**DIPLOMARBEIT**

Gesamtprojekt

**Camera Controlled Swarm Robots**

**Positionserkennung**

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

**Swarm Controll**

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

**Visualisierung und Simulation**

Michael reim 5BHEL Betreuer/in: 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 selbständig mit RTOS realisiert. Visualisierung wurde von Grund auf in C++ + OpenGL geschrieben für den speziellen Anwendungsfall. |  |

|  |  |
| --- | --- |
| Typische Grafik, Foto etc.  (mit Erläuterung) | 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 wir 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/Abteilungsvorständin |

**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 out specific purpose. |

|  |  |
| --- | --- |
| Illustrative graph, photo  (incl. explanation) | The graph above pictures the whole System. There you can see the three main components. Tracking, Swarm Control and Visualisation and Simulation. The communication between the individual blocks is realised in TCP. The communication between laptop and cars can be established in two ways WLAN, Bluetooth. |

|  |  |
| --- | --- |
| 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 10](#_Toc67935159)

[1.1 Blockschaltbild 10](#_Toc67935160)

[1.2 Funktionsbeschreibung 10](#_Toc67935161)

[2 Positionserkennung 11](#_Toc67935162)

[2.1 Übersicht 11](#_Toc67935163)

[2.2 Aufbau 12](#_Toc67935164)

[2.2.1 Korrekter Aufbau 12](#_Toc67935165)

[2.3 Kamera 13](#_Toc67935166)

[2.3.1 Kalibrierung der Kamera 13](#_Toc67935167)

[2.4 Erkennung Positions LEDs 15](#_Toc67935168)

[2.4.1 Algorithmus zur Erkennung 16](#_Toc67935169)

[2.4.1.1 Blob Detection 16](#_Toc67935170)

[2.4.1.1.1 Beispielparameter 16](#_Toc67935171)

[2.4.1.2 Filtern von Blobs nach Farbe, Größe, Form 17](#_Toc67935172)

[2.4.2 Berechnung der Position 18](#_Toc67935173)

[2.4.3 Optimierung 18](#_Toc67935174)

[2.5 Auswertung der Bilddaten 19](#_Toc67935175)

[2.6 Kommunikation mit Simulation / Visualisierung 20](#_Toc67935176)

[3 Visualisierung und Simulation 21](#_Toc67935177)

[3.1 Übersicht Softwarearchitektur 21](#_Toc67935178)

[3.1.1 Blockschaltbild 21](#_Toc67935179)

[3.1.2 GUI 21](#_Toc67935180)

[3.1.3 Engine 21](#_Toc67935181)

[3.1.4 Client 21](#_Toc67935182)

[3.1.5 Modelle, Texturen, Shader 22](#_Toc67935183)

[3.1.6 Path Generierung 22](#_Toc67935184)

[3.2 Graphical User Interface 22](#_Toc67935185)

[3.3 Aufbau der Engine 22](#_Toc67935186)

[3.4 Erstellung von 3D-Modellen 22](#_Toc67935187)

[3.5 Bewegungen im Dreidimensionalen Raum 22](#_Toc67935188)

[3.6 Auswertung von erhaltenen Positionsdaten 22](#_Toc67935189)

[3.7 Übertragung zu Swarm Controll 22](#_Toc67935190)

[5 Fahrzeug Software 23](#_Toc67935191)

[5.1 Übersicht der Architektur 23](#_Toc67935192)

[5.2 Kommunikation mit Swarm Controll 23](#_Toc67935193)

[5.2.1 Kommunikationsprotokoll 23](#_Toc67935194)

[5.3 Hardware Ansteuerung 24](#_Toc67935195)

[5.3.1 LED-Ansteuerung 24](#_Toc67935196)

[5.3.2 Kommunikationsmodule 24](#_Toc67935197)

[5.3.2.1 DAP Link 24](#_Toc67935198)

[5.3.2.2 Bluetooth 24](#_Toc67935199)

[5.3.2.3 WLAN 24](#_Toc67935200)

[5.4 TCP Port der Swarm Control 24](#_Toc67935201)

[5.4.1 Motor Ansteuerung 24](#_Toc67935202)

[6 Fahrzeug Software 26](#_Toc67935203)

[6.1 Übersicht der Architektur 26](#_Toc67935204)

[6.2 Kommunikation mit Swarm Controll 26](#_Toc67935205)

[6.2.1 Kommunikationsprotokoll 26](#_Toc67935206)

[6.3 Hardware Ansteuerung 26](#_Toc67935207)

[6.3.1 LED-Ansteuerung 26](#_Toc67935208)

[6.3.2 Kommunikationsmodule 26](#_Toc67935209)

[6.3.3 Bluetooth 26](#_Toc67935210)

[6.3.4 WLAN 26](#_Toc67935211)

[6.3.5 Motor Ansteuerung 26](#_Toc67935212)

[7 Ergebnisse 27](#_Toc67935213)

[7.1 Funktionalität Positionserkennung 27](#_Toc67935214)

[7.2 Steuerung der Fahrzeuge 27](#_Toc67935215)

[7.3 Simulationstest mit Pseudodaten 27](#_Toc67935216)

[7.4 Steuersoftwarte Funktionalitätstest 27](#_Toc67935217)

[7.5 Schwarmbewegung 27](#_Toc67935218)

[7.5.1 Kreis 27](#_Toc67935219)

[7.5.2 Zick-Zack 27](#_Toc67935220)

[8 Anhang 28](#_Toc67935221)

[8.1 Inbetriebnahme (f. 4Klasse TdoT) 28](#_Toc67935222)

[8.2 Projektmanagement 28](#_Toc67935223)

[8.3 Projektplan 28](#_Toc67935224)

[8.4 Projekttagebuch 28](#_Toc67935225)

[8.5 Projektkosten 28](#_Toc67935226)

[9 Quellenverzeichnis 29](#_Toc67935227)

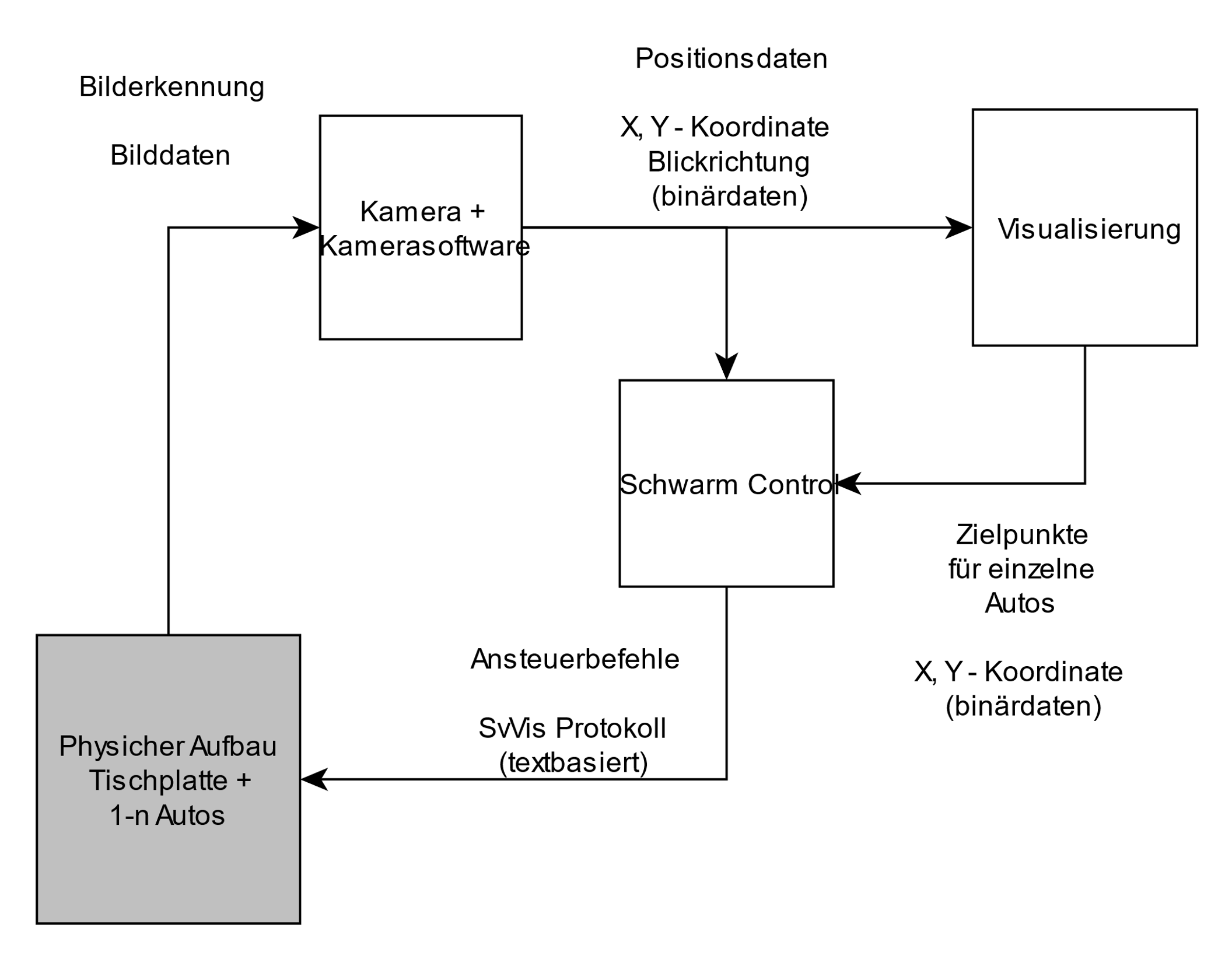
[9.1 Bücher 29](#_Toc67935228)

[9.2 Onlinemedien 29](#_Toc67935229)

[9.3 Zeitschriften 29](#_Toc67935230)

# Camera Controlled Swarm Robots

## Blockschaltbild

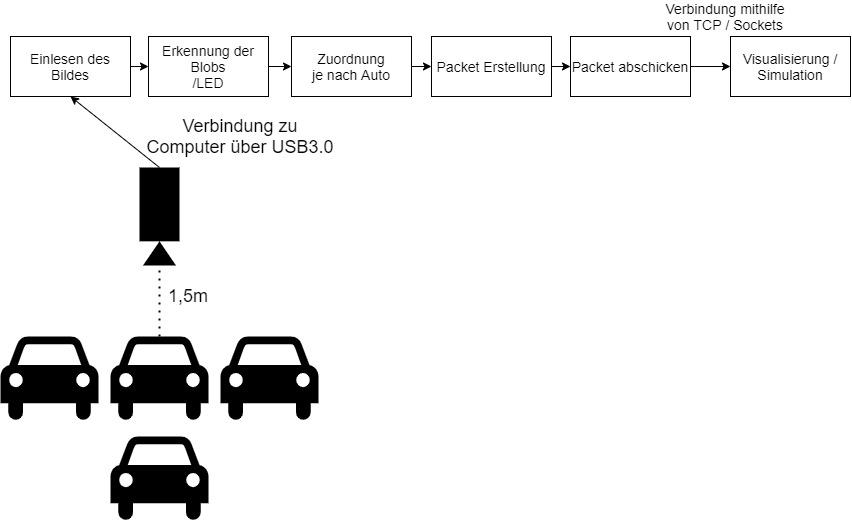


## Funktionsbeschreibung

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 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 and die Autos, welche entsprechend der Fahrkommandos auf der Tischplatte kontrolliert fahren.

# Positionserkennung

## Übersicht

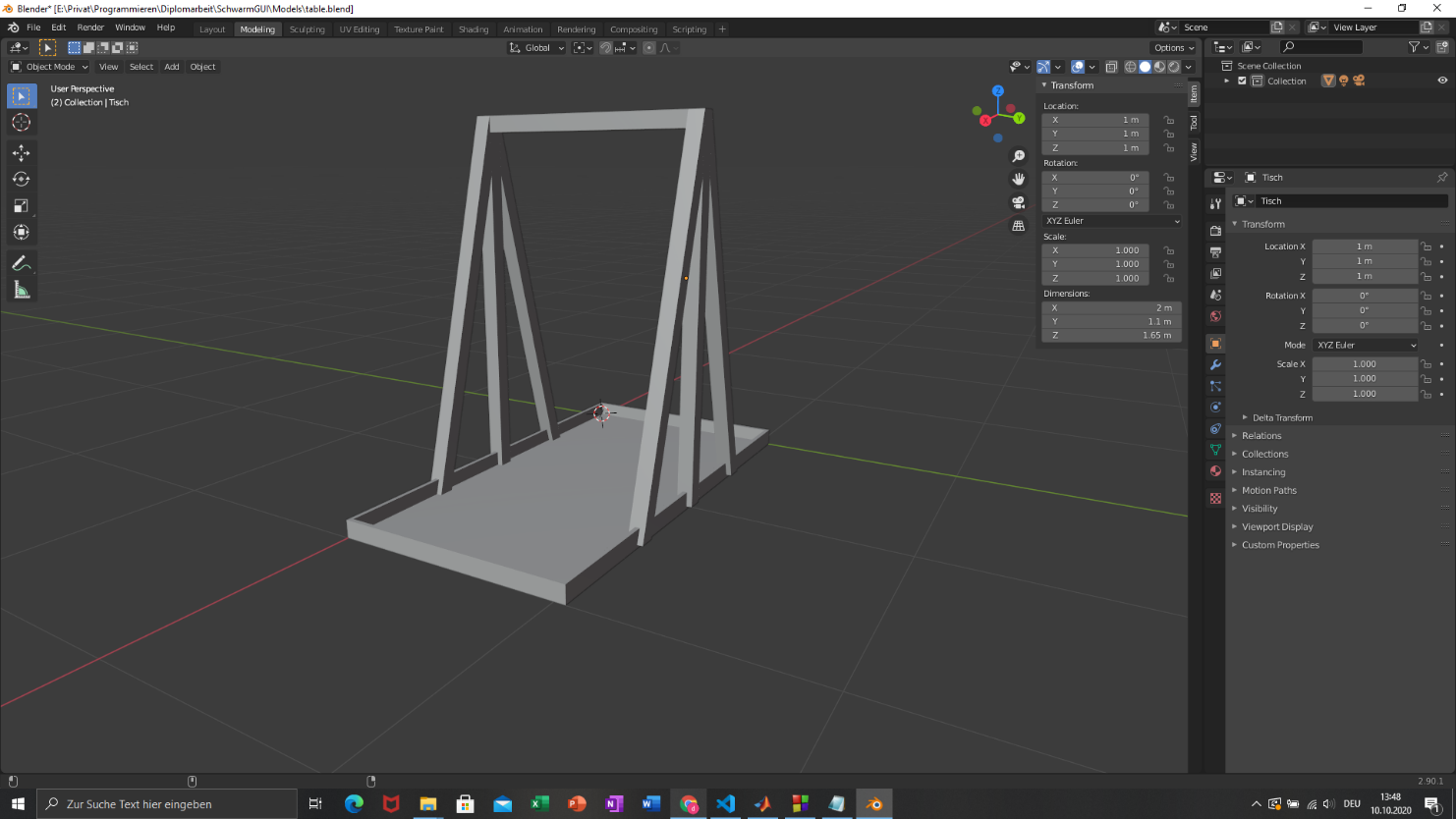


Es werden mehrere Autos mithilfe einer Kamer die sich 1,5m über ihnen befindet. erfasst. Diese Autos sind jeweils mit mehreren Positions LEDs ausgestattet. Die Kamera sendet die Bilddaten über ein USB 3.0 Kabel an einen Computer. Die Software wertet die Bilddaten aus und erkennt die LEDs (Blobs) diese 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.

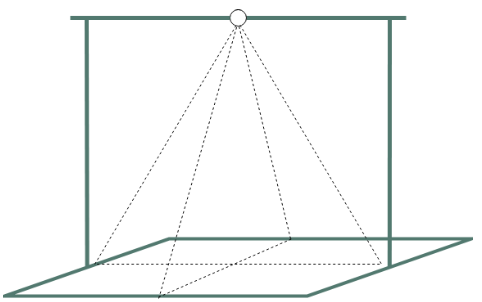
Die Software wurde sowohl in Python als auch in C++ programmiert.

## 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.



### Korrekter Aufbau



Es muss 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.

## Kamera

Bei der verwendeten Kamera handelt es sich um eine „DFK 33UX273 USB 3.0 Farb-Industriekamera“. Diese ist mit einem 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.

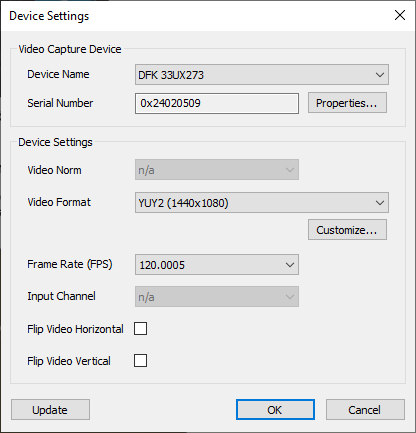


### Kalibrierung der Kamera

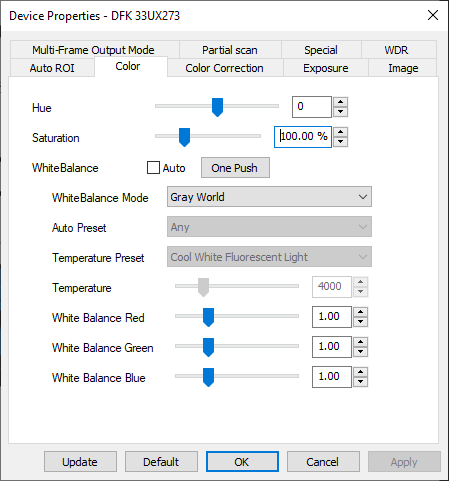
Die Kamera kann auf mehreren Wegen kalibriert werden. Im Programmcode als Befehlszeile oder in einer GUI (Graphical User Interface)

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.

## 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.

### 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.

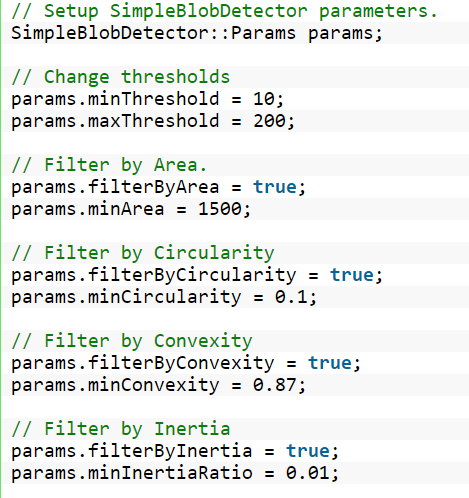
Gruppierung von Zentren aus mehreren Binärbildern anhand ihrer Koordinaten. Nahe Zentren bilden eine Gruppe, die einem Blob entspricht, der vom Parameter minDistBetweenBlobs gesteuert wird.

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

#### Blob Detection

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

##### Beispielparameter



* Threshold = Konvertierung des Quellenbildes in mehrere binäre Bilder. Dieses funktioniert, indem man das Bild in mehrere Verschiedene Bilder mit einen 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.

#### 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 beschrieben.

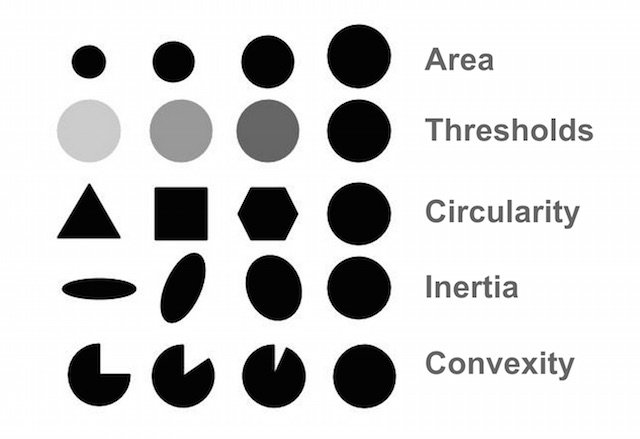
* **Konvexität (Convexity)**

Konvexität ist definiert als . Convex Hull ist die engste Konvexe Form, die die Form komplett umschließt.

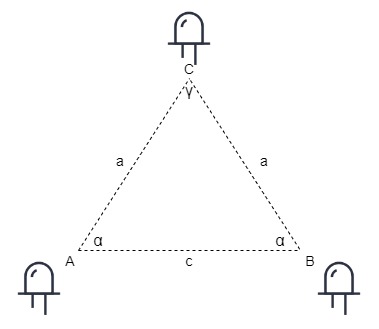
* **Trägheitsverhältnis (Inertia)**

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.



### Berechnung der Position



Um die Seitenlängen zu bestimmen wurde einmal als Referenz a & c in Pixel gemessen.

Es wird weiteres als Kontrolle der Winkel α & γ berechnet.

Darüber hinaus wird in regelmäßigen Abständen die Seitenlänge neu berechnet. Um sicherzustellen, dass es sich noch immer um dasselbe Auto handelt.

### 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.

## 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

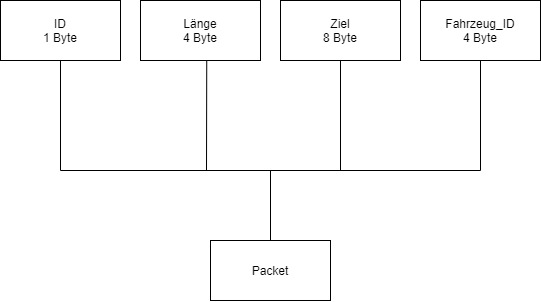
    {

    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);

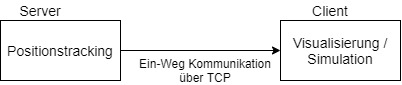
    packet.allocate(packet.min\_size());

    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.

## 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.

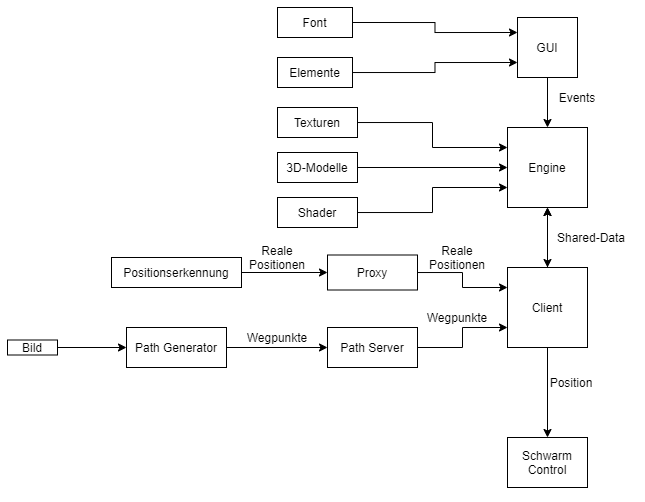
Die Pakete werden wie im Kapitel [2.5 Auswertung der Bilddaten](#_Auswertung_der_Bilddaten) zusammengebaut, encodiert und gesendet.

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

# Visualisierung und Simulation

## Übersicht Softwarearchitektur

### Blockschaltbild



### 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).

### Engine

Die Engine ist das Rückardt 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.

### 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.

### 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).

### 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 wir anschließend vom „Path Generator“ eingelesen und in Wegpunkte untereilt, 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.

## Graphical User Interface

## Aufbau der Engine

## Erstellung von 3D-Modellen

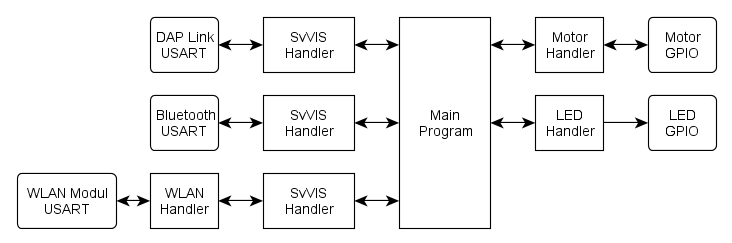
## Bewegungen im Dreidimensionalen Raum

## Auswertung von erhaltenen Positionsdaten

## Übertragung zu Swarm Controll

# Fahrzeug Software

## Übersicht der Architektur



## 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

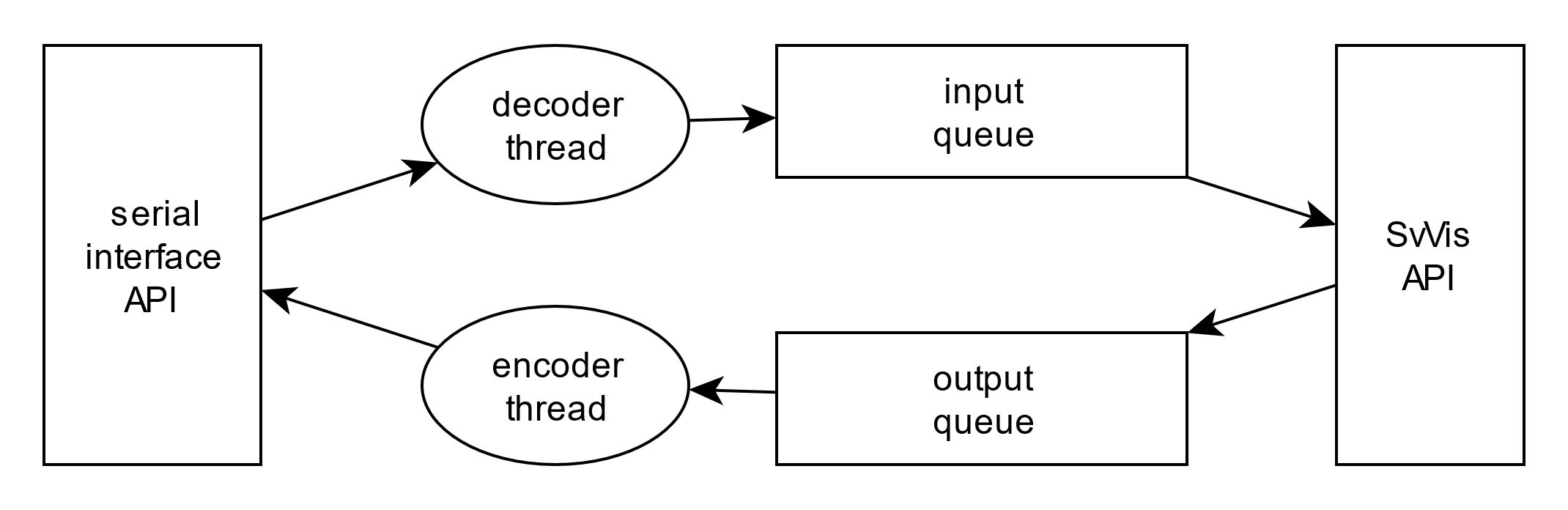
Da die Kommunikation über den DAP-Link Adapter nur für Entwicklungs- und Testzwecke verwendet werden kann, wird die Kommunikation im Endaufbau über Bluetooth oder WLAN stattfinden.

### 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 |

#### Übersicht Software-Architektur des SvVis-Dekodierers



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.

#### 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 undverarbeitet 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 vorgegebn.

#### 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.

## Hardware Ansteuerung

### LED-Ansteuerung

Die Ansteuerung der LEDs ist die simpelste Hardware-Ansteuerung des Fahrzeuges. 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 Positionserkennungsleds einzel ein- und ausschalten.

### Kommunikationsmodule

#### Allgemein

Der Decoder greift über eine abstrakte Interface-Klasse auf die Daten zu. Diese abstrakte Klasse wird verendet, 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 ab geerbten Klasse aufgerufen werden. Dadurch ist es möglich, mit einer Implementierung der Decodierung / Encodierung mehrere verschieden anzusprechende Schnittstellen zu verwenden. (z.B. USART direkt / AT-Kommandos über USART)

#### DAP Link

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

#### 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

#### 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

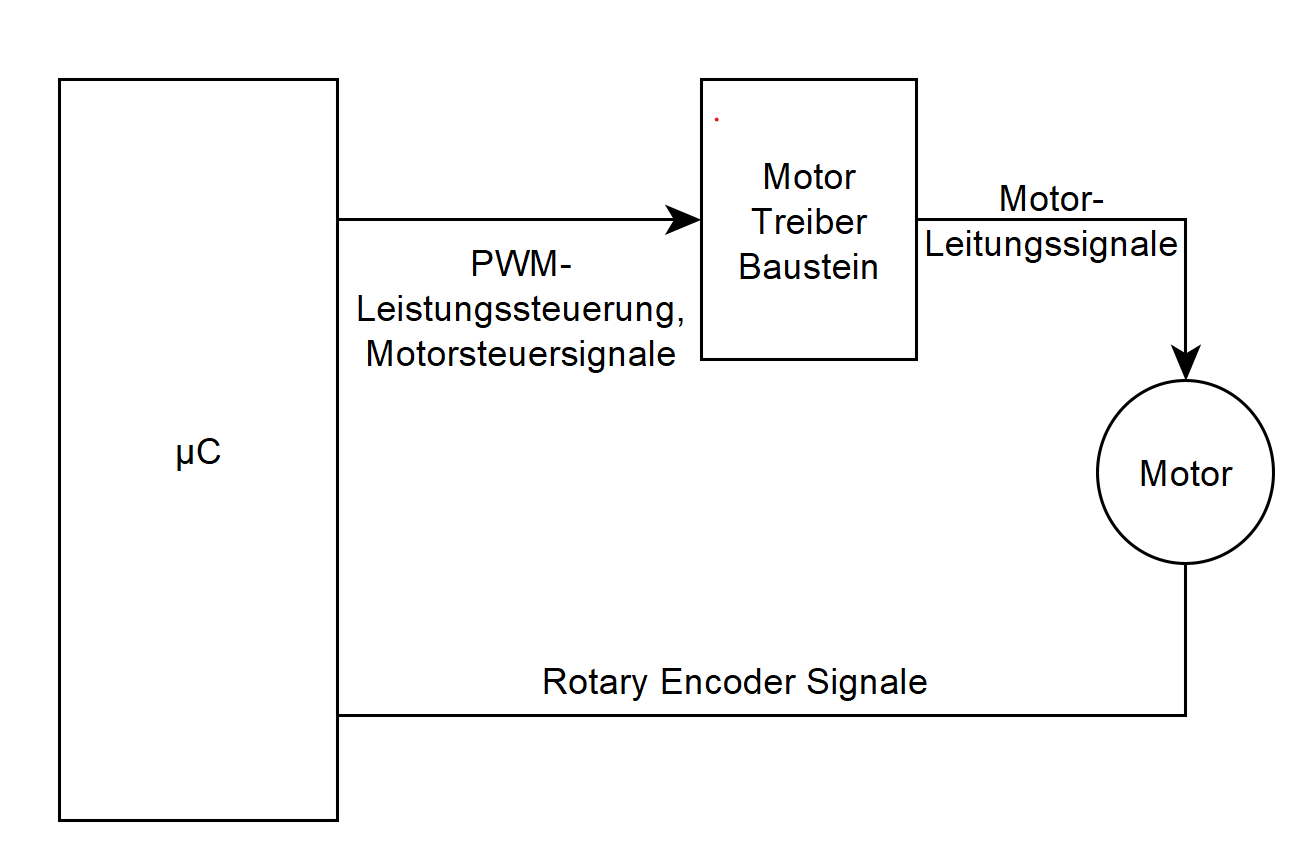
Die Kommunikation erfolgt über AT-Kommandos über eine USART-Schnittstelle.

Da das WLAN Modul die AT-Kommandos benötigt, wurde die abstrakte interface-klasse verwendet.

### Motor Ansteuerung

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

# Ergebnisse

## Funktionalität Positionserkennung

## Steuerung der Fahrzeuge

## Simulationstest mit Pseudodaten

## Steuersoftwarte Funktionalitätstest

## Schwarmbewegung

### Kreis

### Zick-Zack

# Anhang

## Inbetriebnahme (f. 4Klasse TdoT)

## Projektmanagement

## Projektplan

## Projekttagebuch

## Projektkosten

# Quellenverzeichnis

## Bücher

## Onlinemedien

## Zeitschriften