle;
model_error = vehicle.load("Models/", "vehicle.obj", gl::Model::MESH_COMBINED | gl::Model::INVERT_T);

const gl::MeshMaterial table_mtl = table.get_materials().begin()->second; // there is only one material
const gl::MeshMaterial vehicle_mtl = vehicle.get_materials().begin()->second;

if(model_error != gl::model_error_t::MODEL_ERROR_NONE)
{
    return -1;
}
```

Hier wird das Modell des Tisches (table.obj) und der Fahrzeuge (vehicle.obj) geladen.

Das 3D-Modell ist in einer Object-Datei (.obj) abgespeichert und wird mit einer Library ausgelesen. Anschließend werden die Modelle zu den sogenannten Meshes konvertiert, um sie am Bildschirm anzeigen zu können.

```
void load_texture(unsigned int& texture, const std::string path, bool use_gamma)
{
    std::cout << get_msg("INFO / TEXTURE") << "Loading texture \""
        << path << "\"..." << std::endl;

    int x_pixels, y_pixels;
    uint8_t* data = stbi_load(path.c_str(), &x_pixels, &y_pixels, NULL, 4); // always load 4 components because the internal format requires RGBA

    if(data == NULL)
    {
        std::cout << get_msg("ERROR / TEXTURE") << "Unable to load texture \""
            << path << "\"." << std::endl;
        exit(-1);
    }
    else
    {
        glGenTextures(1, &texture);
        glBindTexture(GL_TEXTURE_2D, texture);
        glTexImage2D(GL_TEXTURE_2D, 0, (use_gamma) ? GL_SRGB_ALPHA : GL_RGBA, x_pixels, y_pixels, 0, GL_RGBA, GL_UNSIGNED_BYTE, data);
        glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_WRAP_S, GL_REPEAT);
        glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_WRAP_T, GL_REPEAT);
        glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MAG_FILTER, GL_LINEAR);
        glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MIN_FILTER, GL_LINEAR);
        glBindTexture(GL_TEXTURE_2D, 0);
        stbi_image_free(data);
        std::cout << get_msg("INFO / TEXTURE") << "Texture \""
            << path << "\" loaded." << std::endl;
    }
}
```

Hier werden die Texturen mit der Funktion „load_texture“ geladen. Texturen sind Bilddateien, die für die komplexe Färbung von Objekten verwendet werden. In diesem Codeabschnitt wird eine Bilddatei, mit der stb_master library, in den Speicher eingelesen. Anschließend wird ein neuer Texturpuffer mit „glGenTextures“ auf der Grafikkarte im VRAM erstellt. Zum Schluss wird das eingelesene Bild mit „glTexImage2D“ an die Grafikkarte gesendet und ist nun im VRAM gespeichert. Mit „glTexParameteri“ werden Filteroperationen für Texturen gesetzt. Diese Filteroperationen werden intern von der Grafikkarte berechnet.

```
void load_shader(gl::Shader& shader, const std::string& vertex_fname, const std::string& fragment_fname)
{
    static const std::string path("Shaders/");
    std::cout << get_msg("INFO / LOADING SHADER") << "Loading shader \""
and "\"" << path + fragment_fname << "\"..." << std::endl;
    gl::ShaderLoadError err = shader.load(path + vertex_fname, path + fragment_fname);
    if(err & gl::ShaderLoadErrorType::INVALID_FILE_PATH)
    {
        std::cout << get_msg("ERROR / LOADING SHADER") << "Unable to find path \""
me << "\" or \""
<< path + fragment_fname << "\"." << std::endl;
        exit(-1);
    }
    if(err & gl::ShaderLoadErrorType::SHADER_ALREADY_LOADED)
    {
        std::cout << get_msg("ERROR / LOADING SHADER") << "Shader has already been loaded." << std::endl;
        exit(-1);
    }
    if(err & gl::ShaderLoadErrorType::VERTEX_SHADER_ERROR)
    {
        std::cout << get_msg("ERROR / LOADING SHADER") << "Compile error occurred in vertex shader." <<
std::endl;
        std::cout << get_msg("INFO / VERTEX_SHADER") << shader.get_last_vertex_msg() << std::endl;
        exit(-1);
    }
    if(err & gl::ShaderLoadErrorType::FRAGMENT_SHADER_ERROR)
    {
        std::cout << get_msg("ERROR / LOADING SHADER") << "Compile error occurred in fragment shader." <<
std::endl;
        std::cout << get_msg("INFO / FRAGMENT_SHADER") << shader.get_last_fragment_msg() << std::endl;
        exit(-1);
    }
    if(err & gl::ShaderLoadErrorType::SHADER_LINK_ERROR)
    {
        std::cout << get_msg("ERROR / LOADING SHADER") << "Link error occurred." << std::endl;
        std::cout << get_msg("INFO / LINK_SHADER") << shader.get_last_link_msg() << std::endl;
        exit(-1);
    }
    std::cout << get_msg("INFO / LOADING SHADER") << "Shader \""
<< path + vertex_fname << "\" and \""
<< path + fragment_fname << "\" loaded." << std::endl;
}
```

Hier werden die Shader geladen, die für die Grafikszene benötigt werden. Es werden 2 Shader-Stufen verwendet: Vertex-Shader und Fragment-Shader. Der Vertex-Shader wird für jeden Vertex in der Grafikszene aufgerufen und bestimmt die endgültige Position der Vertices im Raum. Anschließend findet OpenGL intern heraus, welche Pixel das aktuelle Dreieck abdeckt und ruft für jeden Pixel den Fragment-Shader auf. Der Fragment-Shader besitzt die Aufgabe die Endgültige Farbe des Pixels zu errechnen. In ihm finden Lichtberechnungen, Schattenberechnungen, usw. statt. Der Vertex-Shader ist in einer .vert und der

Fragment-Shader in einer .frag abgespeichert und werden mit der Programmiersprache GLSL programmiert. In diesem Codeabschnitt wird ein Paar aus Vertex- und Fragment-Shader mit der Klassenmethode „shader.load“ geladen. Der Shader Code wird intern kompiliert und gelinkt. Anschließend erfolgen Abfragen, ob beim Kompilieren bzw. Linken Fehler aufgetreten sind.

3.3.2.2 Grafik Renderer mit OpenGL

Zunächst werden Puffer auf im VRAM für die Meshes (3D-Daten) erstellt.

Es gibt verschiedene Arten von Puffer, die verwendet werden:

- Vertex-Puffer: Das sind Puffer, wo die Raumpunkte, Vertices genannt, abgespeichert werden.
- Textur-Puffer: Das sind Puffer, wo die Pixel der Texturen, üblicherweise im 8-Bit-Gleitkommaformat, abgespeichert werden.
- Frame-Puffer: Der Frame-Puffer beinhaltet die gerenderten Farbinformationen der Pixel nach dem Aufruf der Shader.
- Render-Puffer: Dieser Puffer beinhaltet ebenfalls Informationen nach dem Rendern. Jedoch zum Unterschied zum Frame-Puffer beinhaltet der Render-Puffer Tiefen- und Stencil-Informationen. Diese Information wird verwendet, um zu erkennen, ob ein Objekt hinter einem anderen liegt.

Nachdem die Meshes in Puffer geladen wurden, wird das fertige Bild gerendert. Da nicht nur ein statisches Bild gerendert werden soll, muss das Rendering in einer Endlosschleife stattfinden. In dieser Renderschleife werden die Renderbefehle von der CPU an die Grafikkarte gesendet.

In dem folgenden Codeabschnitt findet das Rendering der gesamten 3D-Szene statt:

```
/* -----
 * VISUALIZATION SCENE
 * -----
 // set flags
 glEnable(GL_DEPTH_TEST);
 glEnable(GL_TEXTURE_2D);

 /* GENERATE (DRAW) SHADOW MAP(S) */
 glBindFramebuffer(GL_FRAMEBUFFER, fbo_worldlight_shadow);
 glViewport(0, 0, WORLDLIGHT_SHADOW_RESOLUTION, WORLDLIGHT_SHADOW_RESOLUTION);
 glClear(GL_DEPTH_BUFFER_BIT);

 simple_depth_shader.use();
 simple_depth_shader.uniform_matrix_4x4f("lightSpaceMat", 1, false, glm::value_ptr(light_space_transform));

 // draw table to depth map
 glBindVertexArray(vao_table);
 glDrawArrays(GL_QUADS, table_mtl.vertex_offset, table_mtl.vertex_count);
 glBindVertexArray(0);

 // draw vehicles to shadow map
 glBindVertexArray(vehicle_buffer.get_vao());
```

```
glDrawArraysInstanced(GL_QUADS, vehicle_mtl.vertex_offset, vehicle_mtl.vertex_count, vehicle_buffer.get_num_vehicles());
glBindVertexArray(0);

/* DRAW OBJECTS */
glBindFramebuffer(GL_FRAMEBUFFER, fbo_vis);
glViewport(0, 0, vis_window_handler.get_width(), vis_window_handler.get_height());
glClearColor(0.0f, 0.0f, 0.0f, 1.0f);
glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);

// set flags
 glEnable(GL_BLEND);
 glBlendFunc(GL_SRC_ALPHA, GL_ONE_MINUS_SRC_ALPHA);
 glEnable(GL_CULL_FACE);

// set light values for used shaders
vis_shader.use();
set_light_values(vis_shader);

// set view-projection matrix
vis_shader.uniform_matrix_4x4f("VP", 1, false, glm::value_ptr(glm::mat4(vis_projection * vis_view)));
vis_shader.uniform_matrix_4x4f("lightSpaceMat", 1, false, glm::value_ptr(light_space_transform));
);

/* DRAW TABLE */
draw_table(vis_shader, table_mtl, vao_table, table_diffuse_map, table_specular_map, shadow_maps
);

/* DRAW VEHICLES */
draw_vehicles(vis_shader, vehicle_mtl, vehicle_buffer.get_vao(), vehicle_buffer.get_num_vehicles(),
vehicle_diffuse_map, vehicle_specular_map, shadow_maps);

/* -----
 * SCENE (GUI + VISUALIZATION)
 *
 * Everything gets rendered to a seperate framebuffer.
 * Then the output is rendered as a single texture to a quad to be able to
 * make post-processing effects with the scene (e.g. gamma-correction).
 * -----
 */
glBindFramebuffer(GL_FRAMEBUFFER, fbo_scene);
glViewport(0, 0, main_window_handler.get_width(), main_window_handler.get_height());
glClearColor(Main::BACKGROUND_COLOR[0], Main::BACKGROUND_COLOR[1], Main::BACKGROUND_COLOR[2], Main::BACKGROUND_COLOR[3]);
glClear(GL_COLOR_BUFFER_BIT);

// set flags
 glDisable(GL_DEPTH_TEST);
 glEnable(GL_CULL_FACE);
```

```
/* DRAW GUI */
draw_GUI(element_shader, text_shader, element_renderer, element_font);

/* DRAW VISUALIZATION SCENE */
draw_fb(fb_scene_shader, vao_fb, tex_vis, 0);

/*
 * SCREEN (SCENE)
 */
glBindFramebuffer(GL_FRAMEBUFFER, 0);
glClearColor(0.0f, 0.0f, 0.0f, 0.0f);
glClear(GL_COLOR_BUFFER_BIT);

// set flags
glDisable(GL_BLEND);

/* DRAW SCENE */
draw_scene(scene_shader, vao_fb, tex_scene, 4);

// disable remaining active flags
glDisable(GL_TEXTURE_2D);
glUseProgram(0); // unuse shader

glfwSwapBuffers();
main_running = !glfwGetKey(GLFW_KEY_ESC) && glfwGetWindowParam(GLFW_OPENED);

glEndQuery(GL_TIME_ELAPSED);
int time;
glGetQueryObjectiv(timer_query, GL_QUERY_RESULT, &time);
time_results.push_back(time);
```

Zunächst wird eine Sogenannte “Shadow Map“ gerendert. Eine Shadow Map enthält nur Tiefeninformation und wird verwendet, um Schatten zu rendern. Dazu wird der gesamte Inhalt der 3D-Szene aus Sicht der Lichtquelle gerendert, um danach entscheiden zu können, ob ein Pixel im Schatten oder im Licht ist. Danach wird die eigentliche 3D-Szene mit Farbinhalt und allem was dazugehört gerendert. Der Renderinhalt wird in einen separaten Frame-Puffer gerendert, sodass zum Schluss noch Pixel-Post-Processing Operationen durchgeführt werden können. Zuletzt wird das GUI zur Szene dazugelerndert und es werden Post-Processing Operationen, wie Gamma-Korrektur durchgeführt.

3.3.3 Fahrzeug AI

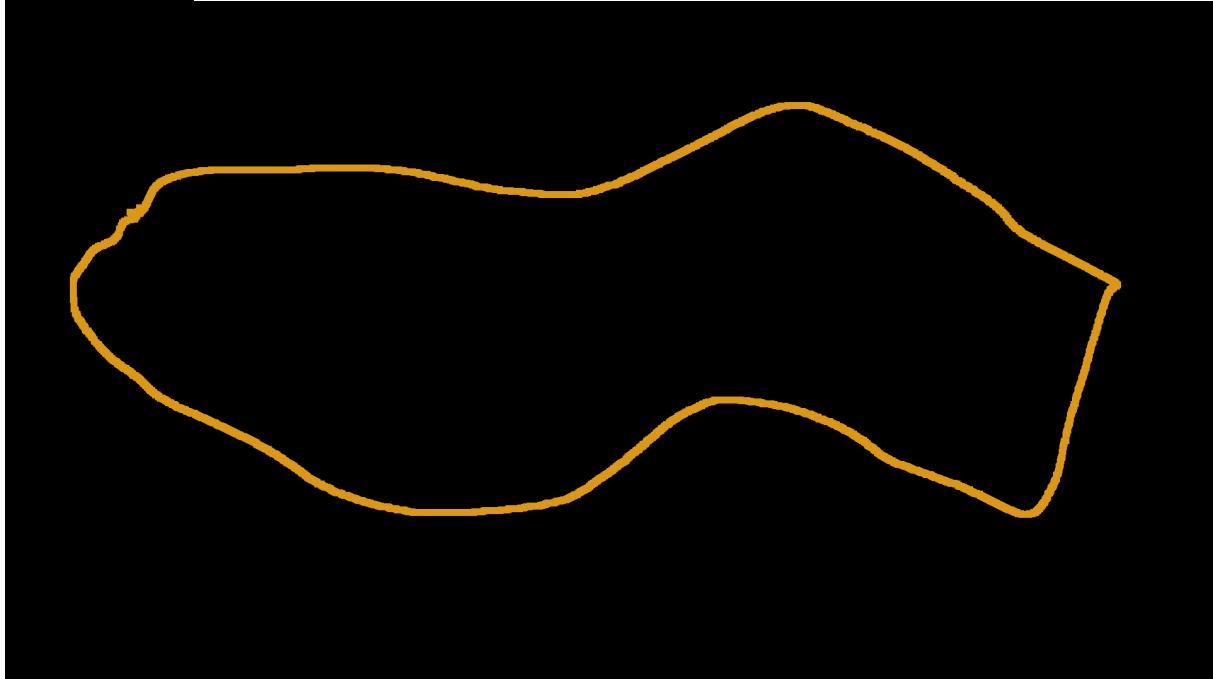
3.3.3.1 Generierung des Weges

Der Weg wird aus einer Bilddatei generiert.

Dieser Algorithmus ist in 3 Teile aufgeteilt:

1. Das Bild zu einem Graustufenbild zu konvertieren.
2. Die Außenlinie des Weges finden.
3. Die Außenlinie in eine bestimmte Anzahl an Punkten unterteilen.

Originalbild:



Konvertierung zu einem Graustufenbild:

```
void to_grayscale(uint8_t** data, image_info_t& ii)
{
    /* Grayscale: 1 color-channel that means a multiplication with the number of channels is useless.
     * |           \
     * |           \/ here
     */
    uint8_t* img_gray_data = new uint8_t[ii.width * ii.height]; // Allocating the new image buffer for
    // the grayscale image.
    image_info_t gray_ii{ii.width, ii.height, 1};                // And also declare a new image_info_t
    // struct for it.

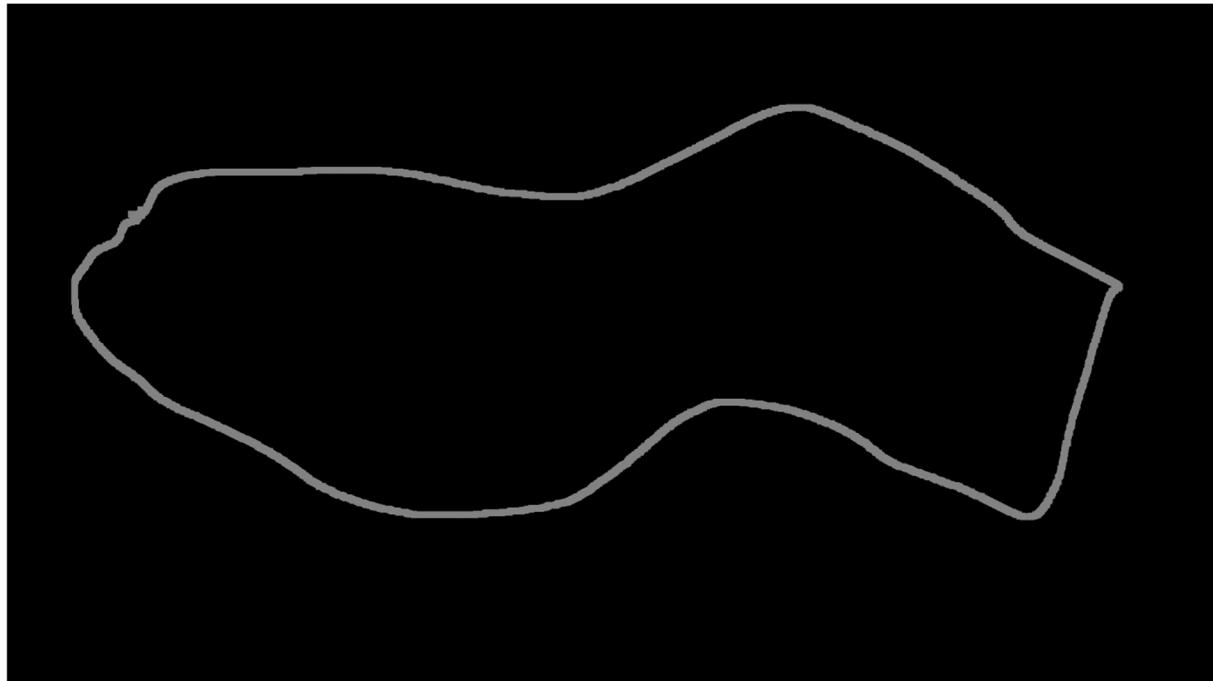
    // 2-dimensional loop that iterates every pixel of the image.
    int x, y; // a optimization for speed
    for(y = 0; y < ii.height; y++)
    {
        for(x = 0; x < ii.width; x++)
        {
            unsigned int sum = 0;
            // Calculate the average value of all color components except the alpha channel if the image has one.
        }
    }
}
```

```
// The average value of the 3 colors (RGB) corresponds to the grayscale value.  
for(int c = 0; c < ((ii.channels < 4) ? ii.channels : 3); c++)  
{  
    sum += (*data)[img_at(x, y, ii) + c];  
}  
// Write the grayscale value to the new generated buffer.  
img_gray_data[img_at(x, y, gray_ii)] = sum / ii.channels;  
}  
}  
ii = gray_ii;           // update the image information  
stbi_image_free(*data); // free the old data  
*data = img_gray_data; // let the data-pointer point to the new data (grayscale image data)  
}
```

Um ein RGB Bild zu einem Graustufenbild zu konvertieren wird folgende Formel verwendet:

$G = \frac{R+G+B}{3}$ also es wird der arithmetische Mittelwert aus den 3 Farbkomponenten gebildet. Enthält die Bilddatei jedoch noch eine 4. Komponente, zum Beispiel einen Alpha-Anteil, wird diese einfach weggelassen.

Das Bild sieht danach so aus:



Außenlinie des Wegs finden:

Um eine Außenlinie bestimmen zu können muss zunächst einmal klar sein, was ist überhaupt Weg und was nicht. Um das bestimmen zu können benötigt man einen Schwellenwert. Dieser Schwellenwert ist sehr niedrig angesetzt. Der Graustufenwert 0 bedeutet komplett schwarz und 255 bedeutet komplett weiß. Der Schwellenwert hat einen Wert von 10:

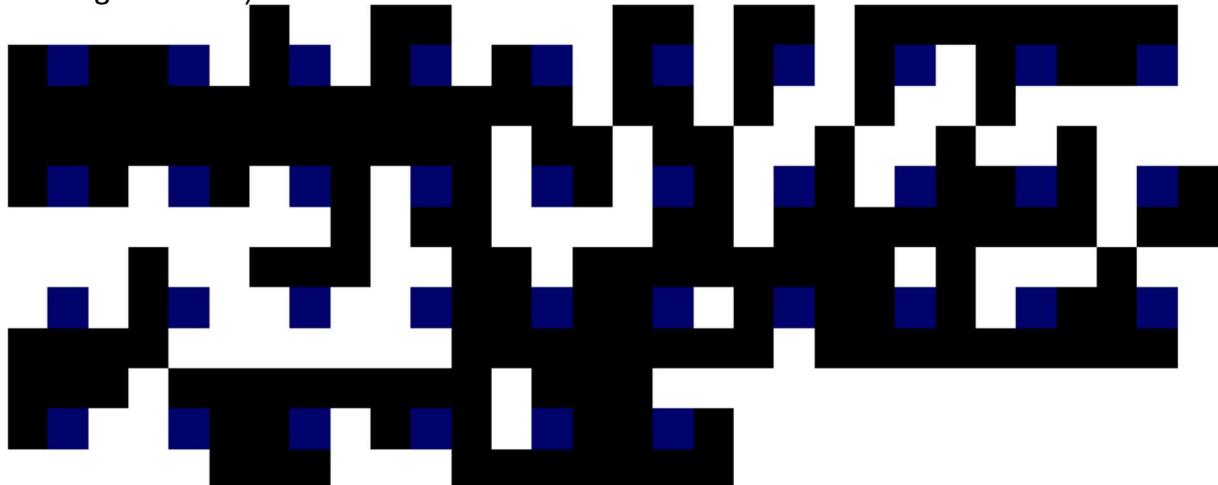
```
#define PATH_THRESHOLD 10
```

Das heißt, dass jeder Farbwert unter 10 als Hintergrund erkannt wird und alles größer oder gleich 10 als Weg erkannt wird.

Um die Außenlinie zu finden wird zunächst das Bild durchgegangen und nach einem Anfang gesucht. Nach dem der Anfang gefunden wurde, wird der Weg im Urzeigersinn nachgefahren und jeder Pixel vom Rand abgespeichert.

Um das nächste Pixel des Randes finden zu können, muss bekannt sein wie die Umgebung, des aktuellen Pixels aussieht. Dazu wird eine 3x3 Matrix, um das Aktuelle Pixel eingelesen und die Richtung bestimmt, in der sich das nächste Randpixel befindet.

Hier sieht man alle Möglichkeiten, die es gibt, um die Umgebung des aktuellen Pixels eindeutig zu bestimmen (das aktuelle Centerpixel ist blau markiert, schwarz sind die Umgebungspixel, die zum Weg dazugehören und weiß sind die Umgebungspixel, die Hintergrund sind):



Anmerkung: Um den Weg mit einer 3x3 Matrix eindeutig bestimmen zu können, muss der Weg mindestens 3 Pixel breit sein, sonst würden sich undefinierte Zustände ergeben und der Weg kann nicht generiert werden.

Hier ist der Code des beschriebenen Algorithmus, um die Außenlinie zu finden:

```
void gen_path(uint8_t* data, std::vector<img_coord_t>& path, const std::map<Path::PathMatrix*, Path::PathDirection*>& m2d, const image_info_t& ii)
{
    // If the image has no pixels, there is no need to generate a path.
    if(ii.width == 0 || ii.height == 0)
        return;

    int bx = 0, by = 0;                                // Not only a optimization for speed, the variables are
    also used later in the code!!!
    Path::PathMatrix begin_mat;                         // Matrix of the begin pixel.
    const Path::PathDirection* dir = nullptr;           // Direction of the matrix initialized to 'nullptr'.

    // Iterate through the pixels of the image.
    // The loop breaks if the image was iterated through if no path was found or, which is more likely,
    // the loop breaks at the first pixel that has a valid matrix.
    for(by = 0; by < ii.height && dir == nullptr; by++)
    {
        for(bx = 0; bx < ii.width && dir == nullptr; bx++)
        {
            begin_mat = gen_pathmatrix(data, bx, by, ii); // Generate matrix of current pixel.
            dir = get_dirp(m2d, begin_mat);                // Get the direction.
```

```
        }

    }

    // Subtract 1 because the loop iterates through one time too much.
    int x = --bx;
    int y = --by;

    // Current matrix.
    // Initialize it to the begin matrix because this is the starting point.
    Path::PathMatrix cur_mat = begin_mat;
    do
    {
        // Get the direction of the current matrix.
        dir = get_dirp(m2d, cur_mat);
        // Maybe the direction is invalid which means the pointer points to 'nullptr'.
        if(dir != nullptr)
        {
            path.push_back({x, y});      // Push the current x and y value into the vector because this
            is a valid path-coordinate.
            x += dir-
>direction_x();    // Add the direction to the x and y value which is returned by the direction_x() and
            direction_y() method.
            y += dir->direction_y();
        }
        cur_mat = gen_pathmatrix(data, x, y, ii);    // Generate the next matrix out of the new x and y
        value.
    }
    while(!(x == bx && y == by) && dir != nullptr); // Break if the begin matrix has been reached again
    or if the direction has become invalid.
}
```

Hier ist ein herangezoomter Ausschnitt, wo die Randlinie (weiß gekennzeichnet) gut zu sehen ist:



Außenlinie in einzelne Punkte (sog. Goals) unterteilen:

Die Anzahl der Pixel der Außenlinie wird durch die Anzahl der Punkte, die generiert werden sollen, unterteilt. Somit erhält man wie viele Randpixel von Punkt zu Punkt sind.

$$distance = \frac{n_{\text{außenlinie}}}{n_{\text{punkte}}}$$

Die resultierenden Punkte werden zum Schluss in einem File abgespeichert.

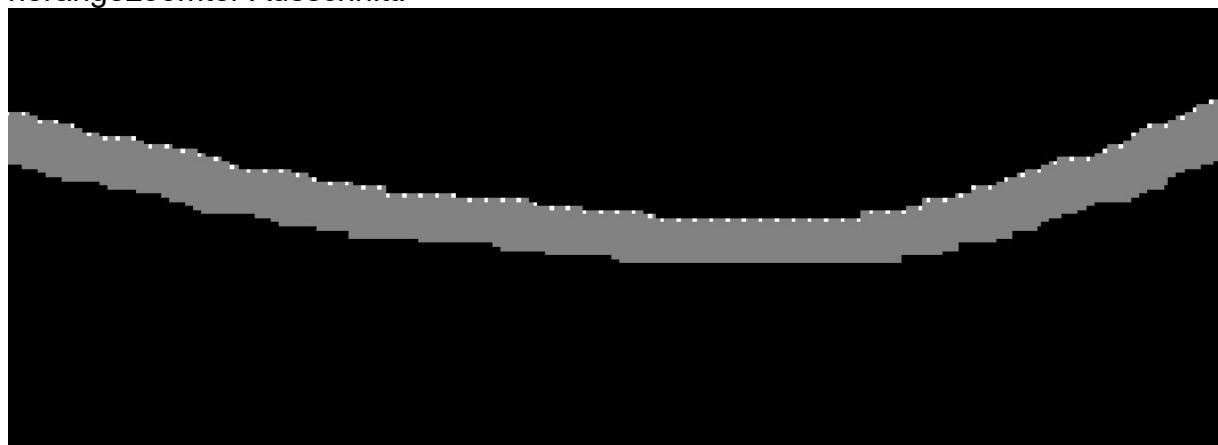
Dazu der Codeausschnitt, um die Randpixel zu unterteilen:

```
bool gen_goals(const std::vector<img_coord_t>& path, std::vector<goal_coord_t>& goals, unsigned int num_goals)
{
    // Invalid number of goals:
    // If the number of goals is to big or if 'path' does not contain any coordinates.
    if(num_goals > path.size() || path.size() == 0)
        return false;

    // Calculate the width between the goals.
    float g2g_width = (float)path.size() / (float)num_goals;
    // Split the path into a certain number of goals.
    for(float i = 0; i < path.size(); i += g2g_width)
    {
        goals.push_back(path.at((size_t)i));
    }
    // Because the vehicle should drive back to the begin where it started,
    // the first goal has to be pushed at the end of the goal-vector.
    // Only if the vector contains at least one element.
    if(goals.size() > 0)
        goals.push_back(goals.at(0));

    return true;
}
```

So sieht das in dem Endresultat im Beispielbild aus, auch hier wieder ein herangezoomter Ausschnitt:



Dieser Beispielweg wurde in 1000 Punkte unterteilt.

3.3.3.2 Aufbereitung der Goals

Jedes Fahrzeug besitzt eine Numerische ID beginnend bei 0. Die Goals werden von dem File, welches vom Pathgenerator erstellt wurde, gelesen und zu dem entsprechenden Fahrzeug zugeordnet. Wurde dem Fahrzeug bereits ein Weg zugeordnet wird der alte Überschrieben.

Dazu der Codeausschnitt:

```
/* If the path has successfully been generated, the goals can be
 *   read from the file.
 */
FILE* file = fopen("pathgenerator/goals.gol", "r"); // Open the file, the filename
e is defined.

if(file == NULL)
{
    /* If the file could not be opened the output file could not be opened
     * or the file has been deleted.
    */
    gettime(time);
    fprintf(shared_variables.logfile, "[%s] [ERROR] Could not find path to goal-
output file.\n", time);
    send_error(&server.get_socket(0), Swarm::packet_error::PACKET_FAILED_GENERATI
NG_PATH);
}
else
{
    /* If the file can be open, the reading process can begin. */
    // Clear the goal vector to write new goals into it.
    goals[shared_variables.pathgenpacket.get_vehicle_id()].clear();
    Goal goal;
    gettime(time);
    fprintf(shared_variables.logfile, "[%s] [INFO] Reading goals...\n", time);
    while(!feof(file)) // Read the whole file...
    {
        fscanf(file, "%f %f\n", &goal.x, &goal.y); // Read one line and pu
t the values into the "Goal" struct.
        goals[shared_variables.pathgenpacket.get_vehicle_id()].push_back(goal); //
Push the goal to the goal vector.
    }
    gettime(time);
    fprintf(shared_variables.logfile, "[%s] [INFO] Successfully generated goals for
vehicle %d.\n", time, shared_variables.pathgenpacket.get_vehicle_id());
    fclose(file); // Close the previously opened file.

    // Send acnoledge.
    Swarm::AcnPacket packet;
    packet.allocate(packet.min_size());
    packet.encode();
    server.get_socket(0).send(packet.rawdata(), packet.size(), 0);
}
```

3.3.3.3 Senden der Goals zur Visualisierung

Die ganze Kommunikation der Aufbereitung der Goals (Path-Server) mit der Visualisierung findet über TCP statt.

Zunächst teilt die Visualisierung mit dem im Punkt 3.2.2.1 genannten Befehl „/simu generate“ dem Path-Server mit, dass ein neuer Weg für eine im Befehl mitgegebene Fahrzeugnummer generiert werden soll. Die generierten Goals werden somit zum angegebenen Fahrzeug abgespeichert.

Danach kann die Visualisierung die einzelnen Goals empfangen. Dies geschieht über eine Anfrage an den Path-Server. Bei dieser Anfrage muss mitgegeben werden, für welche Fahrzeugnummer wird das Goal gesendet wird und das wievielte Goal es sein soll. Wird jedoch ein Goal angefragt, den es nicht gibt, wird ein Error an die Visualisierung zurückgesendet. Hingegen gibt es das Goal, wird ein Paket gesendet welches die Fahrzeug ID, die X-Pixelkoordinate und die Y-Pixelkoordinate beinhaltet. Dazu der Codeausschnitt:

```
void process_packet(cppsock::socket* socket, uint8_t* data, void** persistant)
{
    const uint8_t* id = Schwarm::Packet::id_ptr(data);           // Get pointer to packet id.
    const size_t* size = Schwarm::Packet::size_ptr(data);         // Get pointer to size of packet.
    SharedVariables* shared_variables = (SharedVariables*)*persistant; // Get pointer to shared memory.
    char time[48];

    gettimeofday(time);
    if(*id == Schwarm::ExitPacket::PACKET_ID)
    {
        /* If exit command was received set running value to 'false'.
        The server will shut down. */
        fprintf(shared_variables->logfile, "[%s] [INFO] Received stop command.\n", time);
        shared_variables->running = false;
    }
    else if(*id == Schwarm::PathGeneratePacket::PACKET_ID)
    {
        /* Only process the packet if the main thread is still running (highest priority)
        and only if the server is not already generating a path. */
        if(shared_variables->running && !shared_variables->generating_path)
        {
            shared_variables->pathgenpacket.allocate(*size); // Allocate memory for the packet.
            shared_variables->pathgenpacket.set(data);       // Set the data string.
            shared_variables->pathgenpacket.decode();        // Decode the packet.

            fprintf(shared_variables->logfile,
                    "[%s] [INFO] Generating goals for file %s with %u goals for vehicle %d...\n",
                    time,
                    shared_variables->pathgenpacket.get_filepath(),
                    shared_variables->pathgenpacket.get_num_goals(),
                    shared_variables->pathgenpacket.get_vehicle_id());
        }
        // Set values for shared memory.
```

```
shared_variables->packet_id = shared_variables->pathgenpacket.id();

/* NOTE: It is important that setting the packed id is the last operation to ensure a synchronization
 * between this and the main thread. The main thread will only start to process the packet if the
 * id has been set.
 */

}

else if(shared_variables->running && shared_variables->generating_path)
{
    // If the server is busy generating goals, send an error to the client.
    fprintf(shared_variables->logfile, "[%s] [INFO] Server is already generating goals.\n", time);
    send_error(socket, Schwarm::packet_error::PACKET_SERVER_BUSY);
}

else if(*id == Schwarm::GoalReqPacket::PACKET_ID)
{
    fprintf(shared_variables->logfile, "[%s] [INFO] Received goal request.\n", time);
    if(shared_variables->running && !shared_variables->generating_path)
    {
        shared_variables->goalreqpacket.allocate(*size); // Allocate memory for the packet.
        shared_variables->goalreqpacket.set(data); // Set the packet data.
        shared_variables->goalreqpacket.decode(); // Decode the packet.

        /* The sending mechanic takes place in the main thread because
         * the vector for the goals is located there.
        */
    }

    // Set values for shared memory.
    shared_variables->packet_id = shared_variables->goalreqpacket.id(); // The same thing with the id like before...
}

else if(shared_variables->running && shared_variables->generating_path)
{
    // If the server is busy generating goals, send an error to the client.
    fprintf(shared_variables->logfile, "[%s] [INFO] Can't send goal because server has not finished generating goals\n", time);
    send_error(socket, Schwarm::packet_error::PACKET_SERVER_BUSY);
}
}
```

3.3.3.4 Verarbeitung der Goals in der Visualisierung

Befindet sich die Visualisierung im Simulationsmodus werden nur Daten vom Path-Server empfangen, verarbeitet und am Bildschirm dargestellt.

Befindet sich die Visualisierung nicht im Simulationsmodus, also steuert sie den realen Aufbau an, werden zusätzlich noch die Positionen der realen Fahrzeuge, gesendet von der Positionserkennungs-Software, empfangen. Diese Positionen werden ebenfalls verarbeitet, nämlich es wird sich eine Abweichung daraus berechnet, die die aktuelle Position und Richtung der Fahrzeuge korrigiert. Darüber hinaus werden sich Befehle errechnet, wie „links fahren“, „rechts fahren“, etc., die verwendet werden, um die Fahrzeug-Software anzusteuern.

Genauer wird auf den Auswertealgorithmus im Punkt 3.6 eingegangen.

Eine genauere Erklärung, wie die Pakete aufgebaut sind, befinden sich im Punkt 2.5.

3.4 Erstellung von 3D-Modellen

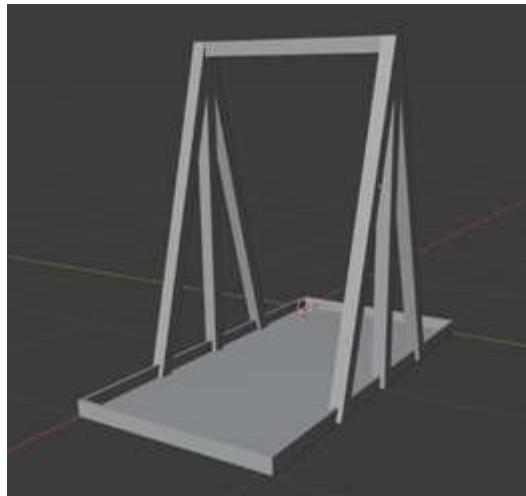
Die 3D-Modelle für die Visualisierung wurden mit Blender realisiert. Blender ist ein Programm, mit dem 3D-Modelle modelliert werden können. Diese Modelle können im Anschluss exportiert werden, damit das 3D-Modell in der Visualisierung implementiert werden kann. Darüber hinaus unterstützt Blender die meisten 3D-Modell-Formate.

Es wurden 2 Modelle mit Blender erstellt:

3.4.1.1 Tisch

Das Modell des Tisches ist eine 1:1 Modellierung vom realen Aufbau des Tisches.

Tisch als Modell in Blender:



Darüber hinaus werden realistische Texturen verwendet: eine Holztextur für die Halterung der Kamera und Wände des Tisches und eine weiße Fläche mit Reifenspuren für die Tischplatte.



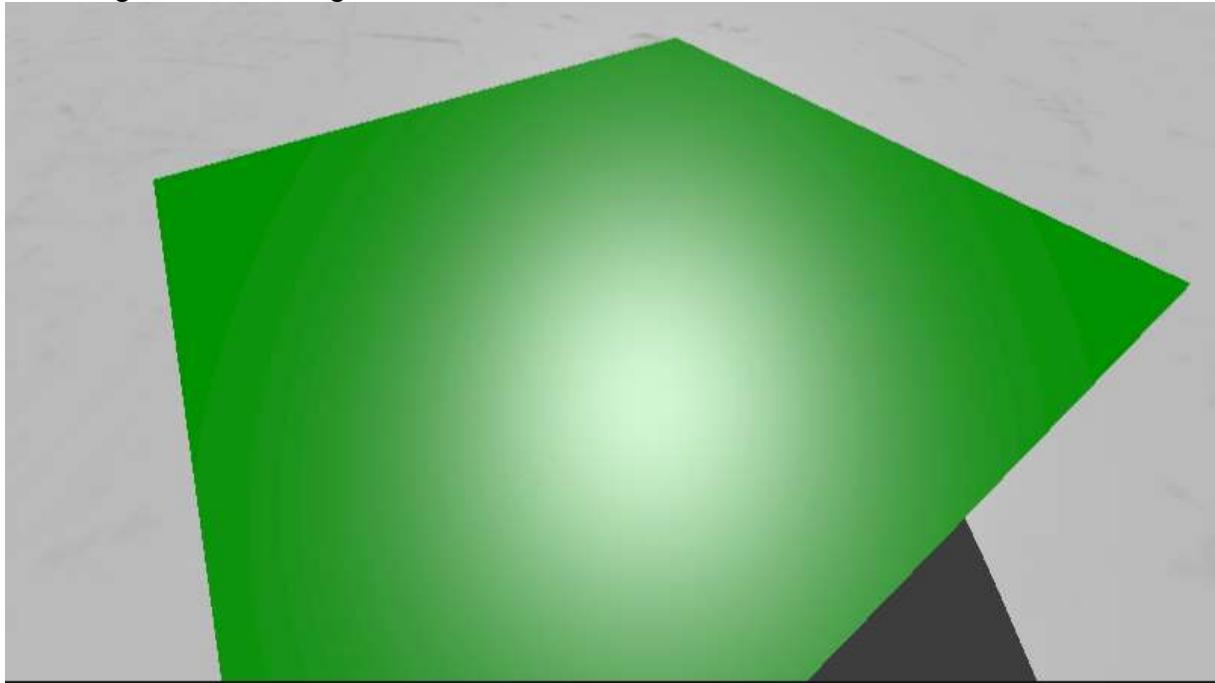
Hier kann man die Holztextur für die Halterung und Wände des Tisches sehen, ebenfalls ist die weiße Fläche mit Reifenspuren erkennbar. Die schwarzen Flächen des Bildes werden nicht verwendet. Anmerkung: Die schwarzen Flächen der Textur sind in Wirklichkeit transparent, sie wurden lediglich schwarz gemacht, damit es in der Dokumentation besser erkennbar ist.

Neben der farblichen Textur wird eine zweite Textur verwendet, mit der man den Lichteinfluss pro Pixel steuern kann, auch „Specular-Map“ genannt.



Der Rotanteil der so genannten Specular-Map skaliert wie viel Licht von der Oberfläche reflektiert wird. Der Grünanteil gibt an, wie klein bzw. groß der reflektierte Lichtpunkt auf der Oberfläche erscheint.

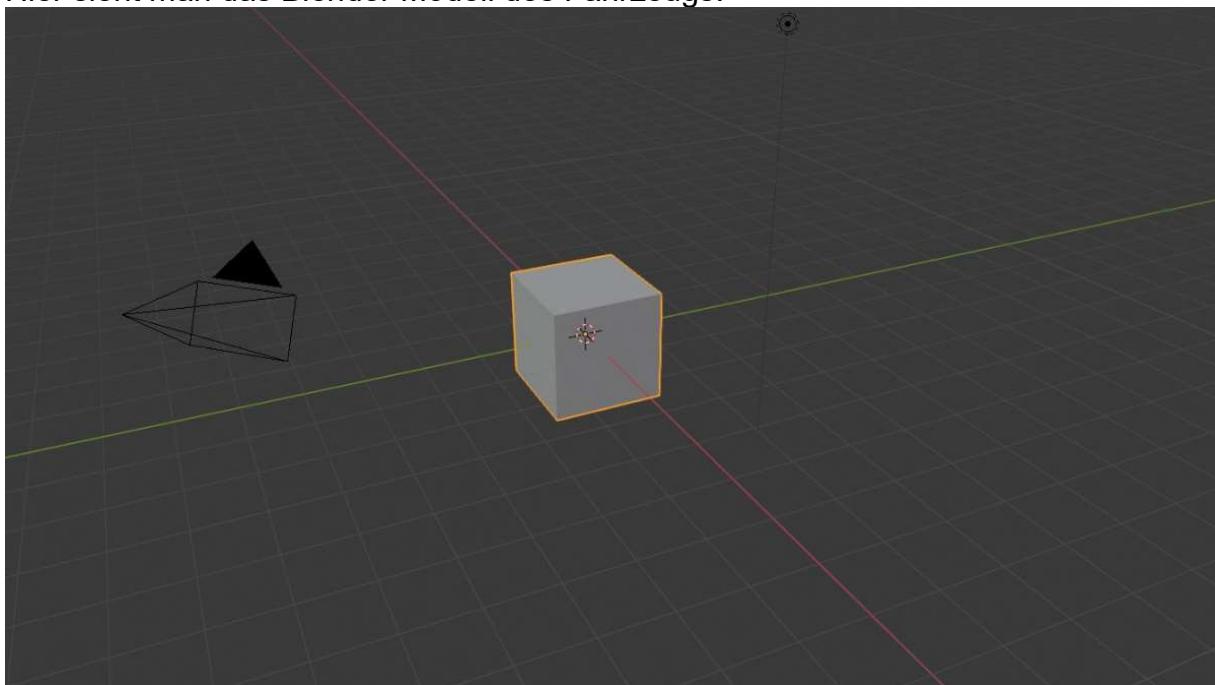
Der Lichtpunkt kommt daher, dass durch die Oberflächenrauigkeit das Licht in alle Richtungen, also nicht gerichtet, reflektiert wird:



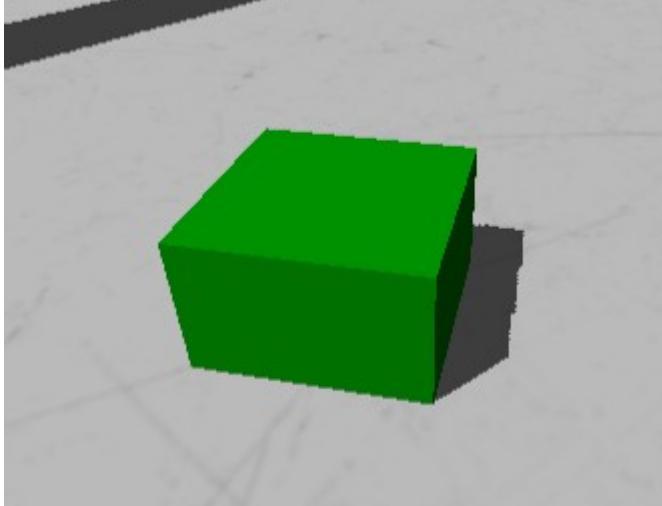
3.4.1.2 Fahrzeug

Da das Fahrzeug wurde als einfacher Würfel realisiert, da es auch in der Wirklichkeit fast wie ein Würfel aussieht. Dieser Würfel besitzt eine einfache grüne Farbe, damit er auf dem Tisch-Modell heraussticht.

Hier sieht man das Blender Modell des Fahrzeugs:

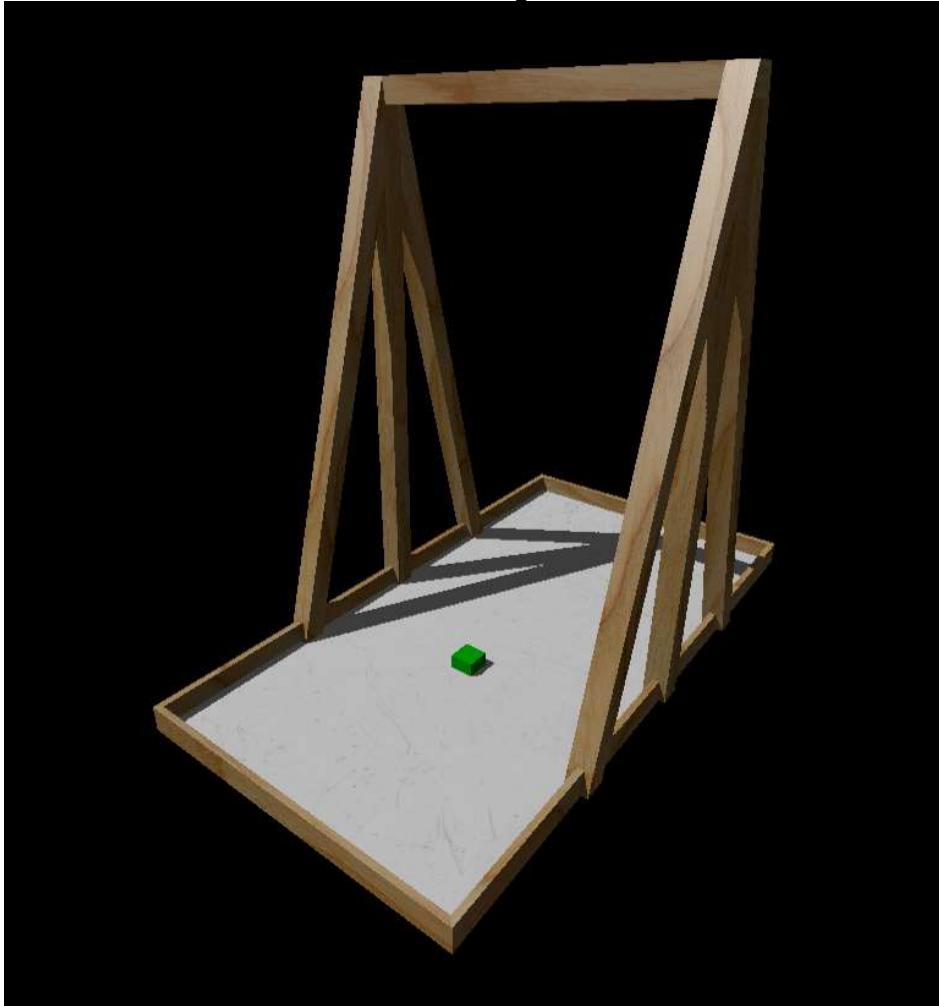


So sieht das Fahrzeug in der Visualisierung aus:



3.4.1.3 Komplette 3D-Szene

Das Endresultat mit allem was dazugehört, nämlich Modelle, Texturen und Shader:



3.5 Bewegungen im Dreidimensionalen Raum

Die Bewegung im 3D-Raum wurde nach Computerspielen nachempfunden.
Es ist möglich mit der Maus herumzuschauen und sich mit den Tasten W, A, S, D zu bewegen. Darüber hinaus kann man den Tisch aus Sicht der Kamera beobachten.

Tastenbelegung:

- W: Vorwärts
- A: Links
- S: Rückwärts
- D: Rechts
- Leertaste: Hinauf
- LSHIFT: Hinunter
- F1: Pause
- F5: Ansicht wechseln
- ESCAPE: Beenden

3.5.1.1 Kamerarotation

```
void mouse_action(int sx, int sy, double& rotx, double& roty, float sensetivity)
{
    constexpr double roty_max = 89.9/180.0*M_PI;
    const int sx_half = sx / 2;
    const int sy_half = sy / 2;

    int mx, my;
    glfwGetMousePos(&mx, &my);

    const int deltax = sx_half - mx;
    const int deltay = sy_half - my;

    rotx += deltax * sensetivity;
    roty += deltay * sensetivity;

    if(rotx >= 2*M_PI)
        rotx -= 2*M_PI;
    else if(rotx <= -2*M_PI)
        rotx += 2*M_PI;

    if(roty > roty_max)
        roty = roty_max;
    else if(roty < -roty_max)
        roty = -roty_max;

    glfwSetMousePos(sx_half, sy_half);
}
```

Der Mauscursor wird für jedes Bild, welches gerendert wird, eingelesen. Da der Mauscursor standardgemäß in der Mitte ist, kann die Abweichung von der aktuellen Mausposition zur Mitte berechnet werden.

$$\text{Abweichung} = \text{Mitte} - \text{Aktueller Position}$$

Der resultierende Rotationswinkel ergibt sich mit folgender Formel:

$$\text{Rotation} = \text{Abweichung} * \text{Sensitivity}$$

Somit hat die Sensitivity die Einheit Radian pro Pixel.

Darüber hinaus muss beachtet werden, dass die vertikale Rotation nicht über 90° bzw. unter -90° kommen darf!

Um die Kamera mit dem berechneten Winkel zu rotieren sind folgende Formeln notwendig:

$$X = pos_x + \sin(rotation_x) * \cos(rotation_y)$$

$$Y = pos_y + \sin(rotation_y)$$

$$Z = pos_z + \cos(rotation_x) * \cos(rotation_y)$$

3.5.1.2 Kamerabewegung

```
void move_action(double& posx, double& posy, double& posz, double rotx, float speed)
{
    static double old_time = glfwGetTime();
    double deltatime = glfwGetTime() - old_time;
    old_time = glfwGetTime();

    if(glfwGetKey('W') == GLFW_PRESS)
    {
        posx += speed * deltatime * sin(rotx);
        posz += speed * deltatime * cos(rotx);
    }
    if(glfwGetKey('A') == GLFW_PRESS)
    {
        posx += speed * deltatime * sin(rotx + M_PI / 2);
        posz += speed * deltatime * cos(rotx + M_PI / 2);
    }
    if(glfwGetKey('S') == GLFW_PRESS)
    {
        posx += speed * deltatime * sin(rotx + M_PI);
        posz += speed * deltatime * cos(rotx + M_PI);
    }
    if(glfwGetKey('D') == GLFW_PRESS)
    {
        posx += speed * deltatime * sin(rotx + M_PI * 1.5);
        posz += speed * deltatime * cos(rotx + M_PI * 1.5);
    }
    if(glfwGetKey(GLFW_KEY_SPACE) == GLFW_PRESS)
        posy += speed * deltatime;
    if(glfwGetKey(GLFW_KEY_LSHIFT) == GLFW_PRESS)
        posy -= speed * deltatime;
}
```

Als Input wird die alte XYZ-Position der Kamera verwendet und sich daraus die neue Position berechnet. Um eine konstante Bewegung bei instabiler Bildrate zu gewährleisten, wird die Renderzeit vom vorherigen Bild benötigt. Da mehrere Tasten

gleichzeitig gedrückt werden können müssen die einzelnen Richtungen vektoriell zusammenaddiert werden.

Daraus ergeben sich folgende Formeln für die Bewegung:

- Vorwärtsbewegung:

$$\Delta FW_x = v * \Delta t * \sin(rotation_x)$$

$$\Delta FW_y = v * \Delta t * \cos(rotation_y)$$

- Linksbewegung:

$$\Delta L_x = v * \Delta t * \sin(rotation_x + \frac{\pi}{2})$$

$$\Delta L_y = v * \Delta t * \cos(rotation_y + \frac{\pi}{2})$$

- Rückwärtsbewegung:

$$\Delta R_x = v * \Delta t * \sin(rotation_x + \pi)$$

$$\Delta R_y = v * \Delta t * \cos(rotation_y + \pi)$$

- Rechtsbewegung:

$$\Delta BW_x = v * \Delta t * \sin(rotation_x + \frac{3\pi}{2})$$

$$\Delta BW_y = v * \Delta t * \cos(rotation_y + \frac{3\pi}{2})$$

- Aufwärtsbewegung:

$$\Delta UP = v * \Delta t$$

- Abwärtsbewegung:

$$\Delta DOWN = -v * \Delta t$$

3.6 Auswertung von erhaltenen Positionsdaten

```
// Get the received normalized texture coordinates.  
    processor->sharedsimumem->sync.lock();  
    const float ntc_x = processor->sharedsimumem->goalpacket.get_goal_x();  
    const float ntc_y = processor->sharedsimumem->goalpacket.get_goal_y();  
    processor->sharedsimumem->sync.unlock();  
  
    // Calculate the real position from the normalized coordinates.  
    // Formula: origin + size * normalized_position  
    const float pos_x = processor->tableorigin_x + processor->tablesize_x * ntc_x;  
    const float pos_y = processor->tableorigin_y + processor->tablesize_y * ntc_y;  
  
    // Calculate move direction vector  
    glm::vec2 oldpos(cur_vehicle->pos_x(), cur_vehicle->pos_z());  
    glm::vec2 newpos(pos_x, pos_y);  
    cur_vehicle->set_distance(glm::distance(oldpos, newpos));  
    newpos = glm::normalize(newpos -  
    oldpos); // reuse "newpos" as new direction  
  
    // set direction vector  
    processor->sharedsimumem->sync.lock();  
    glm::vec2 olddir = cur_vehicle->direction(); // get old direction  
    cur_vehicle->direction() = newpos; // update old direction to new direction  
    processor->sharedsimumem->sync.unlock();  
  
    // set target of vehicle  
    cur_vehicle->set_next_goal(pos_x, pos_y);  
    cur_vehicle->set_goal_needed(false);  
  
    // calculate angle of new direction vector in (-180 to +180)  
    glm::vec2 baseline_vector(1.0f, 0.0f); // vector of the base line  
    float alpha = acosf(glm::dot(olddir, baseline_vector)) * ((olddir.y > 0)  
? -1.0f : 1.0f);  
    float beta = acosf(glm::dot(newpos, baseline_vector)) * ((newpos.y > 0)  
? -1.0f : 1.0f);  
  
    // only do the check if alpha and beta have a different sign  
    if((alpha < 0 && beta >= 0))  
    {  
        // convert alpha from (-180 to +180) to (0 to 360)  
        const float alpha360 = 2 * M_PI + alpha;  
        // compare if alpha or alpha360 is the shortest angle to beta, and set  
        alphas to the shortest  
        alpha = (VehicleProcessor::absf(beta -  
        alpha360) < VehicleProcessor::absf(beta - alpha)) ? alpha360 : alpha;  
    }
```

```
        else if(alpha >= 0 && beta < 0)
    {
        // convert beta from (-180 to +180) to (0 to 360)
        const float beta360 = 2 * M_PI + beta;
        // compare if alpha is the shortest angle to beta or beta360, and set b
eta to the shortest
        beta = (VehicleProcessor::absf(beta360 -
alpha) < VehicleProcessor::absf(beta - alpha)) ? beta360 : beta;
    }
    cur_vehicle->set_old_targetangle(alpha);
    cur_vehicle->set_new_targetangle(beta);
```

Hier wird nach dem Empfangen der Winkel berechnet, um wie viel Radian sich das Fahrzeug drehen muss, damit es in die Richtung des nächsten Punktes fährt. Nach dem sich der Winkel berechnet wurde wird daraus ein Richtungsvektor berechnet, in dessen Richtung sich anschließend das Fahrzeug bewegt. Darüber hinaus wird entschieden, ob sich das Fahrzeug nach Links oder Rechts bewegt. Diese Entscheidung geht darauf zurück, dass sich das Fahrzeug maximal um 180° drehen darf. Müsste es sich um 200° drehen, würde es dieselbe Endlage erreichen, wenn es sich um 160° in die andere Richtung dreht.

```
// calculate move distance
    // s = v * t
    float delta_s = cur_vehicle-
>get_speed() * (deltatime / 1000000000.0); // use deltatime in seconds
    glm::vec2 delta_pos = cur_vehicle-
>direction() * delta_s; // dpos = direction * ds

    // get distance to goal
    // Distance_vec = Goal - Position
    // Distance = sqrt(Distance_vec.x ^ 2 + Distance_vec.y ^ 2)
    float remain_dis = glm::distance(glm::vec2(cur_vehicle-
>get_next_goal_x(), cur_vehicle->get_next_goal_y()),
    glm::vec2(cur_vehicle->pos_x(), cur_vehicle-
>pos_z()));

    // check if translation would go over the goal
    // if not ... translate
    if(remain_dis > delta_s)
    {
        // move vehicle along the direction vector
        cur_vehicle->translate(delta_pos.x + cur_vehicle-
>pos_x(), 0.015f, delta_pos.y + cur_vehicle->pos_z());
        const float new_angle = lerp(cur_vehicle-
>get_new_targetangle(), cur_vehicle->get_old_targetangle(), remain_dis / cur_vehicle->get_distance());
        cur_vehicle->rotate(0.0f, new_angle, 0.0f);
    }
    else // if so set it to goal's position
    {
        cur_vehicle->translate(cur_vehicle->get_next_goal_x(), 0.015f, cur_vehicle-
>get_next_goal_y());
        cur_vehicle->rotate(0.0f, cur_vehicle->get_new_targetangle(), 0.0f);
        cur_vehicle->set_goal_needed(true);
    }

    cur_vehicle->calc();
    vehicle_buffer->update_vehicle(i);
```

Anschließend wird das Fahrzeug entlang des vorher berechneten Richtungsvektor verschoben, bis er das Goal erreicht hat. Nebenbei wird das Fahrzeug langsam um den berechneten Winkel rotiert, sodass es besser den Eindruck vermittelt, dass sich das Fahrzeug um eine Kurve bewegt und nicht entlang einer Linie.

Nebenbei werden die Daten aus der Positionserkennung empfangen. Es wird ein zweites Fahrzeug in die Visualisierung geladen, welches das „reale“ Fahrzeug darstellt. Die Daten aus der Positionserkennung werden verwendet, um das „reale“ Fahrzeug in der Visualisierung den realen Weg fahren zu lassen, währenddessen wird das Simulationsfahrzeug transparent und fährt die ideale Linie ab.

```
    float detec_pos_x;
    float detec_pos_y;

    // only calculate difference if visualization is in real-life mode
    if (real)
    {
        const float detec_ntc_x = (*processor->shared_memory)[Schwarm::Client::DETECTION_SERVER].detec_coords[(i / 2)].x;
        const float detec_ntc_y = (*processor->shared_memory)[Schwarm::Client::DETECTION_SERVER].detec_coords[(i / 2)].y;

        detec_pos_x = processor->tableorigin_x + processor->tablesize_x *
detec_ntc_x;
        detec_pos_y = processor->tableorigin_y + processor->tablesize_y *
detec_ntc_y;

        cur_vehicle_real->translate(detec_pos_x, 0.015f, detec_pos_y);
    }
```

In diesem Codeabschnitt werden die realen Positionen ausgelesen, die Positionen sind in normalisierten Koordinaten¹ gespeichert, und ausgewertet. Die Auswertung besteht aus der Umrechnung der normalisierten Koordinaten in die Koordinaten des Tisches. Dazu wird folgende Formel angewendet:

*Fahrzeugposition = Tischursprung + Tischgröße * normalisierte Koordinaten*

Zum Schluss wird das „reale“ Fahrzeug auf die resultierende Fahrzeugposition gesetzt.

¹Normalisierte Koordinaten bedeutet, dass die Bildkoordinaten im Intervall von 0 bis 1 abgespeichert werden und nicht in den üblichen Pixelkoordinaten. Der Vorteil dabei ist, sie können leicht in ein beliebiges Koordinatensystem umgerechnet werden, wie in diesem Fall in das Koordinatensystem des Tisches.

3.7 Übertragung zu Fahrzeug Software

Die Übertragung zum Schwarm Control geschieht über TCP. Es wird ein Packet generiert, welches den Winkel, um wie viel Radiant sich das Fahrzeug drehen muss, und eine Länge in Meter, wie weit das Fahrzeug noch fahren muss, bis es beim aktuellen Goal angekommen ist, beinhaltet. Darüber hinaus beinhaltet das Paket bzw. der Befehl ebenfalls eine numerische ID für das Fahrzeug, welches die Aktion ausführen soll.

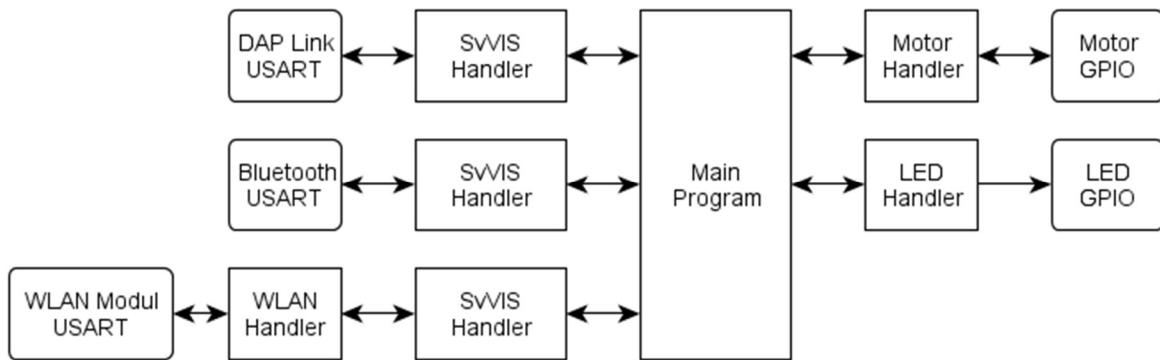
Ein positiver Winkel bedeutet, dass das Fahrzeug sich gegen den Uhrzeigersinn, also nach links, bewegt. Ein negativer Winkel bedeutet, dass das Fahrzeug sich im Uhrzeigersinn, also nach rechts, bewegt.

```
// encode and send vehicle command packet
Schwarm::VehicleCommandPacket command;
command.set_vehicle_id(i / 2);
command.set_angle(beta - alpha);
command.set_length(cur_vehicle->get_distance());
command.allocate(command.min_size());
command.encode();
(*processor->shared_memory)[Schwarm::Client::CONTROL_SERVER].client-
>send(command.rawdata(), command.size(), 0);
```

In diesem Codeabschnitt wird ein Packet erstellt, welches die drei oben genannten Parameter: Fahrzeug ID, Winkel und Länge enthält. Dieses Packet wird im Anschluss an die Fahrzeugsoftware gesendet.

4 Fahrzeug Software

4.1 Übersicht der Architektur



4.2 Kommunikation mit Swarm Controll

Ein Fahrzeug kann über 3 verschiedene Wege mit der Außenwelt kommunizieren:

- 1) Serielle Schnittstelle über DAP-Link Adapter
- 2) Serielle Schnittstelle über Bluetooth mittels HC06 Modul
- 3) TCP/IP Kommunikation mittels ESP8266 Modul

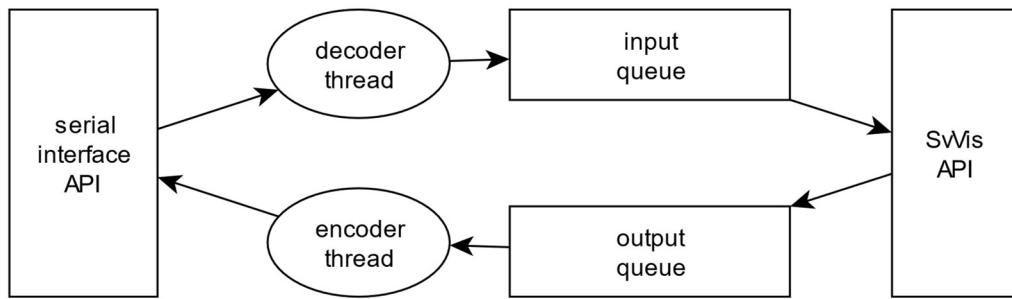
Da die Kommunikation über den DAP-Link Adapter nur für Entwicklungs- und Testzwecke verwendet werden kann, wird die Kommunikation im Endaufbau über Bluetooth oder WLAN stattfinden.

4.2.1 Kommunikationsprotokoll

Die Software des Fahrzeugs kommuniziert über das SvVis Protokoll mit der Swarm Control. Dieses Protokoll besteht aus einem Header mit der Größe von 1 Byte und darauffolgenden Nutzdaten. Die Länge der Nutzdaten ist vom Header und den Daten selbst abhängig.

Header Byte(s)	Bedeutung	Länge der Daten
0x0A 10	String Aq-Event(aq-on / aq-off)	Daten werden mit '\0' beendet. Implementierung ist auf eine String-Länge von 31 begrenzt
0x0B-0x13 11-19	16-bit integer (int16_t)	2 Bytes 16 Bit
0x15-0x1D 21-29	32-bit floating point number (float)	4 Bytes 32 Bit

4.2.1.1 Übersicht Software-Architektur der SvVis-API



Die serial-interface-API ist dafür zuständig, dass der Datenstrom für die Kommunikation abgearbeitet wird. Ob dieser Datenstrom intern gebuffert wird, ist durch das Interface nicht festgelegt. In der Implementation dieser Version der Software ist der Empfangsteil gebuffert, aber der Send-Teil ist nicht gebuffert. Der decoder-thread verarbeitet den Datenstrom in verwendbare Messages. Diese Messages werden dann in der input-queue gespeichert, in der sie vom Hauptprogramm jederzeit abgeholt werden können.

Das Hauptprogramm kann Messages senden, welche in der output-queue gespeichert werden. Der encoder-thread wandelt diese Messages in einen binären Datenstrom um, der durch das Serielle Interface an die Swarm Control weitergibt.

4.2.1.2 Decodierung

Die Pakete der Übertragung müssen vom Mikrocontroller in verwendbare Pakete dekodiert werden. Dazu müssen die binären Rohdaten aufgearbeitet werden.

```
msg.len = 0;
tar->interface->pop(recvbuf, osWaitForever);
msg.channel = recvbuf;
maxlen = ::SvVis::chid2len(msg.channel);
memset(&msg.data, 0, sizeof(msg.data));
```

Dieser Code-Ausschnitt initialisiert en Buffer für die Message und verarbeitet das erste Byte als Channel-Nummer. Danach muss zwischen string-messages und nicht-string messages unterschieden werden. Während die Länge bei nicht-string-messages durch den 1-Byte header vorgegeben ist, endet eine Strin-Message mit dem '\0'-Zeichen.

```
// handle non-string messages
while (msg.len < maxlen)
{
    tar->interface->pop(recvbuf, osWaitForever);
    if(msg.len < ::SvVis::data_max_len) {msg.data.raw[msg.len++] = recvbuf;}
}
osMessageQueuePut(tar->queue_recv, &msg, 0, osWaitForever);
```

Dieser Code Empfängt die Anzahl an Bytes, die für die Daten benötigt werden.

```
// handle string message
while (recvbuf != '\0')
{
    tar->interface->pop(recvbuf, osWaitForever);
    if(msg.len < ::SvVis::data_max_len1) {msg.data.raw[msg.len++] = recvbuf;}
}
msg.data.raw[::SvVis::data_max_len-1] = '\0';
```

Dieser Code empfängt Daten, bis der String mit dem '\0'-Zeichen beendet wird. Außerdem sorgt die Letzte Zeile dafür, dass das Ende immer mit einem '\0'-Zeichen beendet ist.

```
if(msg.data.i16 == 0)
{
    // aq off
    osEventFlagsClear(tar->event_flags, ::SvVis::flags::aq_on);
    osMessageQueueReset(tar->queue_send); // clear message queue
}
else if(msg.data.i16 == 1)
{
    // aq on
    osEventFlagsSet(tar->event_flags, ::SvVis::flags::aq_on);
}
else
{
    // string message
    osMessageQueuePut(tar->queue_recv, &msg, 0, osWaitForever);
}
```

Dieser Code handelt die Aq-on und Aq-off Events. Die Kontrolle ist über die von RTOS zur Verfügung gestellten EventFlags gelöst. Zusätzlich wird bei einem Aq-off-Event die sende Queue geleert.

Der Aufbau von Aq-Events ist von der SvVis-Software vorgegeben.

4.2.1.3 Encodierung

Das Encodieren von Messages ist weniger Aufwand als das Decodieren.

```
__NO_RETURN void SvVis::SvVis::func_send(void *this_void)
{
    ::SvVis::SvVis *tar = (::SvVis::SvVis*)this_void;
    ::SvVis::message_t msg;
    for(;;)
    {
        osMessageQueueGet(tar->queue_send, &msg, nullptr, osWaitForever);
        osEventFlagsWait(tar->event_flags, ::SvVis::flags::aq_on, osFlagsWaitAny | osFlagsNoClear, osWaitForever);
        tar->interface->put(msg.channel, osWaitForever);
        tar->interface->put_blocking(msg.data.raw, msg.len);
        if(osMessageQueueGetCount(tar->queue_send) == 0)
            { osEventFlagsSet(tar->event_flags, ::SvVis::flags::done_sending); }
    }
}
```

Dieser Thread verarbeitet Daten aus der sende-queue und sendet diese, sofern die Aq es erlaubt.

4.3 Hardware Ansteuerung

4.3.1 LED-Ansteuerung

Die Ansteuerung der LEDs ist die simpelste Hardware-Ansteuerung des Fahrzeugs. Es gibt eine Led, die periodisch Blinkt, um eine visuelle Bestätigung zu geben, dass die Software des Autos aktiv ist. Außerdem kann die Software die 3 Positionserkennungs-leds einzeln ein- und ausschalten.

4.3.2 Kommunikationsmodule

4.3.2.1 Allgemein

Der Decoder greift über eine abstrakte Interface-Klasse auf die Daten zu. Diese abstrakte Klasse wird verwendet, um eine bessere Flexibilität zu gewährleisten. Durch diese Klasse wird der Programmcode wesentlich besser lesbar.

```
class interface
{
public:
    virtual osStatus_t pop(uint8_t &data, uint32_t timeout) = 0;
    virtual osStatus_t put(uint8_t data, uint32_t timeout) = 0;
    virtual osStatus_t flush(void) = 0;
    virtual void      put_blocking(const void *data, size_t len) = 0;
};
```

Durch pointer oder Referenzen auf diese Klasse können Methoden einer abgeerbten Klasse aufgerufen werden. Dadurch ist es möglich, mit einer Implementierung der Decodierung / Encodierung mehrere verschiedene anzusprechende Schnittstellen zu verwenden. (z.B. USART direkt / AT-Kommandos über USART)

4.3.2.2 DAP Link

Die serielle Schnittstelle über den DAP-Link ist hauptsächlich für die Softwareentwicklung verwendbar, da ein USB-Kabel für die Kommunikation benötigt wird. Die Kommunikation erfolgt über eine direkte USART-Verbindung.

4.3.2.3 Bluetooth

Das verwendete HC06 Modul erlaubt kabellose Datenübertragung mit der Fahrzeugsoftware. Allerdings ist die Reichweite dieser Verbindung nicht sehr groß. Die Kommunikation erfolgt über eine direkte USART-Verbindung

4.3.2.4 WLAN

Das ESP8266 Modul ermöglicht die Kommunikation über eine TCP-Verbindung, die die TCP-Pakete über WLAN sendet und empfängt.

Allerdings müssen dazu einige Details über die zu verwendende Verbindung in die Konfigurationsdatei des Programms Eingetragen werden:

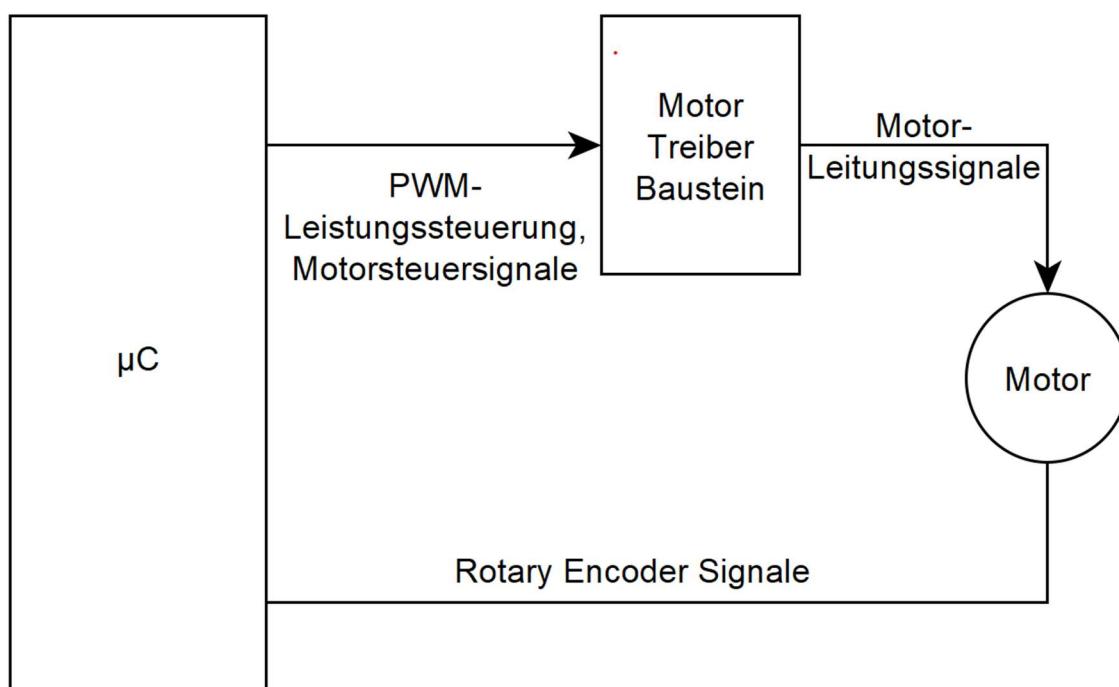
- SSID des WLAN Access Points
- Password für das WLAN-Netzwerk
- IP Adresse der Swarm Control
- TCP Port der Swarm Control

Die Kommunikation erfolgt über AT-Kommandos über eine USART-Schnittstelle. Da das WLAN Modul die AT-Kommandos benötigt, wurde die abstrakte interface-klasse verwendet.

4.3.3 Motor Ansteuerung

Die Ansteuerung der Motoren ist einer der wichtigsten Komponenten der Auto-Software, da dieser Teil dafür verantwortlich ist, dass sich das Auto bewegt.

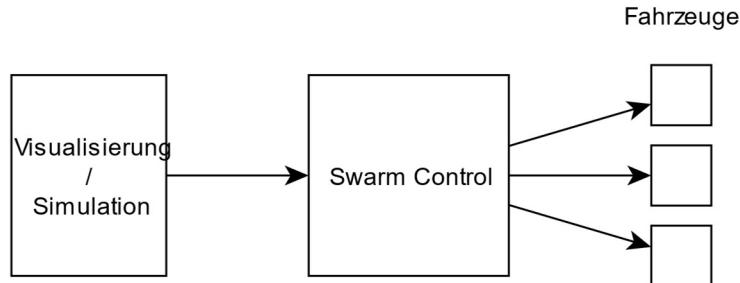
Die Motoren werden mithilfe eines Motortreiberbausteins angetrieben. Dieser Motortreiberbaustein bietet die Möglichkeit, die Motorleistung mit einem PWM Signal zu regeln. Der Verwendete Mikrocontroller (STM32F107RB) bietet eine Möglichkeit, ein PWM-Signal mit einstellbaren duty-cycle über die Integrierten Timer-Peripherie Komponenten generieren zu lassen. Die Motoren haben zusätzlich Inkrementalgeber, um die derzeitige Drehzahl des Motors auslesen zu können.



Eine Verbesserung dieses Designs wäre ein Hardwaremäßiger Rotary Decoder Baustein, der die Decoder Signale auswertet und dadurch Arbeit des Microcontrollers abnimmt.

Eine weiter Möglichkeit wäre Schrittmotoren zu verwenden, da Schrittmotoren keine Drehzahlregelung mit Rückmeldung benötigen. Außerdem kann mit Schrittmotoren eine wesentlich genauere Positionierung des Fahrzeugs erreicht werden.

5 Swarm Control



Die Swarm Control ist für die Ansteuerung der Fahrzeuge verantwortlich. Die Kommunikation mit der Visualisierung erfolgt über Schwarm Pakete, Die Kommunikation mit den Fahrzeugen Erfolgt über das SvVis Protokoll. An diesem Punkt kann mit der SvVis – Applikation die Kommunikation mitgehört werden. Die Swarm Control ist dafür zuständig, dass die Visualisierung eine unkomplizierte Ansteuerung der Fahrzeuge zur Verfügung hat, aber auch um eine Abstraktionsebene der Steuerbefehle aufzubauen.

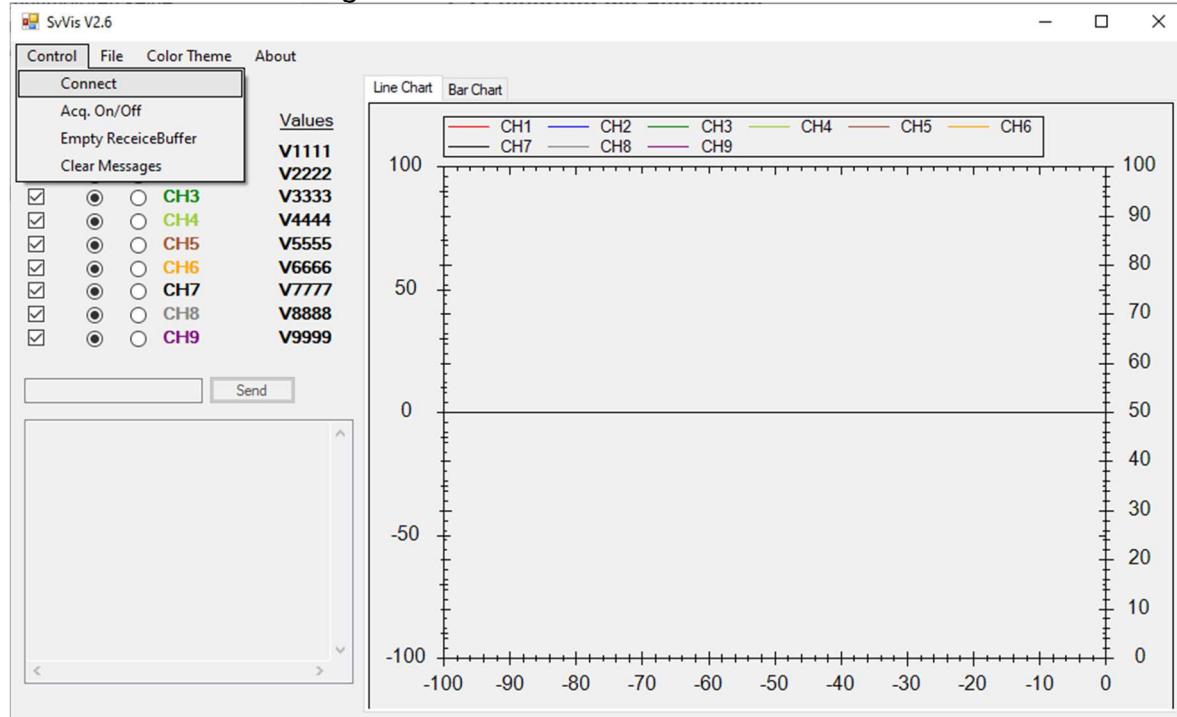
6 SvVis – Visualisierung - crashkurs

Das SvVis-Übertragungsprotokoll kann mithilfe einer windows-Applikation visualisiert werden.

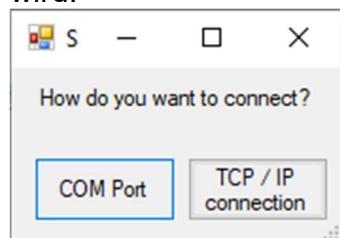
6.1 Verbindung aufbauen

6.1.1 COM-Port

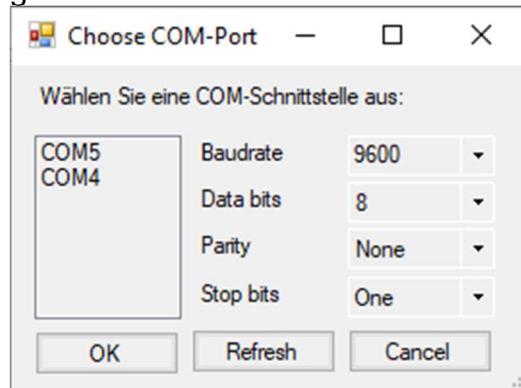
Um eine Verbindung zu einem COM-Port aufzubauen, wird als ersten unter „Control“ der Punkt „Connect“ angeklickt.



Danach öffnet sich ein weiteres Fenster, in dem auf den Button „COM Port“ gedrückt wird.

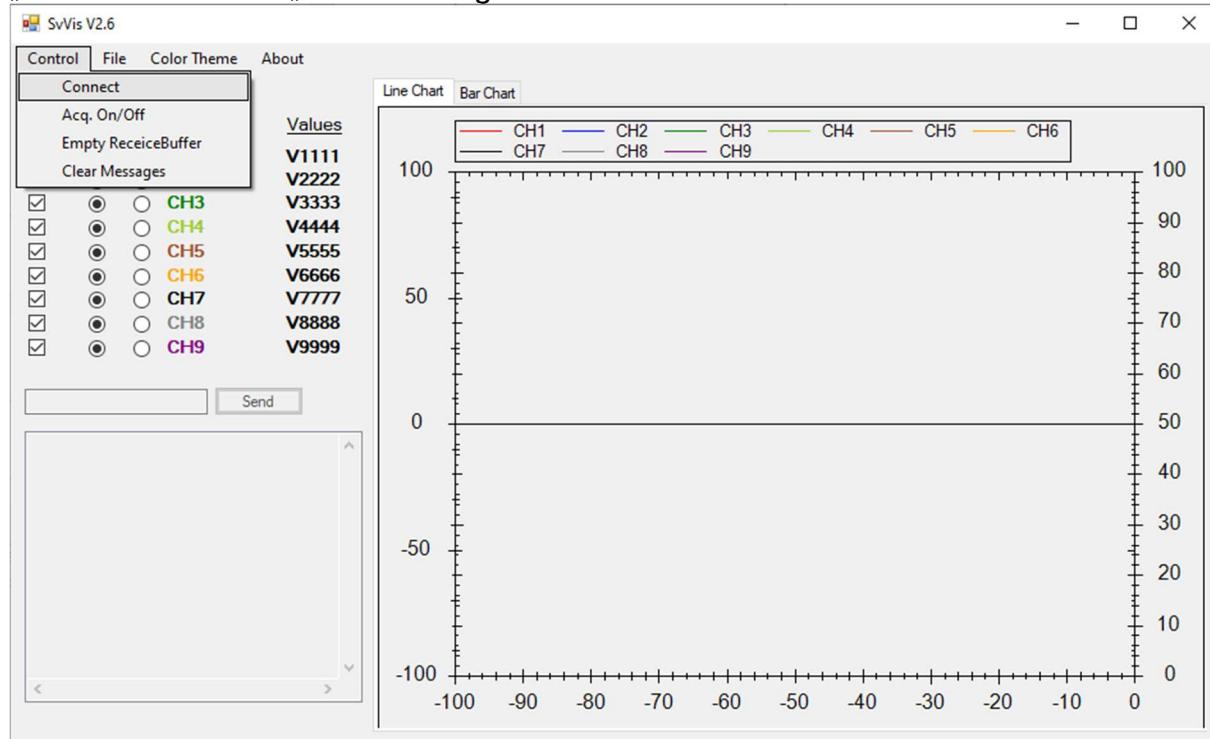


Anschließend wird ein COM-Port und eine Baudrate ausgewählt und auf OK gedrückt.

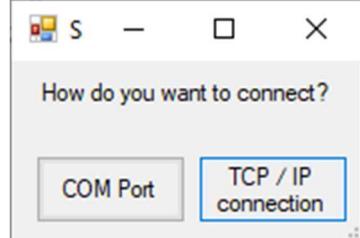


6.1.2 TCP/IP – Verbindung

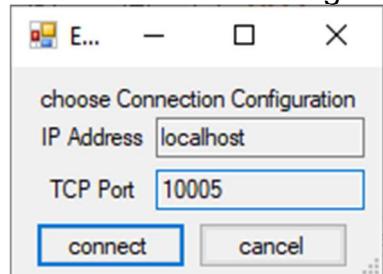
Um eine Verbindung über TCP/IP zu einem Server aufzubauen, wird als ersten unter „Control“ der Punkt „Connect“ angeklickt.



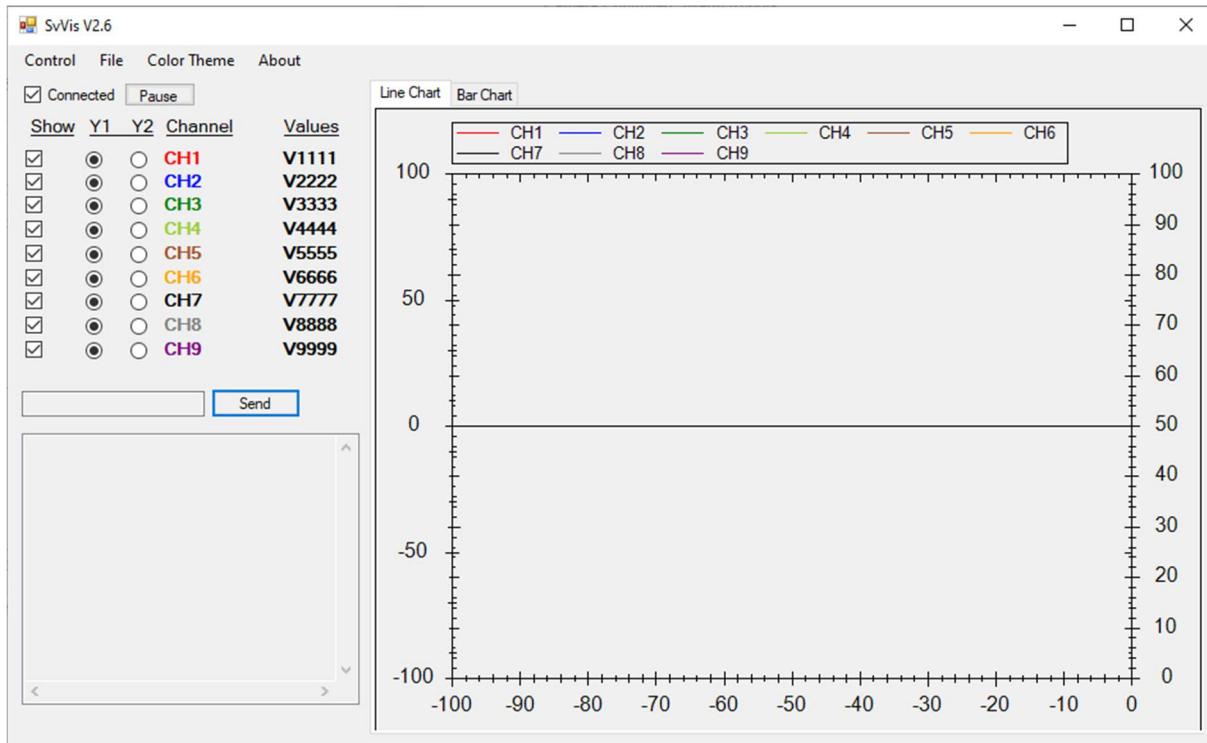
Danach öffnet sich ein weiteres Fenster, in dem der Button „TCP/IP connection“ angeklickt wird.



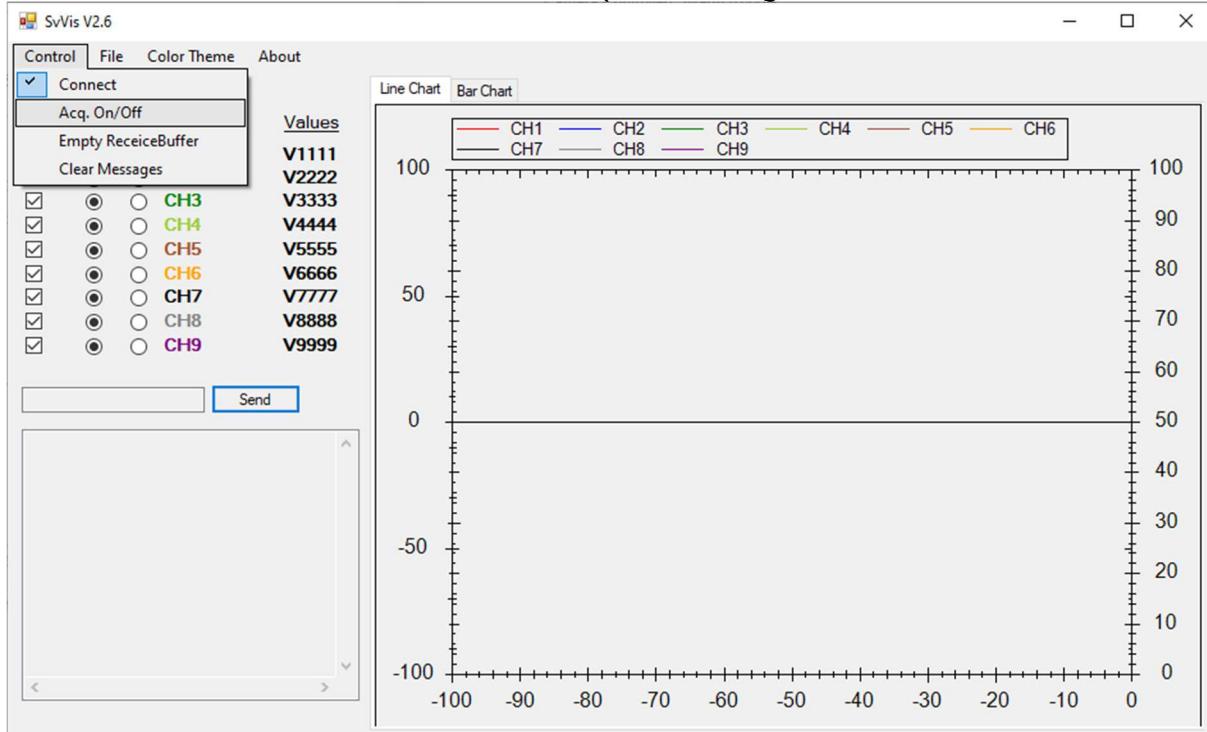
Anschließend wird IP-Adresse oder Hostname und TCP Port des Servers angegeben und auf connect gedrückt. In diesem Beispiel wird auf den eigenen Rechner auf Port 10005 eine Verbindung aufgebaut.



6.2 Daten senden / Empfangen

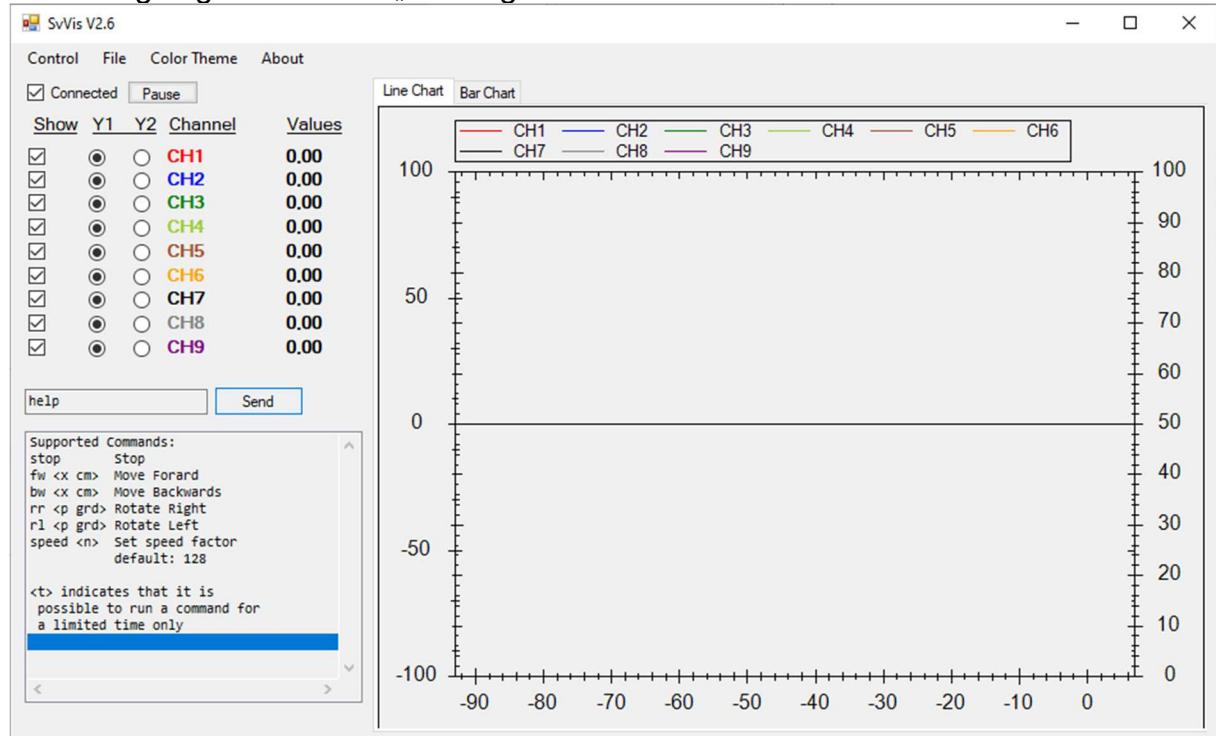


Um Daten zu senden, muss zuerst die acquisition eingeschaltet werden.

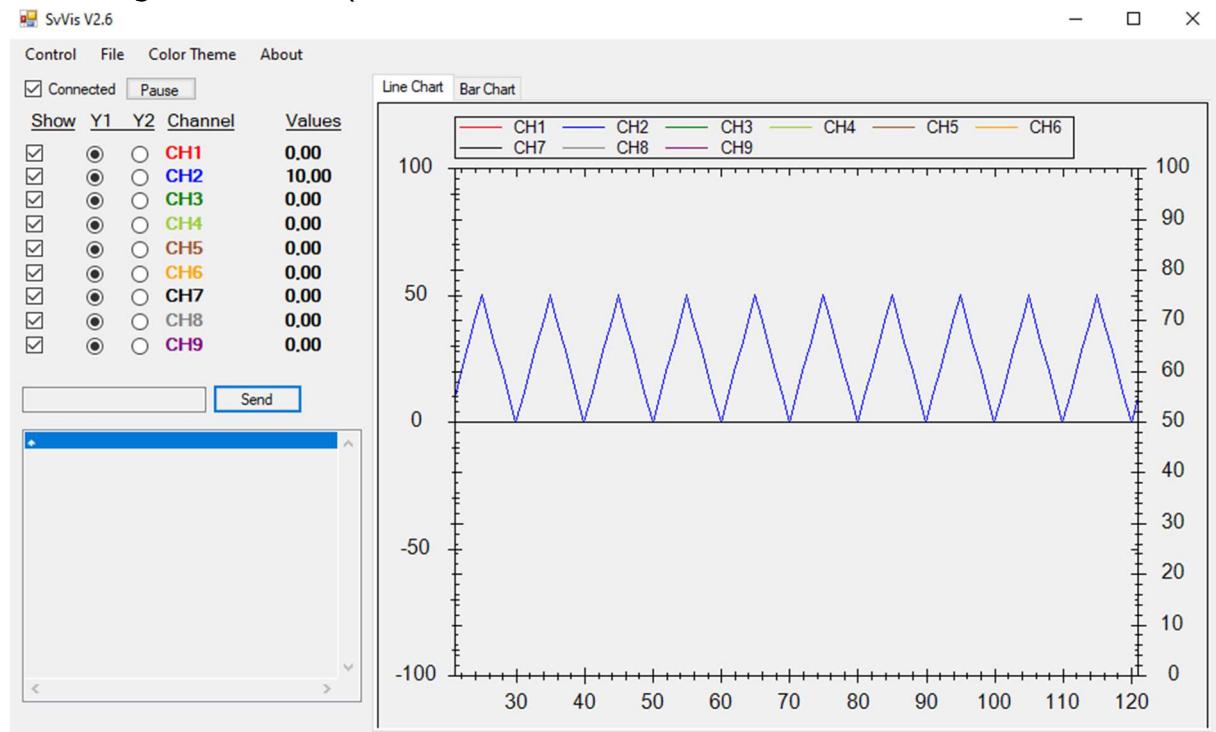


Nachdem Auf den Menüpunkt „Acq. On/Off“ geklickt wurde, wird ein Aq On – Event über die Verbindung gesendet, welche dem µC-Programm mitteilt, dass ab diesem Zeitpunkt Daten gesendet werden können. Wenn Die acquisition nicht eingeschaltet wird, werden keine Daten gesendet. Wird erneut auf denselben Menüpunkt geklickt, wird ein aq off – event gesendet, welches Mitteilt, dass keine Daten mehr gesendet werden sollen.

Um eine Nachricht zu senden, wird diese im Textfeld eingegeben und auf den danebengelegenen Button „Send“ geklickt.



In diesem Beispiel hat der µC mit mehreren Textnachrichten geantwortet. Es ist aber auch möglich, dass der µC mit „Mess“-Werten Antwortet.



7 Ergebnisse

7.1 Funktionalität Positionserkennung

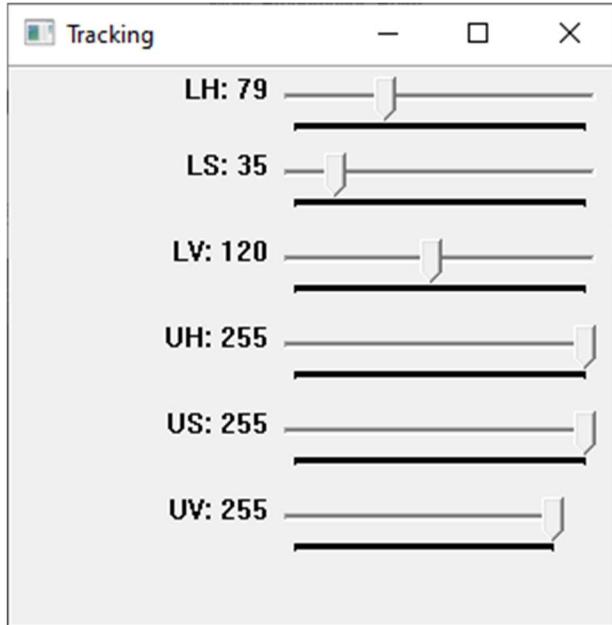
Um die Funktionalität der Positionserkennung zu beweisen, wird sie in zwei Teile aufgeteilt. In einem Erkennungs-Teil und ein einem Tracking-Teil.

7.1.1 Erkennung der Fahrzeuge

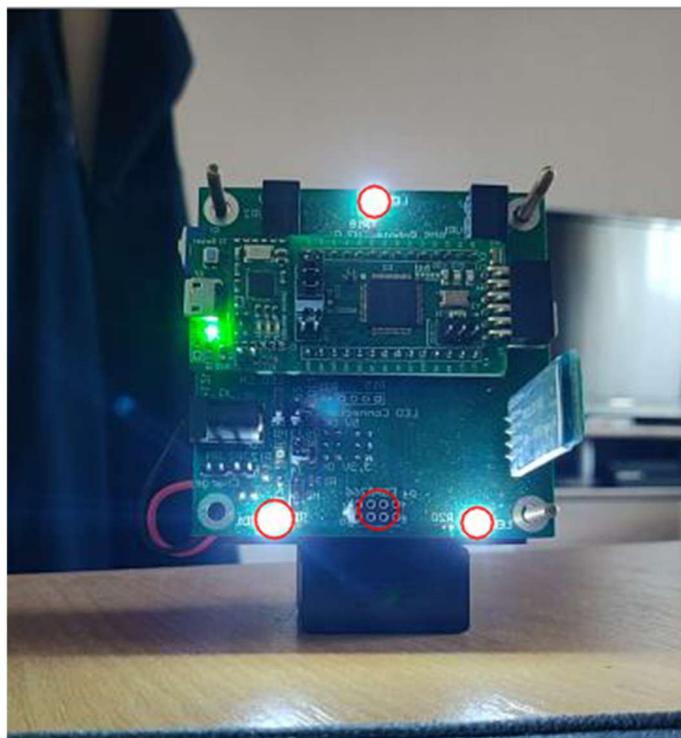
Die Erkennung wir bewiesen, indem ich die Kalibrierungsfunktion der Software verwendet um das Referenzauto. Mit den Referenzlängen messe.



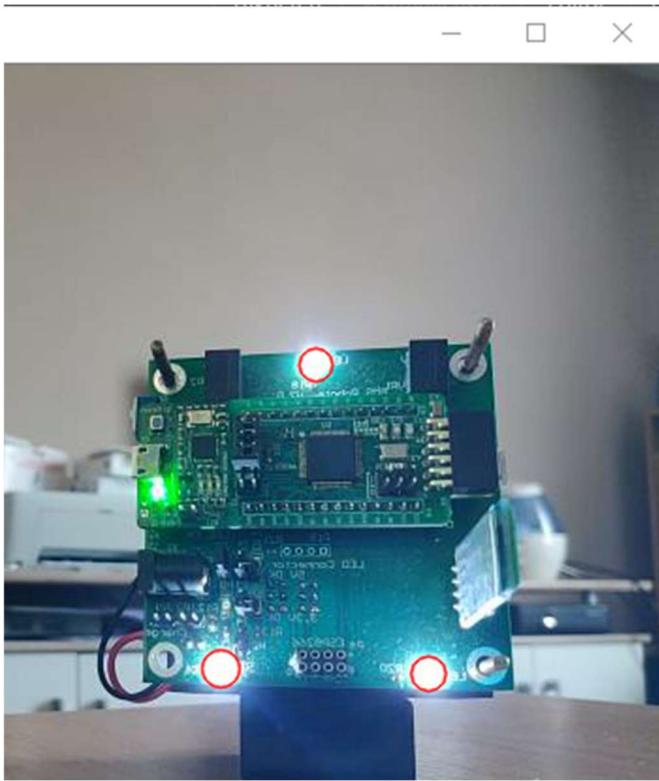
Referenzauto ohne den LEDs aktiv. Jetzt werden die LEDs aktiviert und die Parameter eingestellt nach dem erkannt werden soll.



Für diesen Test wurden diese Einstellungen verwendet. Und es entstehen diese Ergebnisse bei der Erkennung.



Hier werden zwar die LEDs erkannt aber auch eine ein PIN-HEADER durch das Licht im Raum. Daher müssen wir die Einstellungen noch etwas ändern um das gewünschte Ergebnis zu bekommen.



Hier sieht man die volle Funktionalität der Erkennung ohne, dass andere Lichtquellen erkannt werden.

7.1.2 Tracking

Um das Tracking der Software zu beweisen ist das Auto auf eine Position gestellt worden. Und dann auf eine andere.

7.1.2.1 Position 1

```
[DEBUG]Number of keypoints: 3  
[DEBUG]ABv = 107.809Pixel  
[DEBUG]BCv = 176.993Pixel  
[DEBUG]ACv = 179.207Pixel  
[DEBUG]Median Xrel: 0.553929 Median Yrel: 0.448899
```

Hier sieht man die Koordinaten Xrel & Yrel. Als auch die Distanzen zwischen den einzelnen LEDs ABv BCv und ACv.

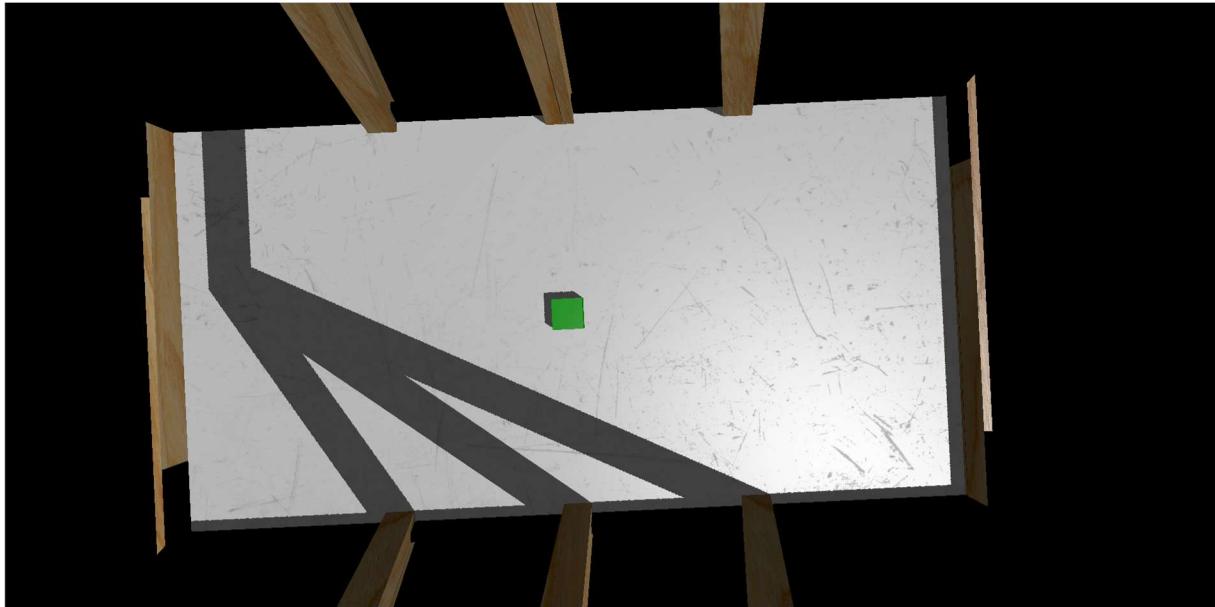
7.1.2.2 Position 2

```
[DEBUG]Number of keypoints: 3  
[DEBUG]ABv = 110.528Pixel  
[DEBUG]BCv = 170.492Pixel  
[DEBUG]ACv = 176.566Pixel  
[DEBUG]Median Xrel: 0.715333 Median Yrel: 0.558238
```

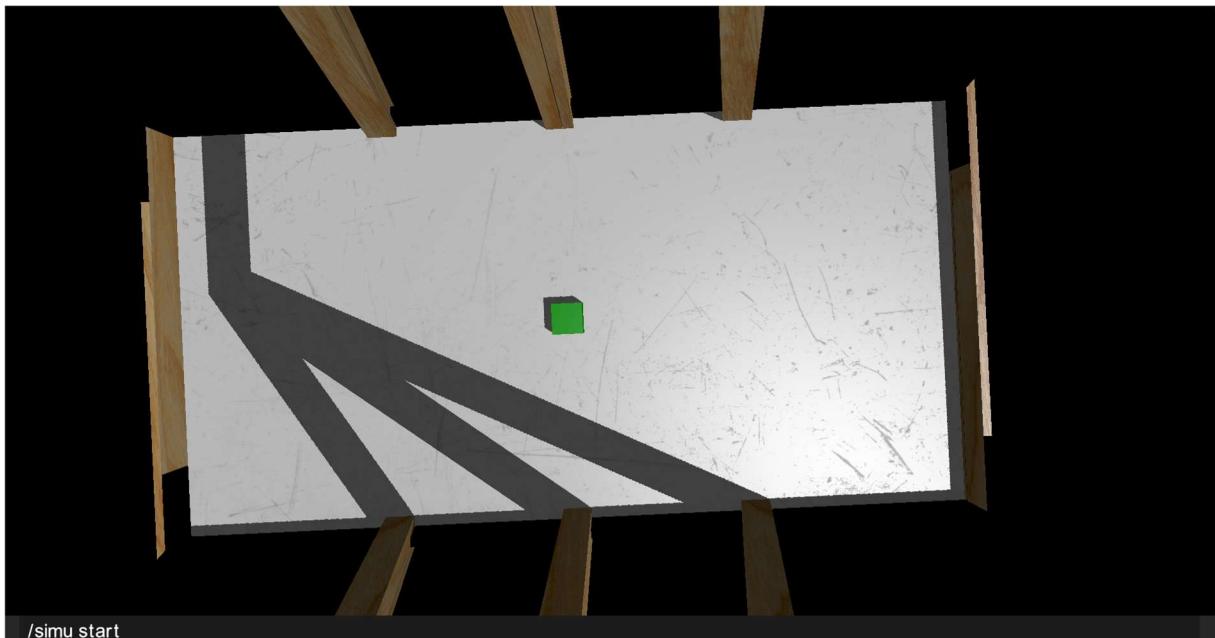
Hier sieht man die Änderung in den Positionsdaten. Man sieht auch das wie bei Position 1, drei Keypoints (drei LEDs) erkannt wurden.

7.2 Steuerung der Fahrzeuge

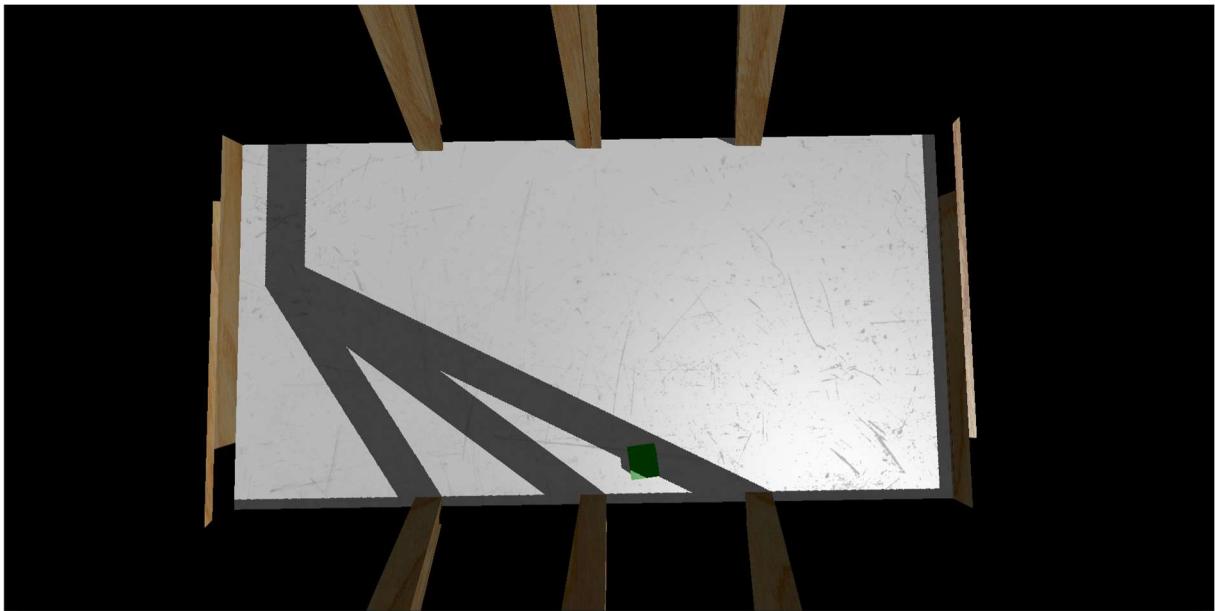
7.3 Simulationstest mit Pseudodaten



Hier sieht man wie das Fahrzeug auf dem Ursprung (Mitte des Tisches) steht.



Nun wird der Simulationsprozess mit dem Befehl /simu start eingeleitet.

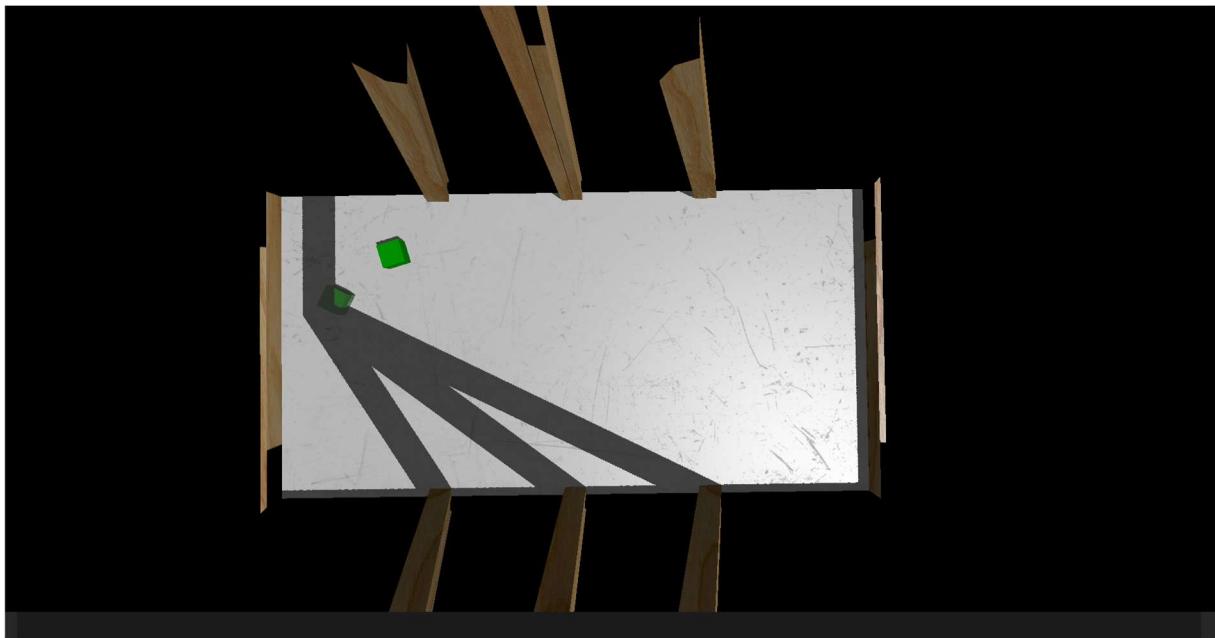


Es ist zu erkennen, dass sich das Fahrzeug bewegt hat, nämlich fährt es einen generierten Weg nach.

```
[19:22:05] [INFO / CLIENT]: Generating path from file "test.png"...
[19:22:05] [ERROR]: A command has to begin with '/'.
[19:22:05] [INFO / CLIENT]: Successfully generated path.
[19:22:07] [INFO / SIMU]: Started simulation.
[19:22:07] [INFO / SIMU]: Simulation is processed for 1 vehicles.
[19:22:32] [INFO / SIMU]: Simulation finished for vehicle 0
[19:22:32] [INFO / SIMU]: Simulation finished.
```

Das ist die Konsolenausgabe während der Simulation. Es ist zu erkennen, dass ein Weg generiert wird, die Simulation für ein Fahrzeug durchgeführt wird und zum Schluss erfolgreich endet.

7.4 Steuerung der Fahrzeuge



Das simulationsfahrzeug ist auf diesem Bild das halbtransparente und das reale Fahrzeug das nicht transparente. Wie man erkennen kann, fährt das reale Fahrzeug nicht exakt auf derselben Linie. Der Versatz kommt daher, dass das reale Fahrzeug nicht 100% gerade fahren kann und es eine Zeitverzögerung in der Regelschleife gibt.

7.5 Steuersoftware Funktionalitätstest

Die Steuersoftware ist dafür zuständig, die Steuerbefehle der Visualisierung / Simulation auf für die Fahrzeuge verständliche Befehle umzubauen. Dafür werden COM-Ports für die Verbindung verwendet. Um die COM-Ports anzusprechen, wird python verwendet, um die COM-Ports zu öffnen und als TCP/IP Server für die Swarm Control zur Verfügung zu stellen.

```
[SERVER]: connecting to conversion server at 127.0.0.1:1003
[tcptocom] Opening COM Port
[tcptocom] Waiting for connection on 0.0.0.0 10003
[SERVER]: started listening on 0.0.0.0:10002
```

So sieht ein beispielhafter Programmstart aus. Nachdem sich die Visualisierung verbinden hat sieht die Ausgabe beispielsweise so aus:

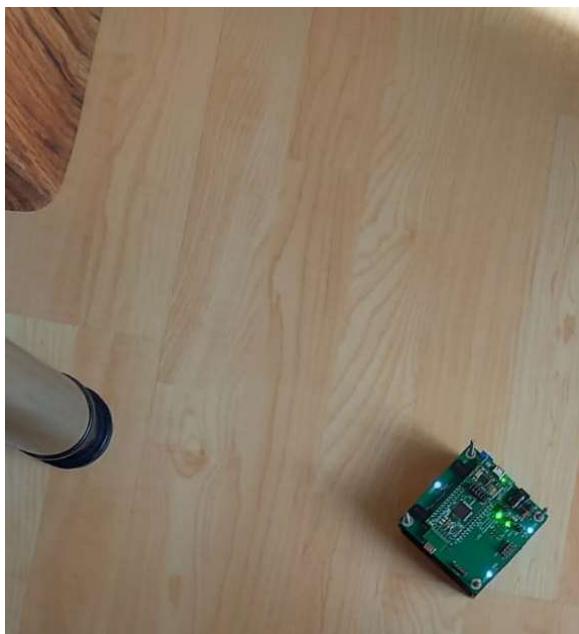
```
[SERVER]: visualisation connected
```

Sobald die Visualisierung verbunden ist und befehle sendet, sieht die Ausgabe z.B. so aus:

```
[SERVER]: Command received: 0.00126419m , 0.620791rad
[SERVER / SvVis]: Command to send: rl 35
[tcptocom] Data from tcp to serial = b'\n'
[tcptocom] Data from tcp to serial = b'rl 35\x00'
[SERVER / SvVis]: Command to send: fw 0
[SERVER]: time since last packet: 268ms
[tcptocom] Data from tcp to serial = b'\n'
[tcptocom] Data from tcp to serial = b'fw 0\x00'
```

Das Auto ändert die Position.

Vorher:



Nachher:



8 Anhang

8.1 Inbetriebnahme (f. 4Klasse TdoT)

Unbedingt Benötigte Applikationen:

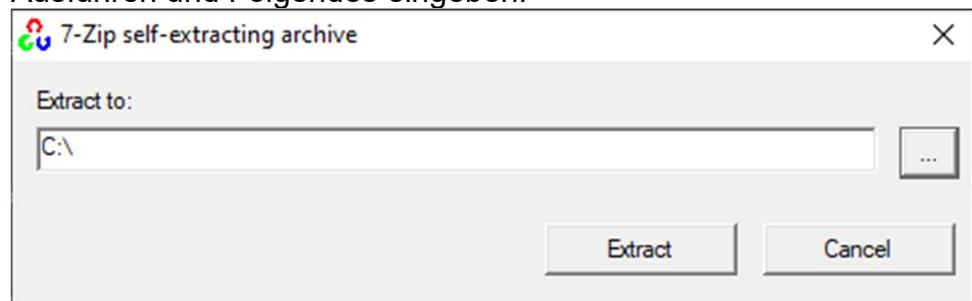
- Microsoft Visual Studio (<https://visualstudio.microsoft.com/de/downloads/>)
- Python (<https://www.python.org/ftp/python/3.9.3/python-3.9.3-amd64.exe>)
 - Die python-installation kann mit dem Befehl „python --version“ überprüft werden.
- Clang (<https://github.com/llvm/llvm-project/releases/download/llvmorg-11.0.1/LLVM-11.0.1-win64.exe>)
 - Die clang installation kann mit dem Befehl „clang++ --version“ überprüft werden.
- Keil uVision 5 (durch Werkstätte vorinstalliert)

Empfohlene zusätzliche Applikationen:

- Visual Studio Code (<https://code.visualstudio.com/download>)
- SvVis (Software vom Betreuer holen)

TODO-Liste:

- Die oben genannten Programme Installieren (alle Programme im standart-Ordner Installieren)
- Alle Benötigten Libraries installieren
 - Repository-libraries installieren (git submodule update –init)
 - OpenCV (<https://opencv.org/releases/>)
 - Diese Anleitung wurde für die Version 4.5.1 geschrieben.
 - (direkter download-link:
https://sourceforge.net/projects/opencvlibrary/files/4.5.1/opencv-4.5.1-vc14_vc15.exe/download)
 - Dieser Download enthält ein self-extracting zip-file als .exe-datei.
 - Ausführen und Folgendes eingeben:



- OpenCV wird direkt in das C:\ - Verzeichnis installiert. Danach sollte es den folgenden Order geben

This PC > Windows (C:) > opencv > sources > include				
Name	Date modified	Type	Size	
opencv2	07.06.2020 08:36	File folder		
CMakeLists.txt	07.06.2020 08:36	Text Document	1 KB	

- Eventuell ist es notwendig, den Namen der Library-Dateien in einem Projekt zu ändern. Diese Namen können im Folgenden Ordner überprüft werden.

This PC > Windows (C:) > opencv > build > x64 > vc15 > lib				
Name	Date modified	Type	Size	
opencv_world451.lib	22.12.2020 02:07	Object File Library	2 929 KB	
opencv_world451d.lib	22.12.2020 01:52	Object File Library	3 009 KB	

8.1.1 Inbetriebnahme Fahrzeug Software

Es existieren 2 Versionen der Fahrzeug Software. Die alte Version funktioniert mit dem WLAN-modul, während die neue Version einen Crack benötigt, um übersetzt zu werden. Es ist empfohlen, die neue Version zu verwenden, da die SvVis-Implementation in dieser Version ausgereifter ist.

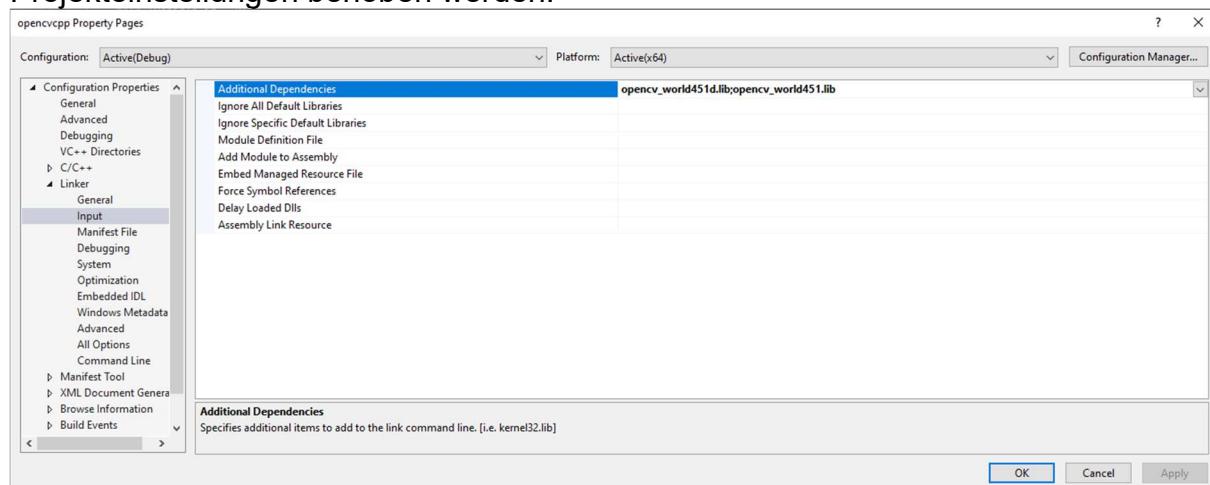
Die Fahrzeug-Software befindet sich im Ordner „SvVis/cortex_Program_new_svvis“ (bevorzugte version), oder im Ordner „SvVis/ cortex_Program_old“ (veraltete Version). Beide beinhalten ein µVision-Projekt, welches mit Keil µVision 5 geöffnet werden müssen.

Damit die Visualisierung mit den Fahrzeugen kommunizieren kann, muss noch das PC-Gegenstück der SvVis Software gestartet werden. Diese befindet sich im Ordner „SvVis\SvVis_PC“. Dieses Projekt ist ein CMake-Projekt und dieses Programm wurde mit Verwendung von Microsoft Visual Studio entwickelt und getestet. Wenn der Ordner das erste mal mit Visual Studio geöffnet wird, dauert es eine Zeit, bis das Programm übersetzt werden kann.

8.1.2 Inbetriebnahme Kamera Software

Die Kamera Software befindet sich im Ordner „positiontracking“ und ist ein Microsoft Visual Studio Projekt. (.sln – Datei). Das Projekt mit einem Doppelklick öffnen. Unter der Voraussetzung, dass OpenCV richtig installiert wurde, lässt sich das Projekt übersetzen und ausführen.

Falls es Fehler beim Übersetzen gibt, kann es sein, dass Visual Studio die falsche OpenCV Version verwenden will. Falls das der Fall ist, kann das in den Projekteinstellungen behoben werden.



Die beiden Dateinamen auf die richtige Version umbenennen, damit sich das Projekt übersetzen lässt.

8.1.3 Inbetriebnahme Visualisierung / Simulation

Der Hauptordner Des Projekts ist „visualization/SchwarmGUI“. Dieser Ordner beinhaltet ein CMake-Projekt und dieses Programm wurde mit Visual Studio entwickelt und getestet.

Anmerkung zum Starten des Programms: dieses Programm benötigt Interprozesskommunikation mit anderen Programmen.

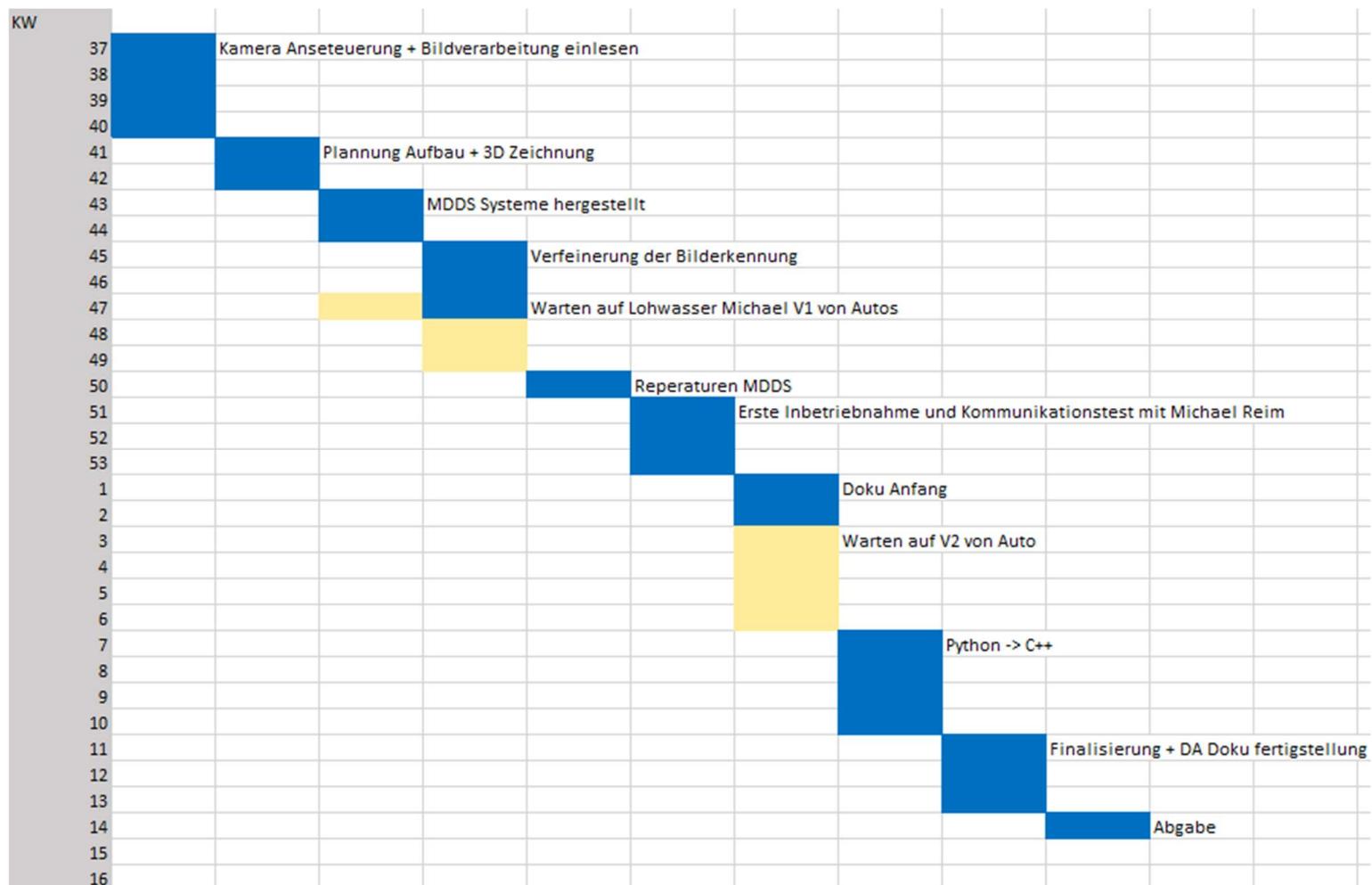
Benötigte Programme:

- Pathserver
Der Path Server kann mit dem Skript „visualization/SchwarmGUI/pathserver/start.bat“ gestartet werden.
- Kamera Software
Die Kamera Software wird über das Visual Studio Projekt „positiontracking/opencvcpp.sln“ gestartet
- Swarm Control
Die Swarm Control wird mit dem CMake-Projekt in „SvVis/SvVis_PC“ gestartet.

8.2 Projektmanagement

8.2.1 Projektplan

8.2.1.1 Mottl



8.2.1.2 Reim

Reim	KW	
37		GUI für die Visualisierung programmiert
38		
39		
40		Modelle in Blender erstellt
41		
42		
43		Programmierung von Shadern in GLSL / OpenGL
44		
45		
46		Pathgenerator und Pathserver programmiert
47		
48		
49		
50		Engine für die Simulation und Visualisierung programmiert
51		
52		
53		
1		
2		
3		
4		
5		CortexM3 gelötet
6		
7		
8		
9		
10		
11		
12		
13		Dokumentation geschrieben und Datenübertragung zur Fahrzeugsoftware fertiggestellt
14		
15		
16		

8.2.1.3 Pruggmayer

KW		
37		QR Code Generator (nicht verwendet) und erste version socket library
38		
39		
40		
41		test programme geschrieben
42		
43		
44		
45		SvVis library für µC und erste Version PC Version geschrieben
46		
47		
48		
49		MDDS System gebaut und µC programm angepasst
50		
51		warten auf AutoV1
52		
53		SvVis GUI-Programm um sockets erweitert
1		
2		
3		
4		µC Cprogramm erweitert
5		µC Programm Fehlersuche
6		µC Programm Kommunikationsversuche
7		
8		warten auf AutoV2
9		
10		
11		Zusammenbau gesamtsystem
12		
13		
14		DA Doku schreiben
15		
16		Abgabe

8.2.2 Projekttagebuch

8.2.2.1 Mottl

Datum:	KW	Zeit		h Schule:	h Freizeit:	h Gesamt:	Tätigkeit
		von	bis				
24.04.20	17	21:00	23:30	0,00	2,50	2,50	Einlesen in Python Modul OPENCV. Erste Musterprogramme getestet
25.04.20	17	22:00	0:00	0,00	2,00	2,00	Farberkennung programmiert(bisher nur eine Farbe)
27.04.20	18	8:00	12:00		4,00	4,00	Besprechung über Datenübertragung an Reim/Pruggmayer + ball tracking(camera tracking einer Farbe) Programm programmiert
28.04.20	18	22:00	23:00		1,00	1,00	Farberkennung verbessert. Unendlich viele Farben möglich
30.04.20	18	21:00	23:30		2,50	2,50	Tracking programm für Kamera/Webcams erweitert. Erste tests der TCP Übertragung.
01.05.20	18	14:00	17:00		3,00	3,00	Übertragung getestet. Bug Fixes. Barcode/QR Scanner entwickelt & getestet
02.05.20	18	10:00	11:30		1,50	1,50	TCP connection auf binary geändert. Größen der Variablen gbesprochen mit Reim & Pruggmayer und angelegt(z.B. int_8t für RGB)
18.05.20	21	10:00	11:00		1,00	1,00	DA- Besprechung Vorgangsweise
27.05.20	22	10:00	11:00		1,00	1,00	DA- Besprechung Vorgangsweise BULME LÖSUNG
03.06.20	23	18:30	19:00		0,50	0,50	DA- Besprechung Graz Reise
05.06.20	23	6:30	17:21		10,85	10,85	Graz Reise Informationen und Ideen gesammelt
10.06.20	24	8:00	11:25	3,42	0,00	3,42	Blop Erkennung Programmiert und kleine erste Tests durchgeführt
10.06.20	24	11:30	16:35	5,08	0,00	5,08	Erste Tests der Blop Erkennung mit Elfie. Bestellliste für Autos fertiggestellt. Verbindung mit VPN hergestellt
26.06.20	26	7:50	16:35		8,75	8,75	MDMMS Bestücken und löten
27.06.20	26	7:50	16:35		8,75	8,75	MDMMS Bestücken und löten
16.07.20	29	8:50	17:50		9,00	9,00	Elfie Aufbau mit Fehlerbehebung
24.07.20	30	16:00	21:45		5,75	5,75	Elfie Verkabelung und Test der Motoren
30.07.20	31	13:20	18:25		5,08	5,08	Elfie Steuersoftware entwicklung und finalisierung
04.08.20	32	8:30	12:10		3,67	3,67	LED-Richtungsdreieck löten und erste Tests zur erkennung
08.08.20	32	11:15	18:15		7,00	7,00	Einlesen in Python Wrapper für Kamera
12.08.20	33	10:35	16:45		6,17	6,17	Linux installiert und Kamera mithilfe des Python Wrappers angesteuert und eingestellt
24.08.20	35	18:30	18:45		0,25	0,25	TCP to UART nicht blockierend Skript

Camera Controlled Swarm Robots

							für Pruggmayer programmiert
11.09.20	37	7:50	13:10	4,33	1,00	5,33	Lagebesprechung mit Betreuer. Tests mit Elfie. VPN Probleme gelöst
02.10.20	40	7:50	13:10	4,33	1,00	5,33	ARM Cortex MDDS M4 System repariert und fertiggestellt
05.10.20	41	16:30	19:45		3,25	3,25	Erste Prototyp für Server Struktur für WLAN Module
09.10.20	41	7:50	13:10	4,33	1,00	5,33	Positionstracking Software kommentiert und leserlich umgeschrieben
12.10.20	42	16:35	18:40		2,08	2,08	Tischaufbau in Blender (Reim) verwendet für erste Baupläne
14.10.20	42	14:20	19:11		4,85	4,85	Prototyp für Kommunikation Positionstracking --> Simulation programmiert
16.10.20	42	7:50	13:10	5,33	0,00	5,33	MDDS System löten
23.10.20	43	7:50	13:10	5,33	0,00	5,33	Projektplan
30.10.20	44	7:50	13:10	5,33	0,00	5,33	Projektplan
06.11.20	45	7:50	13:10	5,33	0,00	5,33	DA-Doku
20.11.20	47	7:50	13:10	5,33	0,00	5,33	MDDS System löten
27.11.20	48	7:50	13:10	5,33	0,00	5,33	Tischaufbau reparieren
04.12.20	49	7:50	13:10	5,33	0,00	5,33	Tischaufbau reparieren
22.01.21	3	7:50	13:10	5,33	0,00	5,33	MDDS System löten
29.01.21	4	7:50	13:10	5,33	0,00	5,33	MDDS System löten
05.02.21	5	7:50	13:10	5,33	0,00	5,33	Kamera Software von Python --> C++
12.02.21	6	7:50	13:10	5,33	0,00	5,33	Kamera Software von Python --> C++
19.02.21	7	7:50	13:10	5,33	0,00	5,33	Kamera Software von Python --> C++
26.02.21	8	7:50	13:10	5,33	0,00	5,33	Kamera Software von Python --> C++
05.03.21	9	7:50	13:10	5,33	0,00	5,33	Kamera Software von Python --> C++
12.03.21	10	7:50	13:10	5,33	0,00	5,33	Kamera Software von Python --> C++
19.03.21	11	7:50	13:10	5,33	0,00	5,33	Kamera Software von Python --> C++
26.03.21	12	7:50	13:10	5,33	0,00	5,33	Kamera Software von Python --> C++
29.03.21	13	10:00	23:00		13,00	13,00	DA-DOKU / Fertigstellung und letzte Funktionstests
30.03.21	13	14:00	22:00		8,00	8,00	DA-DOKU / Fertigstellung und letzte Funktionstests
31.03.21	13	14:00	23:00		9,00	9,00	DA-DOKU / Fertigstellung und letzte Funktionstests
01.04.21	13	10:00	23:00		13,00	13,00	DA-DOKU / Fertigstellung und letzte Funktionstests
02.04.21	13	14:00	21:00		7,00	7,00	DA-DOKU / Fertigstellung und letzte Funktionstests
03.04.21	13	14:00	23:00		9,00	9,00	DA-DOKU / Fertigstellung und letzte Funktionstests
02.04.21	13	7:50	13:10	0,00	5,33	5,33	DA Doku
04.04.21	13	13:00	23:00		10,00	10,00	DA-DOKU / Fertigstellung und letzte Funktionstests

8.2.2.2 Reim

Dat um :	K W	Zeit		h Sch ule:	h Frei zeit :	h Ges amt :	Tätigkeit
		v o n	b i s				
24. 04. 20	1 7				3,0 0	3,00	Testdaten eingelesen und graphisch als punkte am Bildschirm angezeigt.
27. 04. 20	1 8				4,0 0	4,00	Datenübertragung besprochen
29. 04. 20	1 8				4,0 0	4,00	Diverse Projektbesprechungen
01. 05. 20	1 8				2,0 0	2,00	Datenübertragung getestet und diverse Bugfixes
05. 05. 20	1 9				3,0 0	3,00	
28. 05. 20	2 2				3,0 0	0,00	
25. 06. 20	2 6	10 :0 0	14 :5 0	4,8 3	0,0 0	4,83	Cortex minimalsysteme gelötet und Kurzschlüsse entfernt
26. 06. 20	2 6	7: 50	11 :2 0	3,5 0	0,0 0	3,50	Cortex minimalsysteme gelötet, Kurzschlüsse entfernt, ersten funktionierenden Cortex getestet
29. 06. 20	2 7	7: 50	13 :3 0	5,6 7	0,0 0	5,67	Cortex minimalsystem gelötet, Motherboard für Cortex gelötet, Projektbesprechung
29. 06. 20	2 7	14 :3 0	16 :0 0	1,5 0	0,0 0	1,50	Cortexsysteme auf Fehler kontrolliert, Fehler behoben, Cortexsysteme getestet
30. 06. 20	2 7	7: 50	14 :3 0	6,6 7	0,0 0	6,67	Fehler bei nicht-funktionierenden Cortexsystemen behoben und getestet
01. 07. 20	2 7	15 :2 0	17 :1 0		1,8 3	1,83	Programmierung der Headerfile eines Eventhandlers.
02. 07. 20	2 7	16 :4 5	18 :3 0		1,7 5	1,75	Deklarierte Funktionen des Eventhandlers ausprogrammiert und Testprogramm geschrieben.
03. 07. 20	2 7	13 :0 0	14 :1 0		1,1 7	1,17	Fehlersuche, erstellen der Library.
12. 07. 20	2 8	12 :0 0	14 :4 5		2,7 5	2,75	Erstellung und ausprogrammierung der "Server" klasse der MultiSocketHanlder Library.
13.	2	12	14		2,0	2,00	Ausprogarmmierung der Methoden der Server klasse.

07.	9	:3 0	:3 0		0			
14.	2	15 :1 0	17 :0 0		1,8 3	1,83	Test-Server zum Testen der Server-klasse geschrieben, Fehler behoben, Änderungen am Sourcecode vorgenommen.	
15.	2	11 :0 0	14 :0 0		3,0 0	3,00	Socket-Handler-Klasse entworfen und angefangen auszuprogrammieren, intensive Fehlersuche	
16.	2	11 :3 0	14 :2 0		2,8 3	2,83	Fehler behoben (falsche Abfrage in der Hauptschleife des Programms) im Testprogramm, Ausprogrammierung der Socket-Hanlder-Klasse, Anpassungen vorgenommen.	
17.	2	11 :1 5	15 :0 0		3,7 5	3,75	Ausprogrammierung der Socket-Hanlder-Klasse, Testprogramm geschrieben, Socket-Handler in Verbindung mit dem Server getestet.	
19.	2	11 :4 0	14 :5 0		3,1 7	3,17	Änderungen am Server und Socket-Handler vorgenommen, getestet.	
20.	3	10 :2 0	13 :0 0		2,6 7	2,67	Optimierungen am Server und Socket-Handler vorgenommen, Änderungen und Anpassung vorgemommen.	
21.	3	12 :1 0	15 :2 0		3,1 7	3,17	Client-Klasse entworfen, angefangen Methoden auszuprogrammieren.	
22.	3	13 :0 0	14 :1 0		1,1 7	1,17	Methoden der Client-Klasse ausprogrammiert. Client klasse umgeschrieben, damit sie optimierter ist.	
23.	3	11 :0 0	12 :4 0		1,6 7	1,67	Methoden der Server und Socket-Hanlder-Klasse kommentiert und dokumentiert.	
24.	3	10 :3 0	14 :0 0		3,5 0	3,50	Client-Klasse fertiggestellt, Test-Client geschrieben, Test-Server umgeschrieben, Anpassungen vorgenommen, Änderungen vorgenommen.	
25.	3	10 :5 0	13 :3 0		2,6 7	2,67	Methoden der Client-Klasse kommeniert und dokumentiert, Anpassnungen am Test-Client und Server vorgenommen, Version 1.0.0 der Library fertiggestellt.	
26.	3	12 :0 0	13 :0 0		1,0 0	1,00	Library angepasst und Version 1.0.1 der Library erstellt.	
27.	3	14 :0 0	18 :0 0		4,0 0	4,00	Font-Klasse einer bestehenden Library komplett überarbeitet, Änderungen, Anpassungen, Optimierungen vorgenommen.	
28.	3	15 :3 0	19 :1 0		3,6 7	3,67	Character-Klasse der selben Library komplett überarbeitet, Änderungen, Anpassungen, Optimierungen vorgenommen.	
29.	3	11 :0 0	15 :3 0		4,5 0	4,50	Shader-Library für OpenGL programmiert, Shader-Klasse entworfen, Methoden ausprogrammiert.	
29.	3	16 :0 0	17 :1 0		1,1 7	1,17	Methoden der Shader-Klasse ausprogrammiert.	
29.	3	22 :0 0	23 :4 0		1,6 7	1,67	Testprogramm erstellt, Fehler korrigiert, Anpassungen vorgenommen, Erstellung der Version 1.0.0 der Library.	

31. 07. 20	3 1	11 :3 0	13 :3 0		2,0 0	2,00	Deklaierung der Methoden der String-Klasse für OpenGL
03. 08. 20	3 2	11 :0 0	14 :4 5		3,7 5	3,75	Ausprogrammierung der Methoden für die String-Klasse
04. 08. 20	3 2	12 :0 0	13 :3 0		1,5 0	1,50	Ausprogrammierung der Methoden für die String-Klasse fertiggestellt und Änderungen vorgenommen
12. 08. 20	3 3	13 :0 0	15 :1 5		2,2 5	2,25	Klassen der Library getestet, Testprogramm geschrieben, Fehler und Errors behoben
13. 08. 20	3 3	21 :0 0	23 :5 9		2,9 8	2,98	Optimierungen vorgenommen, Library kommentiert und erstellt
07. 09. 20	3 7	11 :2 0	15 :0 0	3,6 7	0,0 0	3,67	Erstellung des Projektes für das GUI
08. 09. 20	3 7	8: 00	11 :2 0	3,3 3	0,0 0	3,33	Erstellung einer Klasse für Buttons & erstellung von Button-Events
09. 09. 20	3 7	8: 00	10 :0 0	2,0 0	0,0 0	2,00	Erstellung des Button-Handlers
10. 09. 20	3 7	8: 00	10 :4 0	2,6 7	0,0 0	2,67	Am Button-Handler programmiert und getestet, Button-Renderer erstellt
14. 09. 20	3 8	8: 40	11 :3 0	2,8 3	0,0 0	2,83	Button-Render programmiert
15. 09. 20	3 8	12 :0 0	15 :0 0	3,0 0	0,0 0	3,00	Button-Renderer programmiert und Anpassungen vorgenommen
17. 09. 20	3 8	9: 00	15 :0 0	6,0 0	0,0 0	6,00	OpenGL auf Linux aufsetzen, Testprogramm zum Laufen bringen, Update von GLFW2.7 zu GLFW 3.3.2
18. 09. 20	3 8	8: 00	13 :0 0	5,0 0	0,0 0	5,00	Befehlseingabe programmiert
25. 09. 20	3 9	8: 00	13 :0 0	5,0 0	0,0 0	5,00	Userinterface fertiggestellt, 3D renderscene programmiert
02. 10. 20	4 0	8: 00	13 :0 0	5,0 0	0,0 0	5,00	In blender eingelesen, model vom Tisch erstellt
09. 10. 20	4 1	8: 00	13 :0 0	5,0 0	0,0 0	5,00	Tisch texturiert
16. 10. 20	4 2	8: 00	13 :0 0	5,0 0	0,0 0	5,00	Klasse geschrieben, mit der man .obj (3D Object) und .mtl (Material) files einlesen kann.
18. 10.	4 2	13 :2	15 :4		2,3 3	2,33	Models bearbeitet / korrigiert, UV-Mapping von Texturen bearbeitet / korrigiert.

Camera Controlled Swarm Robots

20		0	0					
19.	4	14 :5 0	19 :0 0		4,1 7	4,17		In das Phong Lichtmodell eingelesen, angewendet und ausprogrammiert.
10.	3							
20								
20.	4	15 :1 5	21 :0 0		5,7 5	5,75		In Materials, Lichtmaps, sowie in die Funktionalität von verschiedenen und mehreren Lichtquellen eingelesen. Materials und Lichtmaps (ambient - diffuse - map und specular - map) erstellt und implementiert.
10.	3							
20								
21.	4	17 :3 0	21 :1 0		3,6 7	3,67		2. Lichtquelle implementiert (Glühbirne). Herumexperimentiert mit verschiedenen Lichtparametern, um ein gutes Ergebnis zu bekommen.
10.	3							
20								
22.	4	14 :4 0	18 :0 0		3,3 3	3,33		Addition von Lichtquellen verbessert. Verbessertes Licht-Rendering implementiert.
10.	3							
20								
07.	4	13 :0 0	19 :0 0		6,0 0	6,00		Rendering von Schatten, ausbesserung von Nebeneffekten vom Schattenrendering.
11.	5							
20								
14.	4	15 :0 0	20 :0 0		5,0 0	5,00		Implementierung von Fahrzeugen.
11.	6							
20								
22.	4	17 :0 0	22 :0 0		5,0 0	5,00		Generator programmiert, der aus einem Bildfile einen Path für die Fahrzeuge generiert.
11.	7							
20								
30.	4	16 :3 0	21 :4 5		5,2 5	5,25		Pathgenerator verbessert, fehler behoben.
11.	9							
20								
04.	4	8: 9	13 00		5,0 0	5,00		Server programmiert, der Daten über den Path zur Simulations-Software sendet.
12.	9							
20								
05.	4	15 :3 0	20 :1 5		4,7 5	4,75		Server weiterprogrammiert, testung der Datenübertragung.
12.	9							
20								
06.	4	12 :0 0	16 :0 0		4,0 0	4,00		Fehler des Servers behoben und Verbesserungen vorgenommen.
12.	9							
20								
11.	5	8: 0	13 00		5,0 0	5,00		Mechanik der Bewegungen der Fahrzeuge (setzen auf Positionen).
12.	0							
20								
18.	5	8: 1	13 00		5,0 0	5,00		Flüssige Bewegung der Fahrzeuge.
12.	1							
20								
19.	5	13 :0 0	20 :0 0		7,0 0	7,00		Fehler gesucht und behoben (Falsche Bewegung) und Optimierungen vorgenommen.
12.	1							
20								
20.	5	15 :0 0	21 :0 0		6,0 0	6,00		Flüssige Rotationen der Fahrzeuge dazuprogrammiert.
12.	1							
20								
28.	5	12 :0 0	19 :0 0		7,0 0	7,00		Rotationen verbessert, bugs behoben.
12.	3							
20								
08.	1	07 :5 0	13 :1 0	5,3 3	0,0 0	5,33		Übertragung zur Fahrzeugsoftware angefangen zu programmieren
01.	1							
21								

Camera Controlled Swarm Robots

15. 01. 21	2	7: 50	13 :1 0	5,3 3	0,0 0	5,33	Proxy-Server für die Erkennungssoftware geschrieben (wurde wegen der Python implementation verwendet, wird mittlerweile nicht mehr gebraucht)
22. 01. 21	3	07 :5 0	13 :1 0	5,3 3	0,0 0	5,33	Mechanik implementiert, damit man das reale Auto verfolgen kann in der Visualisierung
29. 01. 21	4	7: 50	13 :1 0	5,3 3	0,0 0	5,33	Mechanik implementiert, damit man das reale Auto verfolgen kann in der Visualisierung
05. 02. 21	5	07 :5 0	13 :1 0	5,3 3	0,0 0	5,33	CortexM3 gelötet
12. 02. 21	6	7: 50	13 :1 0	5,3 3	0,0 0	5,33	CortexM3 gelötet
19. 02. 21	7	07 :5 0	13 :1 0	5,3 3	0,0 0	5,33	CortexM3 gelötet
26. 02. 21	8	7: 50	13 :1 0	5,3 3	0,0 0	5,33	CortexM3 gelötet
05. 03. 21	9	07 :5 0	13 :1 0	5,3 3	0,0 0	5,33	CortexM3 gelötet
12. 03. 21	10	7: 50	13 :1 0	5,3 3	0,0 0	5,33	CortexM3 gelötet
19. 03. 31	12	07 :5 0	13 :1 0	5,3 3	0,0 0	5,33	CortexM3 gelötet
26. 03. 21	12	7: 50	13 :1 0	5,3 3	0,0 0	5,33	CortexM3 gelötet
29. 03. 21	13	14 :3 0	23 :0 0		8,5 0	8,50	Dokumentation geschrieben
30. 03. 21	13	14 :0 0	22 :0 0		8,0 0	8,00	Dokumentation geschrieben
01. 04. 21	13	10 :0 0	21 :0 0		11, 00	11,0 0	Übertragung zur Erkennungssoftware fertiggestellt, Projekt auf Visual Studio und clang-compiler umgeschrieben
02. 04. 21	13	14 :0 0	20 :0 0		6,0 0	6,00	Übertragung zur Fahrzeugsoftware fertiggestellt
03. 04. 21	13	14 :0 0	17 :0 0		3,0 0	3,00	Fertiges gesamtmodell getestet und Dokumentation aktualisiert

8.2.2.3 Pruggmayer

29.04.2020	18	13:30	18:00	0,00	4,50	4,50	Beginn QR to CMD
01.05.2020	18	13:00	15:00	0,00	2,00	2,00	QR Code Generator um parametereingabe erweitert
03.05.2020	18	13:00	16:00	0,00	3,00	3,00	QR Code um skalieroption erweitert
05.05.2020	19	9:30	11:30	0,00	2,00	2,00	QR Code Skalierbuffer von stack in Heap verlegt
20.05.2020	21	15:00	18:00	0,00	3,00	3,00	TCP_lib für IPv4 Implementiert
21.05.2020	21	12:00	15:00	0,00	3,00	3,00	TCP lib erweitert und Fehler behoben
22.05.2020	21	13:00	16:00	0,00	3,00	3,00	TCP lib Fehlerbehebungen und Socket für IPv6
23.05.2020	21	12:00	17:00	0,00	5,00	5,00	TCP Server für IPv6
24.05.2020	21	9:00	14:00	0,00	5,00	5,00	TCP Client für IPv6 und Hostname resolve für IPv6
10.06.2020	24	9:00	18:00	0,00	9,00	9,00	TCP lib in Socket lib umbenannt und UDP_client hinzugefügt
17.06.2020	25	12:00	13:00	0,00	1,00	1,00	TCP lib move and swap functionality
18.06.2020	25	16:00	20:00	0,00	4,00	4,00	TCP Server get listen ip / port + google spreadsheet angelegt
19.06.2020	25	15:00	18:00	0,00	3,00	3,00	Socket library leichter zu verwenden gemacht
29.06.2020	27	12:00	16:00	0,00	4,00	4,00	echo-server geschrieben
30.06.2020	27	12:00	16:00	0,00	4,00	4,00	request-server mit .cfg file geschrieben
08.07.2020	28	12:00	15:00	0,00	3,00	3,00	request-server um logging-optionen erweitern
19.07.2020	29	9:00	15:00	0,00	6,00	6,00	SvVis library für µC mit RTOS neuschreiben
20.07.2020	30	9:00	16:00	0,00	7,00	7,00	SvVis library für µC mit RTOS neuschreiben fertiggestellt
23.07.2020	30	12:00	17:00	0,00	5,00	5,00	SvVis library für µC verbessert
25.07.2020	30	12:00	13:00	0,00	1,00	1,00	beginn SvVis library für PC
26.07.2020	30	9:00	10:00	0,00	1,00	1,00	request client geschrieben
03.08.2020	32	17:00	18:00	0,00	1,00	1,00	Socket zu Serielle Schnittstelle - skript geschrieben
04.08.2020	32	16:00	20:00	0,00	4,00	4,00	SvVis Library für PC begonnen
14.08.2020	33	20:00	21:00	0,00	1,00	1,00	SvVis Library für PC erweitert
23.08.2020	34	19:00	21:00	0,00	2,00	2,00	bessere Socket library begonnen
04.09.2020	36	11:00	17:00	0,00	6,00	6,00	SvVis library für PC mit besserer Socket library umgeschrieben
05.09.2020	36	13:00	16:00	0,00	3,00	3,00	bessere Socket library fertig gestellt und Fehler behoben
11.09.2020	37	7:50	13:10	5,33	0,00	5,33	SvVis library portable implementiert
12.09.2020	37	12:30	14:30	2,00	0,00	2,00	cppsockets um tcp funktionalitäten erweitert
17.09.2020	38	9:00	16:00	7,00	0,00	7,00	Linux subsystem installiert, GLEW applikation fehler suche
18.09.2020	38	10:40	11:00	0,33	0,00	0,33	Cortex Peripherie f. Lohwasser gruppe festgelegt
18.09.2020	38	11:45	13:10	1,42	0,00	1,42	Github repository angelegt
25.09.2020	39	7:50	13:10	1,00	4,33	5,33	Manjaro Linux Umgebung installiert
02.10.2020	40	7:50	13:10	1,00	4,33	5,33	MDDS Cortex repariert
06.10.2020	41	9:45	10:20	0,00	0,58	0,58	MDDS Cortex baudratenehler workaround gefunden
09.10.2020	41	7:50	13:10	0,00	5,33	5,33	MDDS Cortex Fehlersuche
09.10.2020	41	14:00	17:00	3,00	0,00	3,00	Problembehebung Strunz und Testprogramm für Ellfi geschrieben

Camera Controlled Swarm Robots

10.10.2020	41	12:00	14:00	2,00	0,00	2,00	Fehlersuche und Behebung SvVis cortex library
14.10.2020	42	8:20	8:40	0,33	0,00	0,33	Fehlerbehebung cppsock library
16.10.2020	42	7:50	13:10	1,00	4,33	5,33	SvVis Library in c++ umgeschrieben
19.10.2020	43	13:00	16:00	3,00	0,00	3,00	SvVis Library auf Interrupts umgeschrieben
13.11.2020	46	8:00	12:00	0,00	4,00	4,00	SvVis test server schreiben, C partition Speicherplatz beschaffen
20.11.2020	47	11:00	14:00	0,00	3,00	3,00	Multithread syncronisation testing
24.11.2020	48	14:00	19:00	3,00	2,00	5,00	SvVis Um Sockets erweitert
23.11.2020	48	11:00	14:00	0,00	3,00	3,00	SvVis um Color Themes erweitert
26.11.2020	48	15:00	17:00	0,00	2,00	2,00	SvVis Color Theme Aufgeräumt
27.11.2020	48	10:00	13:00	0,00	3,00	3,00	Cortex Programm auf Neue Pinbelegung umgeschrieben & PWD-Generierung hinzugefügt
01.12.2020	49	8:00	12:00	0,00	4,00	4,00	SvVis Programm Color Theme Fehlerbehebungen
14.12.2020	51	9:00	11:00	0,00	2,00	2,00	Cortex Programm Positions LEDS Ansteuerung
15.12.2020	51	14:00	16:35	2,58	0,00	2,58	SvVis Dark Theme fertig gestellt
26.12.2020	52	9:00	13:00	0,00	4,00	4,00	Software für Fahrzeug PWM mit Hardwretimer generieren lassen
27.12.2020	52	12:00	15:00	0,00	3,00	3,00	Software für Fahrzeug Übertragung multi-interfacefähig gemacht
30.12.2020	53	10:00	14:00	0,00	4,00	4,00	Bluetooth modul funktionstüchtig gemacht
02.01.2021	53	9:00	15:00	0,00	6,00	6,00	WLAN kommunikationsversuche (cortex zu WLAN modul)
03.01.2021	53	10:00	14:00	0,00	4,00	4,00	WLAN kommunikationsversuche (cortex zu WLAN modul)
05.01.2021	1	11:00	13:00	0,00	2,00	2,00	Projekt "update"
06.01.2021	1	11:00	15:00	0,00	4,00	4,00	erste WLAN kommunikationsversuche
07.01.2021	1	11:00	15:00	0,00	4,00	4,00	weitere Kommunikationsversuche
09.01.2021	1	11:00	13:00	0,00	2,00	2,00	WLAN modul funktionalität bestätigt, WLAN library implementation begonnen
15.01.2021	2	12:00	15:00	0,00	3,00	3,00	schwarm driver vorbereitet
19.01.2021	3	11:00	13:00	0,00	2,00	2,00	cortex programm um PWM änderung erweitert
22.01.2021	3	8:00	13:00	5,00	0,00	5,00	cortex program byte queues auf ring pipes umgeschrieben
23.01.2021	3	11:00	17:00	0,00	6,00	6,00	cortex programm kommentare geschrieben und motorregelung begonnen
27.01.2021	4	18:30	19:30	0,00	1,00	1,00	auto ruiniert und mögliche Lösungen besprochen
29.01.2021	4	12:00	17:00	0,00	5,00	5,00	auto programm um WLAN erweitern versucht
30.01.2021	4	13:00	20:00	0,00	7,00	7,00	auto programm um WLAN erweitern versucht
31.01.2021	4	14:00	19:00	0,00	5,00	5,00	Auto Programm um WLAN erweitert
12.02.2021	6	8:00	13:30	5,50	0,00	5,50	DA schultag
05.03.2021	9	7:50	15:00	0,00	7,17	7,17	MDDS Fehlersuche
12.03.2021	10	7:50	13:10	5,33	0,00	5,33	Zusammenbau 2 MDDS Boards
15.03.2021	11	11:00	14:00	1,00	2,00	3,00	Umschreibung SvVis library
20.03.2021	11	10:30	20:30	0,00	10,00	10,00	DA neu organisiert, socket collection zu cppsock hinzugefügt
29.03.2021	13	14:00	19:00	0,00	5,00	5,00	DA Doku schreiben
30.03.2021	13	14:00	20:00	0,00	6,00	6,00	DA Doku schreiben
31.03.2021	13	14:00	17:00	0,00	3,00	3,00	DA Doku schreiben & cppsock fehler beheben

01.04.2021	13	12:00	19:00	0,00	7,00	7,00	DA Doku schreiben + COM-Python script verbessert
02.04.2021	13	13:00	20:00	0,00	7,00	7,00	DA weiterarbeiten
03.04.2021	13	15:00	20:00	0,00	5,00	5,00	DA testing
04.04.2021	13	13:00	20:00	0,00	7,00	7,00	DA Doku schreiben

8.2.3 Projektkosten

Projektkosten	Ausgabe	Preis
	Kamera + Linse	299,00 Euro
	Holz für Aufbau	20 Euro
		Gesamt
		319,00 Euro

9 Quellenverzeichnis

9.1 Bücher

9.2 Onlinemedien

9.2.1 [\[1\] Wikipedia HSV](#)

9.2.2 [Dokumentation CMSIS-RTOS](#)

9.3 Zeitschriften