

Bachelor-Thesis an der Hochschule Luzern – Technik & Architektur

Titel «variable-rate Application» System für das SBB Heisswasser Spritzfahrzeug

Diplomandin/Diplomand Düggele Jonas

Bachelor-Studiengang Bachelor Maschinentechnik

Semester FS21

Dozentin/Dozent Lanter Joshua

Expertin/Experte Dubach Roger

Abstract Deutsch

Diese Bachelor-Thesis beschäftigt sich mit der «variable-rate Application» und deren Einsatz auf den Gleisen. Der Industriepartner ist die SBB. Für sie wurde das OpenSourceprojekt AgOpenGPS analysiert und bewertet. Die Thesis vertieft sich in der Analyse der Software, die «variable-rate Application» beinhaltet. Dabei wird herausgearbeitet, ob die Software einsetzbar ist und wo Verbesserungspotential besteht. Um dies systematisch zu machen, wird mit Lösungskonzepten und in unterschiedlichen Phasen gearbeitet. Die Lösungskonzepte bewegen sich zwischen «Software selbst neu programmieren» und «Software unverändert einsetzen». In der ersten Phase wird die Software unverändert beurteilt. In der zweiten Phase wird der Quellcode analysiert. Resultierend aus den Erkenntnissen aus Phase eins und zwei wird der SBB empfohlen, eine eigene Software zu schreiben. In der dritten Phase wird das Laden einer Karte genauer analysiert, um weitere Erkenntnisse für die eigene Software zu erhalten. Am Ende befindet sich ein Ausblick für die Zukunft mit weiteren Erkenntnissen.

Abstract Englisch

This bachelor thesis deals with the "variable-rate application" and its use on the tracks. The industry partner is the SBB. The open source project AgOpenGPS was analysed and evaluated for them. The thesis delves into the analysis of the software that contains the "variable-rate application". It will be worked out whether the software can be used and where there is potential for improvement. In order to do this systematically, we work with solution concepts and in different phases. The solution concepts range between "reprogramming the software yourself" and "using the software unchanged". In the first phase, the software is assessed unchanged. In the second phase, the source code is analysed. As a result of the findings from phases one and two, SBB is recommended to write its own software. In the third phase, the loading of a map is analysed in more detail to gain further insights for the own software. At the end, there is an outlook for the future with further insights.

Inhaltsverzeichnis

1	Einführung in den Projektauftrag.....	3
1.1	Kurzportrait des SBB Heisswasserspritzzuges.....	3
1.2	Problemstellung.....	4
1.3	Projektziele der SBB.....	5
1.4	Projektziele parallellaufender Bachelor-Thesen.....	5
1.4.1	Ventilansteuerung.....	5
1.4.2	Testfahrzeug.....	5
1.5	Projektziele dieser Bachelor-Thesis.....	6
2	Projektmanagement.....	7
2.1	Prinzip.....	7
2.2	Lösungsprinzipien.....	7
3	Erste Phase.....	8
3.1	Grafische Oberfläche.....	8
3.2	Erste Analyse.....	10
4	Zweite Phase.....	11
4.1	Positionsverarbeitung.....	11
4.2	Karteninterpretation.....	12
4.3	Berechnungsroutine.....	13
4.4	Variablen.....	15
4.5	Signalausgabe.....	16
5	Fazit.....	16
5.1	Lösungsprinzip 1.....	17
5.2	Lösungsprinzip 2.....	18
5.3	Lösungsprinzip 3.....	19
6	Dritte Phase.....	20
6.1	Karten Laden.....	20
7	Ausblick.....	23
7.1	Angepasste Software.....	23
7.2	Plattform.....	23
7.3	Schnittstelle.....	23
7.4	Datenbewältigung.....	24
7.5	Sicherheit.....	25
8	Reflexion.....	25
9	Glossar.....	26

10	Abbildungsverzeichnis.....	27
11	Tabellenverzeichnis	27
12	Quellenverzeichnis.....	28
13	Anhang.....	29
13.1	Grafische Oberflächen AgOpenGPS	29
13.1.1	Geräte Einstellungen	29
13.1.2	Fahrzeugeinstellungen.....	32
13.1.3	Simulationskoordinaten.....	34
13.2	NMEA Sätze	35
13.2.1	GPGGA	35
13.2.2	GPVTG.....	36
13.2.3	GPRMC.....	36
13.2.4	GPHDT.....	36
13.2.5	PAOGI.....	37
13.2.6	PTNL.....	37
13.2.7	GNTRA	37
13.2.8	PSTI.....	38
13.3	Datensatz Zeitmessungen Karten Laden.....	39

1 Einführung in den Projektauftrag

In diesem Kapitel wird aufgezeigt, was die Ziele dieser Arbeit sind. Zuerst wird der Heisswasserspritzzug (HWZ) der Schweizerischen Bundesbahn (SBB) vorgestellt. Um diesen HWZ dreht sich die ganze Arbeit. Danach wird die Problemstellung, mit welcher sich die SBB aktuell befasst, erläutert. Anhand der Problemstellung werden die Projektziele der SBB und speziell die Ziele dieser Arbeit beschrieben. Weiter werden in diesem Kapitel die Projektgrenzen dieser Arbeit definiert. Dieses Kapitel wurde zusammen mit Roger Brast erarbeitet. Er und Stig Segat arbeiten parallel an dem Projekt in Form einer Bachelor-Thesis.

1.1 Kurzportrait des SBB Heisswasserspritzzuges

Die Schweiz besitzt eines der dichtesten Schienennetze der Welt. Der größte Teil dieses Netzes wird von der SBB selbst unterhalten. Zum Unterhalt gehört eine regelmässige Bekämpfung des Unkrautes. Würden Pflanzen auf dem Eisenbahnnetz ungehindert wachsen, könnte ein sicherer Betrieb nicht gewährleistet werden.

Um den Pflanzenbewuchs zu bekämpfen, setzt die SBB bis jetzt das Pflanzenschutzmittel Glyphosat ein. Um das Glyphosat aufzutragen, müssen momentan Angestellte den Schienen entlanglaufen und manuell mittels Spritztanks auftragen. Diese Arbeitsweise nimmt viel Zeit in Anspruch, ist jedoch präzise und zuverlässig.

In Zukunft möchte die SBB auf Glyphosat zur Bekämpfung von Pflanzen verzichten. Unter anderem weil Glyphosat in der Politik heiss diskutiert und in der Zukunft wohl verboten wird. Darum hat die SBB verschiedene Möglichkeiten betrachtet, wie verhindert werden kann, dass Pflanzen die Bahnstrecken bewachsen. Eine der Möglichkeiten ist der Einsatz eines HWZ, welcher die Pflanzen mit heissem Wasser verbrüht. Die SBB hat bereits einen Prototyp eines HWZ entwickelt und gebaut. Erste Tests waren sehr vielversprechend. Durch den Prototyp konnten wichtige Erkenntnisse gewonnen werden, wie ein solcher Zug eingesetzt werden kann.



Abbildung 1 Prinzip Bild des Heisswasserspritzzug

Der Zug besteht aus einer Zuglokomotive sowie vier Wagen für die Unkrautbekämpfung. Die zwei Wagen in der Mitte sind die Kesselwagen. In diesen Kesselwagen wird das Wasser auf 95°C erhitzt. Auf dem ersten und vierten Wagen, den sogenannten Spritzwagen, sind die Spritzdüsen angebracht. Die Anordnung der Wagen ist für eine möglichst gute Sprühdauer ausgelegt und kann nicht verändert werden. Der Abstand des ersten Balkens zum achten Balken wurde so gross wie möglich gewählt. Dieser Abstand garantiert auch bei hoher

Geschwindigkeit, dass die Pflanzen dem Heisswasser genug lange ausgesetzt sind. Für eine effektive Bekämpfung muss die Pflanze zwischen drei bis zehn Sekunden 95°C heissem Wasser ausgesetzt werden.

1.2 Problemstellung

Die SBB führte mit ihrem Prototyp des HWZ diverse Tests durch. Dabei stellte sich heraus, dass die Sensoren nicht ideal sind. Die Sensoren erkennen nicht nur die schädlichen Pflanzen, sondern auch andere Pflanzen wie Moos, welches nicht bekämpft werden muss. In gewissen Bereichen der Gleise können die Sensoren Pflanzen nur bedingt erkennen. Da sich die Sensoren auf dem gleichen Fahrzeug wie die Heisswasserdüsen befinden, limitieren sie die maximale Geschwindigkeit des Fahrzeuges auf 40km/h. Dies ist die maximale Geschwindigkeit, bei welcher die Pflanzen noch besprüht werden können. Der Grund dafür liegt in der Distanz von den Sensoren zum ersten Balken. So braucht es zu lange, bis die Sensoren ausgewertet und die entsprechenden Signale ausgegeben haben.

Deshalb sucht die SBB nach neuen Möglichkeiten, wie das Fahrzeug und die Pflanzenbekämpfung verbessert werden können. Ein vielversprechender Ansatz ist die Detektion der Pflanzen vor einem Sprüheinsatz. Dazu können z.B. Drohnen und künstliche Intelligenz verwendet werden. Die vor dem Einsatz gewonnenen Daten könnten dem Fahrzeug z. B. als Geoinformationssystem (GIS) zur Verfügung gestellt werden.

Dieses Vorgehen zur Pflanzendetektion bietet wesentliche Vorteile gegenüber der bisherigen Lösung mit Sensoren. Der HWZ weiss im Voraus, in welchem Streckenabschnitt sich Pflanzen befinden, und kann direkt zu diesen Abschnitten fahren. Die Fahrtrichtung des HWZ spielt keine Rolle, da keine fixen Sensoren benutzt werden. Auch die Dauer, welche der HWZ für eine Strecke benötigt, kann so vorherbestimmt werden. Dies ermöglicht eine genaue Planung des Einsatzes. Weiter bietet dieses System viele Möglichkeiten für zukünftige Erweiterungen. Die künstliche Intelligenz könnte z.B. schädliche Pflanzen von nicht schädlichen Pflanzen unterscheiden und den Sprüheinsatz weiter optimieren.

Die Problemstellung der SBB ist nun, wie ein System, welches Pflanzen vorgängig detektiert, in den bestehenden Prototypen integriert werden kann und welche technischen und physikalischen Limiten ein solches System haben muss.

Weiter stellt sich die SBB die Frage, wie effizient die Berechnungen und die Software zur Pflanzenbekämpfung umgesetzt wurden.

Die SBB startete ein Projekt, um diese Fragen zu klären. Einen Teil der Aufgaben übergab die SBB der Hochschule Luzern Technik und Architektur. Studierende sollen im Rahmen einer Bachelor-Thesis mithelfen, neue Erkenntnisse zu gewinnen und Konzepte auszuarbeiten, um den Prototypen zu verbessern und zu erweitern.

1.3 Projektziele der SBB

Dieses Kapitel ist rein informativ und soll dem Leser einen Überblick über das gesamte Projekt der SBB geben. Die Punkte in diesem Kapitel sind nicht Bestandteil dieser Bachelor-Thesis.

Der aktuelle Prototyp zur Pflanzenbekämpfung auf dem Schienennetz der SBB soll erweitert werden. Die Detektion der Pflanzen mit Sensoren soll durch ein GIS-Karte ersetzt werden. Für den nächsten Ausbau des Prototyps hat die SBB folgende Ziele definiert:

- Erweiterung der Spritzsystemsteuerung mit einer GIS- und GPS-Daten basierten Spritzdüsensteuerung („variable-rate Application“). Das heisst, die Pflanzen werden z.B. durch Drohnen und künstliche Intelligenz vor einem Spritzeinsatz detektiert und als GIS Datensätze kartiert. Anhand dieser GIS Datensätze und der Fahrzeugposition sollen dann die Spritzdüsen bei Überfahrt über die zu behandelnden (Pflanzen-) Flächen geöffnet werden.
- So wenig heisses Wasser wie möglich einsetzen. Um Energie zu sparen und den Prozess zu verbessern, dürfen die Düsen nur so kurz wie notwendig angesteuert werden.
- Eine möglichst hohe Durchschnittsgeschwindigkeit erreichen. Das Ziel ist es, die Geschwindigkeit vom Streckenabschnitt abhängig zu machen. Abschnitte ohne Bewuchs sollen so schnell wie möglich abgefahren werden. Strecken mit Bewuchs, mit der maximal möglichen Geschwindigkeit, welches das Sprühsystem zulässt. Dabei wird das Ziel der effektiven Pflanzenbekämpfung über die maximal mögliche Fahrgeschwindigkeit gesetzt. Dieses Ziel kann durch eine vorgängige Detektion der Pflanzen z.B. mit Drohnen erreicht werden.

1.4 Projektziele parallellaufender Bachelor-Thesen

Parallel zu meiner Arbeit werden von zwei andern Kommilitonen Bachelor-Thesen geschrieben, die sich mit demselben Projekt befassen. Sie bewegen sich jedoch in einem anderen Themenbereich.

1.4.1 Ventilansteuerung

Roger Brast befasst sich in seiner Bachelor-Thesis mit der Ventilansteuerung. Diese Ansteuerung läuft auf der SPS. Der Kontaktpunkt zu meiner Arbeit liegt in der Schnittstelle zwischen AgOpenGPS und der SPS.

1.4.2 Testfahrzeug

Stig Segat befasst sich in seiner Bachelor-Thesis mit der Konstruktion eines Testfahrzeuges. Dieses soll unabhängig von den Gleisen Konzepte veranschaulichen und verifizieren können. Im Verlaufe des Projektes der SBB soll dort die SPS und die GIS Software für den HWZ überprüft und validiert werden können.

1.5 Projektziele dieser Bachelor-Thesis

Wie in den vorherigen Kapiteln erwähnt, will die SBB die Sensoren ersetzen durch ein System, welches aus GIS Datensätzen und einer Positionierung besteht. Um dies möglichst günstig zu erreichen, suchten sie nach einem existierenden Programm. Dabei haben sie das open source Projekt AgOpenGPS gefunden. AgOpenGPS besteht aus einer Software und einem Arduino, welcher mit mehreren Hardwarekomponenten erweitert werden kann. Die Hardwarekomponenten werden in dieser Bachelor-Thesis nicht analysiert. Für die Anwendung auf der Schiene spielen sie auch keine grosse Rolle. Man kann die Signale der Software auch direkt an die SPS des Zuges schicken. Die Analyse dreht sich nur um die Windows basierte Software. Die Software ist in C# geschrieben und wird normalerweise auf einem Tablet installiert. Das ganze Projekt wird von Bauern entwickelt, um ihre Traktoren zu automatisieren. Es beinhaltet unter anderem die Option für variable-rate Application (VRA), an welcher die SBB sehr interessiert ist. Da das Programm jedoch hauptsächlich für die Spurführung entwickelt worden ist, stellt sich nun für die SBB die Frage, ob die Software auch für die Schienen geeignet ist. Dabei sind folgende Punkte abzuklären:

- ➔ Ist die Software auf den Schienen einsetzbar?
Im Unterschied zu einem Feld sind die Konturen der Gleise viel unförmiger. Die meisten Felder sind grosse Flächen, bei welchen alles bearbeitet werden muss. Die Gleise sind eher lange Pfade, bei welchen nur sehr gezielt bearbeitet werden muss.
- ➔ Muss am Quellcode etwas verändert werden?
Der SBB wäre es lieber, wenn nichts am Quellcode verändert werden müsste. Dies begründet sie mit den damit verbundenen Kosten. Bei einer Veränderung im Quellcode muss man sich für eine von zwei Optionen entscheiden. Entweder passt man jedes Update an, um von den Veränderungen zu profitieren die, das Update bringt. Oder man behält die Version, bei welcher man die Veränderungen vorgenommen hat.
- ➔ Wenn am Quellcode etwas verändert wird: Wie wird dies umgesetzt?
Auch hier bieten sich zwei Varianten an. Die erste ist das Herauskopieren der notwendigen Zeilen. Dabei erhält man ein Programm, welches auf seine Wünsche zugeschnitten ist. Die zweite Variante ist das Verändern der gewünschten Zeilen. Hierbei erhält man ein Programm, welches auf die Wünsche der SBB zugeschnitten ist, aber auch die anderen Funktionen behält.

Das Projekt wird iterativ und agil geführt. Da es sich um eine analytische Arbeit handelt, kann das Projekt nicht statisch geplant werden. Durch neu gewonnene Erkenntnisse können sich weitere Fragen ergeben, die Klärungsbedarf haben. Dies ist vom Auftraggeber so gewünscht.

2 Projektmanagement

2.1 Prinzip

In dieser Arbeit wird Iterativ gearbeitet. Die SBB will einschätzen, ob sie AgOpenGPS auf ihrem HWZ einsetzen können. Das heisst bei dieser Arbeit handelt es sich um eine Analysearbeit. Da bei einer Analyse nicht vorher klar ist, was man alles findet und wo die Probleme liegen, wurde kein Zeitplan erstellt. Stattdessen wurde bei jeder Phase ein wenig tiefer in den Quellcode geschaut. Als Erstes wurden die Lösungsprinzipien erarbeitet. Danach wurde die Software das erste Mal analysiert, ohne den Quellcode anzuschauen. Dabei wurde analysiert, welche Funktionen die Software bereits hat und welche man auf dem Gleis auch brauchen kann. Bei der zweiten Phase wurde dann in den Quellcode geschaut und dessen Ablauf festgestellt. Um den Überblick zu behalten, wurden vorgängig die erwarteten Blöcke in einer Zustandsmaschine dargestellt. Bei der dritten Iteration konzentrierte man sich auf wichtige Details.

2.2 Lösungsprinzipien

Um die Arbeit iterativ zu analysieren, wurden zuerst drei Lösungsprinzipien erarbeitet. Dies hilft dabei zu entscheiden, wie wichtig die Erkenntnisse sind bei der Analyse. Bei den Lösungsprinzipien ist die Abgrenzung fließend. Die Lösungsprinzipien unterscheiden sich im eigenen Programmieraufwand. Dabei gibt es zwei extreme und einen Kompromiss der Lösungsprinzipien. Das erste extreme Lösungsprinzip ist die Übernahme des Programmes ohne jegliche Veränderung. Die Frage dafür ist, ob das Programm ohne Veränderung überhaupt einsetzbar ist oder nicht. Wenn es einsetzbar ist, fragt sich noch, welche Kompromisse man eingehen muss. Das zweite extreme Lösungsprinzip ist die komplette Neu-Programmierung der Software. Dabei entsteht eine Software, die alle gewünschte Funktionen erfüllt. Es entsteht jedoch ein erheblicher Mehraufwand, da man die Programmierung selbst machen muss. Das letzte Lösungsprinzip ist ein Kompromiss der ersten beiden Lösungsprinzipien. Dabei ist das letzte Lösungsprinzip variabel. Man kann entweder den aktuellen Quellcode so abändern, dass er alle Funktionen beinhaltet, die man braucht. Oder man kann eine eigene Grundlage programmieren und die wichtigen Funktionen, die das AgOpenGPS beinhaltet, hineinkopieren. Entscheidend dabei ist die Anzahl nützlicher Funktionen, die im AgOpenGPS bereits vorhanden sind und wie diese programmiert sind. Wenn der Quellcode von AgOpenGPS schön geschrieben ist und viele Funktionen bereits wunschgemäß abgedeckt sind, lohnt es sich, den Quellcode wunschgemäß zu erweitern. Ist der Quellcode von AgOpenGPS unübersichtlich und muss noch stark verändert werden, lohnt es sich, eine eigene Grundlage zu programmieren. In der folgenden Grafik wird diese Situation nochmals dargestellt. Man muss dazu erwähnen, dass das Lösungsprinzip zwei nicht in der Mitte sein muss. Je nach Quellcode liegt das Lösungsprinzip mehr auf der einen oder anderen Seite.

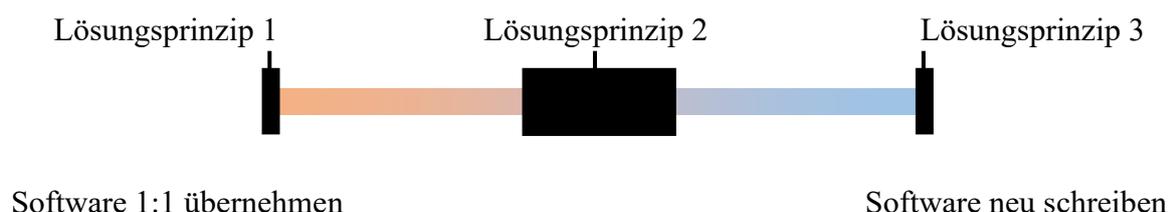


Abbildung 2: Darstellung Lösungsprinzipien

3 Erste Phase

In diesem Kapitel wird die Software unverändert beurteilt. Im ersten Unterkapitel werden die Grundfunktionen herausgehoben, welche auch auf dem Gleis wichtig sind. Im zweiten Unterkapitel wird eine erste Analyse gemacht.

3.1 Grafische Oberfläche

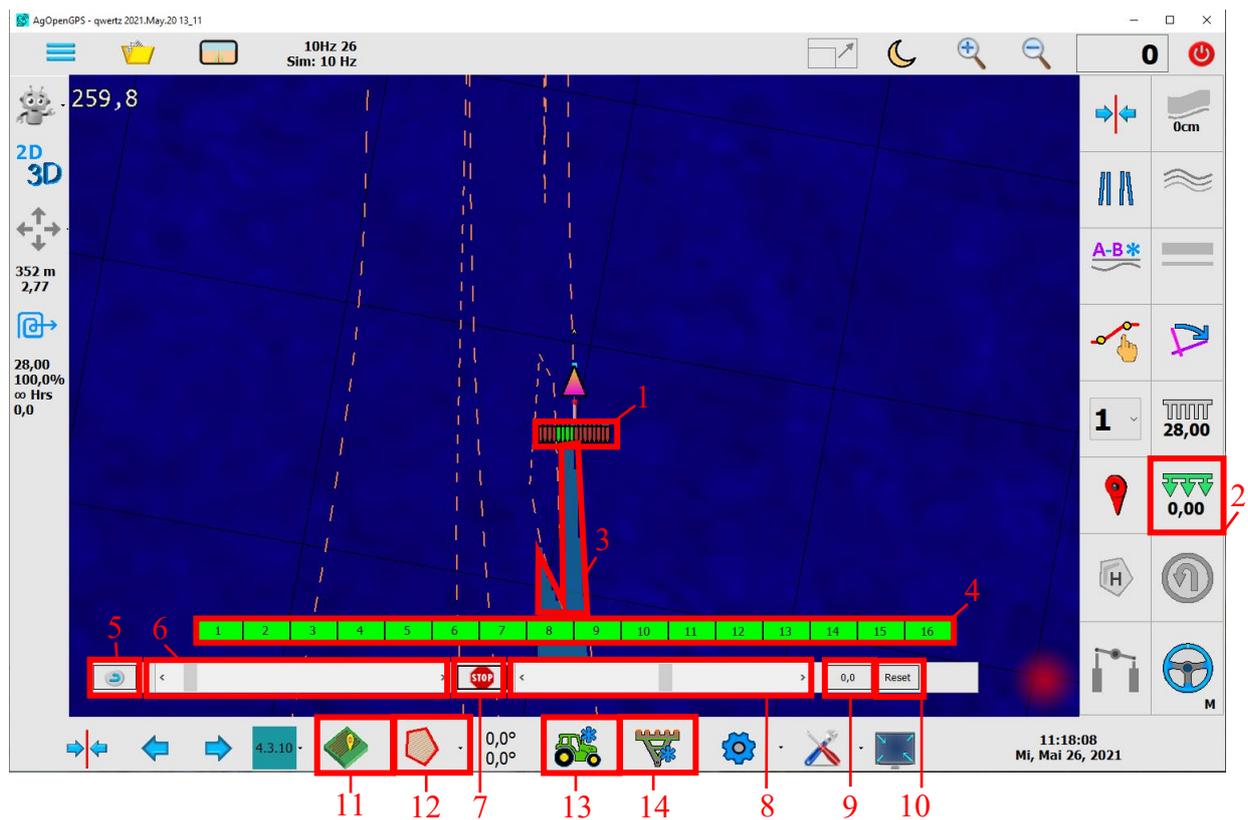


Abbildung 3: Screenshot AgOpenGPS Hauptfenster

Tabelle 1: Kurzbeschreibung Funktionen

Nummer	Beschreibung
1	Liveanzeige aller Sektionen. Die Farbe indiziert die individuellen Status der Sektionen. Rot bedeutet ausgeschaltet, Grün bedeutet eingeschaltet und Gelb bedeutet manuell eingeschaltet.
2	Schaltet alle Sektionen zwischen aus und automatisch um. Im automatischen Betrieb erkennen die Sektionen selbstständig, ob sie sich ein- oder ausschalten müssen. Im Manuellen Betrieb muss man auf die jeweiligen Sektionen (Fläche 4) klicken, um sie ein- oder auszuschalten
3	Die hellblaue Fläche indiziert die Bearbeitung der Fläche. Sie erscheint automatisch hinter den aktiven Sektionen.
4	Modus jeder einzelnen Sektion. Die Felder sind alle einzeln anklickbar und verändern den Modus der Sektion. Dabei verändern sie den Modus wiederholend. Sie wechseln folgendermassen: Von grün (automatisch) zu gelb (manuell an) zu Rot (manuell aus). So kann jede Sektion einzeln übersteuert werden.
5	Kehrt die Ausrichtung des Vehikels (roter Pfeil) um 180°
6	Regelt die simulierte Geschwindigkeit. Dabei kann man auch langsam rückwärtsfahren, wenn man den Schieber nach links verschiebt.
7	Hält das Vehikel sofort an.
8	Simuliert den Lenkwinkel des Vehikels. Der Lenkradius kann in den Optionen «Fahrzeugeinstellungen» geändert werden. Diesen Schieber braucht man hauptsächlich für die simulierte Fahrt.
9	Zeigt den simulierten Lenkwinkel an und setzt diesen bei Anklicken wieder auf 0.
10	Setzt das Vehikel wieder auf die ursprüngliche Position der Simulation zurück.
11	Felder erstellen und Laden. Bevor man Feldgrenzen einlesen kann, muss man ein neues Feld erstellen. Auf diesen Feldern wird dann auch abgespeichert, wo man schon bearbeitet hat und wo nicht.
12	Hier kann man die Konturen des Feldes Laden unter dem Reiter «Schlaggrenze erstellen». So können die vorgefertigten kml-Dateien eingelesen werden
13	Hier findet man die Fahrzeugeinstellungen. Die wichtige Einstellung darin ist die Positionierung der Antenne. Das Menu wird genauer beschrieben im Anhang 12.1.2
14	Hier findet man die Anhängereinstellungen. Die interessanten Einstellungen dabei sind der Vorhalt, die Ausschalt-Verzögerung und die Teilbreiteneinstellungen der Sektionen. Das Menu wird genauer beschrieben im Anhang 12.1.1

3.2 Erste Analyse

Die Software kann bei GitHub heruntergeladen werden. Das ganze Projekt wird als Open-sourceprogramm von verschiedenen Bauern zusammen entwickelt. Die Diskussion über die Weiterentwicklung und über Fehlerbehebungen finden im Forum[1] für AgOpenGPS statt. Neben dem Forum stellen einzelne Bauern auch informative Videos auf Youtube[2] bereit. Eine Gesamtanleitung für die Software steht jedoch nicht zur Verfügung.

Die Installation auf dem Computer geht ohne Probleme von statten. Für den Import der Karte wurde ein Feld angelegt auf Google Earth[3]. Dieses Feld kann problemlos geladen werden. Als Nächstes wird versucht, eine grössere Karte zu öffnen, die aus QGIS[4] (einem GIS-Open-sourceprogramm) als "Keyhole Markup Language" (kml) Datei exportiert wurde. Diese Datei konnte die Software jedoch nicht öffnen. Stattdessen kam von Windows eine Fehlermeldung, dass die Software nicht mehr reagieren würde.

Das Einstellen von Fahrzeug und Anhänger ist einfach. Der Anhänger kann so konfiguriert werden, dass er sechzehn einzelne Sektionen besitzt, welche einzeln angesteuert werden können. Es kann auch ein Vorhalt eingeschaltet werden. Er öffnet die Ventile, bevor die Ventile über der eigentlichen Pflanze sind. Dieser Vorhalt kann in Sekunden eingestellt werden, dies erlaubt die Anpassung an die Geschwindigkeit autonom.

Die VRA funktioniert in der Simulation gut. Die Ventile öffnen, wie gewünscht, sobald man über die Feldgrenzen fährt. Man kann auch im Feld Zonen definieren, welche nicht bespritzt werden sollen. Man kann jedoch nicht viele einzelne Felder, welche nicht zusammenhängen, als ein Feld deklarieren.

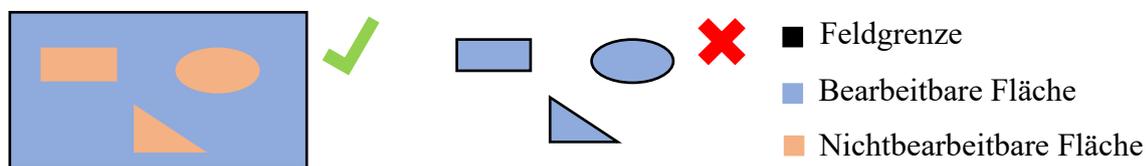


Abbildung 4: Feldgrenzen

Dies ist für die Anwendung auf dem Gleis ein Problem. Beim Bekämpfen des Unkrauts auf den Gleisen hat man sehr viele kleine Felder, welche bearbeitet werden müssen. Eine Lösung dafür wäre die Invertierung des Signales auf der SPS. Dafür muss man nicht nur das Unkraut kartieren, sondern auch die Umrisse der Gleise. So würde der HWZ bei einem Signal «nicht spritzen» spritzen und bei dem Signal «spritzen» nicht spritzen. Es würde funktionieren, kann aber zu Verwirrungen bei den Einstellungen führen. So müsste man dort den Vorhalt bei «turn off ahead» eingeben.

4 Zweite Phase

Um den Quellcode besser analysieren zu können, wurde die Software zuerst als «Statemachine» dargestellt. Diese Darstellung wurde aus logischen Überlegungen gezeichnet. Es wird nicht erwartet, den Code exakt so als Blöcke vorzufinden. Diese Darstellung dient mehr zur Struktur der Analyse. So werden nur die relevanten Stellen im Quellcode gesucht.

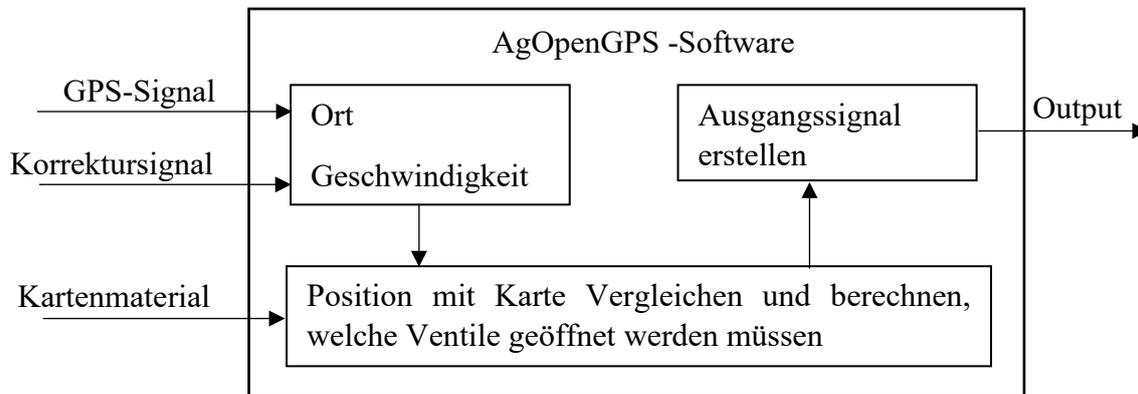


Abbildung 5: Erwartete Zustandsmaschine

Nach dieser Darstellung ordnen sich auch die nachfolgenden Unterkapitel.

4.1 Positionsverarbeitung

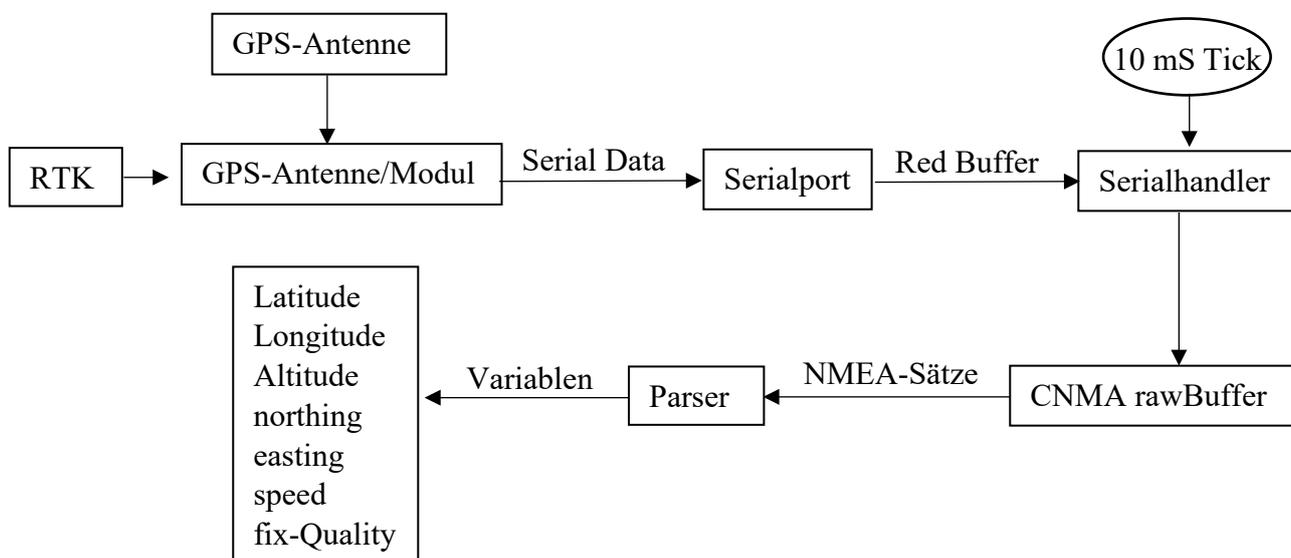


Abbildung 6: Ablauf Signalempfang

Das Standortsignal wird folgendermassen erfasst: Die GPS-Antenne wird an einem GPS-Modul angeschlossen. Wenn man eine genauere Position will, braucht man «real-time Kinematic» (Korrektursignal, kurz RTK). Die RTK soll nach Angaben von Swipos[5] eine Genauigkeit von 2 cm erlauben. Ob diese Präzision geschwindigkeitsabhängig ist, wird nicht erwähnt. Dieses Korrektursignal wird ebenfalls über das GPS Modul verrechnet. Die berechnete Position so wie andere Daten werden dann in Sätzen nach NMEA-Protokoll an den Serialport am Computer geschickt. Diese werden in einem RawBuffer gespeichert. Alle 10 ms wird der Buffer ausgelesen, übersetzt und in den entsprechenden globalen Variablen abgespeichert. Dabei muss

er unter den verschiedenen Sätzen unterscheiden, die geschickt werden können. Diese Sätze werden in ASCII gesendet und können folgende Formate besitzen: GPGGA, GPVTG, GPRMC, GPHDT, PAOGI, PTNL, GNTRA, PSTI. Diese Formate enthalten alle verschiedenen Daten. Teils überschneiden sich diese Informationen. Im Anhang werden diese Sätze einzeln aufgeschlüsselt. Aus dem Programm wird nicht klar, ob man auch alle Formate schicken muss, damit AgOpenGPS funktioniert. Das vorgeschlagene Modul der AgOpenGPS Community ist SimpleRTK2B[6] von ArduSimple und kann so konfiguriert werden, dass nur ausgewählten Sätze geschickt werden.

4.2 Karteninterpretation

Das AgOpenGPS arbeitet stark mit der grafischen Oberfläche der Software. Beim Erstellen eines neuen Feldes oder beim Laden eines alten Feldes werden alle Grenzen auf der Karte eingezeichnet. Die Grenzen und Flächen haben alle spezifische Farben. Wichtig dabei ist der farbliche Unterschied zwischen Grenzen und bereits besprühtem Terrain. Nach diesen Farben wird dann auch gesucht, um zu bestimmen, wann man einzelne Sektionen ein oder ausschalten soll. Die Suche wird in Fahrtrichtung vor den Düsen gemacht bis zu einer vordefinierten Grenze. Diese Grenze kann im Programm selbst bestimmt werden, wie weit geschaut wird. Die Einstellungen dafür befinden sich bei den Einstellungen für den Anhänger (Siehe Anhang 12.1.1 «vorausseilen»). Für die Landwirtschaft bietet es den Vorteil, dass man auf dem Feld kreuz und quer fahren kann. Die Sektionen können mit dieser Methode gut auslesen, wann sie einschalten müssen und wann nicht. So werden Sektionen erst abgeschaltet, wenn über 15% der Fläche innerhalb von 5 Pixeln vor ihnen bereits besprüht wurden. Oder aber eine Feldgrenze zu nahekommt.

4.3 Berechnungsroutine

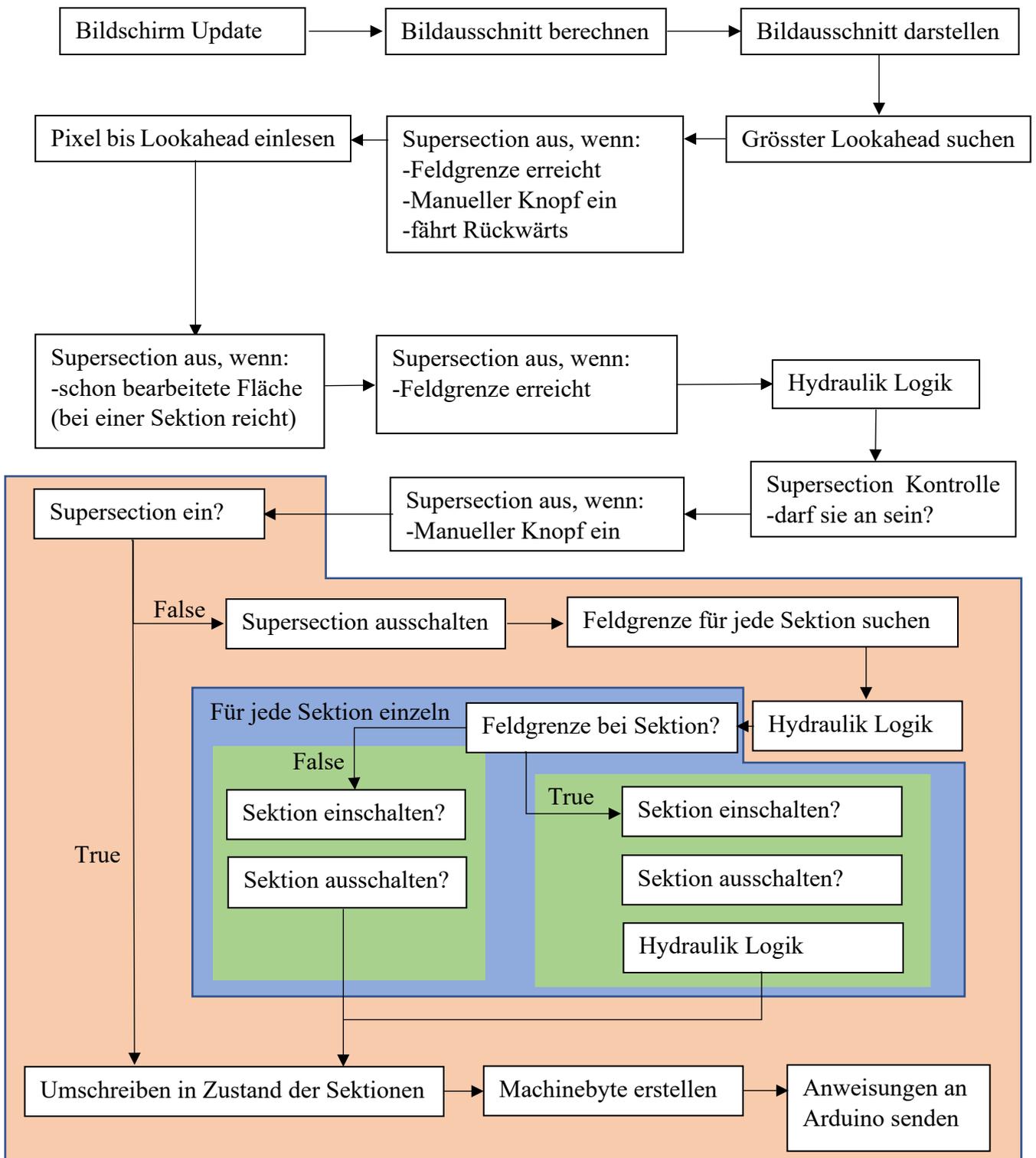


Abbildung 7: Ablauf Hauptroutine

Die Berechnungsroutine ist im Programm AgOpenGPS komplex. Man erkennt gut, dass es sich hier um ein OpenSourceprogramm handelt. Der Code ist sehr organisch erweitert worden und so kommen teils Zeilenblöcke doppelt vor (vgl. Manueller Knopf wird mehrmals abgefragt). Diese Hauptroutine umfasst über 1000 Zeilen und ist somit recht unübersichtlich. Dieser Teil des Programms wird immer aufgerufen, wenn das Programm das Bild aktualisiert. Dies kann durch verschiedene Aktionen geschehen. Ein Beispiel dafür ist die Uhr. Jede Sekunde muss die Zahl geändert werden auf dem Bildschirm für die Uhr. Das Problem dabei ist, dass jegliche Veränderungen diese Routine aufrufen. Da dies nicht regelmässig geschieht, wurde im VisualStudio ein Haltepunkt am Start dieses Abschnittes gesetzt. Danach liess man das Programm 170-mal weiterlaufen. Visual Studio gibt die Laufzeit zwischen den Haltepunkten an. Anhand dessen konnte festgestellt werden, wie schnell die Berechnungen wieder aufgerufen werden. In der folgenden Tabelle sieht man die Resultate daraus.

Tabelle 2: Laufzeit Hauptroutine

	in [ms]	Bei 20 km/h [m]	Bei 40km/h [m]
Max	57	0.317	0.633
Min	8	0.044	0.089
Durchschnitt	22.0113636	0.122	0.245
Median	22	0.122	0.244

Diese Ungewissheit dabei ist sehr störend. Mit diesen 170 Versuchen konnte nur eine erste Einschätzung gemacht werden. Es kann sein, dass auch längere Verzögerungen möglich sind. Neben dem Problem des Aufrufens ist auch die Länge der Hauptroutine ein Problem. Wie man dem obigen Diagramm entnehmen kann, beschäftigt sich die Hauptroutine sehr lange mit der Supersection. Die Supersection ist ein Modus, in welchem alle Ventile offen sind. Auf dem Feld mit dem Traktor ist dies meistens die Normalität. Auf dem Gleis sieht dies jedoch ganz anders aus. Dort dürfen nicht zu viele Ventile gleichzeitig geöffnet werden. Der Grund dafür ist der Druck im Tank. Die Ventile müssen 5 l/s Auslass haben, damit die Pflanze auch verbrannt wird. Sind zu viele Ventile geöffnet, kann dies nicht garantiert werden. Dazu kommt, dass bei 95°C heissem Wasser ein Überdruck im Tank herrschen muss, damit keine Kavitation an den Pumpen entsteht. Somit ist ein grosser Teil des Codes überflüssig. Als Beispiel wurde hier der erste Teil wo, Supersection vorkommt, herauskopiert.

```

624 //assume all sections are on and super can be on, if not set false to turn off.
625 tool.isSuperSectionAllowedOn = true;
626 isHeadlandClose = false;
627 isBoundaryClose = false;
628
629 //find any off buttons, any outside of boundary, going backwards, and the farthest lookahead
630 for (int j = 0; j < tool.numOfSections; j++)
631 {
632     if (section[j].manBtnState == manBtn.Off) tool.isSuperSectionAllowedOn = false;
633     if (!section[j].isInBoundary) tool.isSuperSectionAllowedOn = false;
634
635     //check if any sections going backwards, section turned off waaay below
636     if (section[j].speedPixels < 0) tool.isSuperSectionAllowedOn = false;
637 }

```

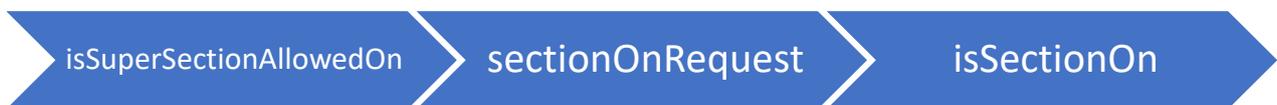
Abbildung 8: Screenshot Quellcode

Hier sieht man, dass Zeit damit verbracht wird für das Überprüfen der Supersection. Am Anfang werden drei wichtige Parameter gesetzt (Zeile 625 bis 627). Danach werden drei Argumente abgefragt, welche die Supersection wieder abschalten können. Im Beispiel wird abgefragt, ob eine Sektion entweder mit einem manuellen Knopf im Programm überschrieben, ausserhalb des

Feldes ist oder rückwärtsfährt. Auch wenn man davon ausgehen kann, dass die Zeit dafür unterhalb einer Millisekunde ist, so braucht sie doch Zeit.

4.4 Variablen

Der Quellcode braucht auch viele Variablen, die dasselbe aussagen. Es werden für die Signale immer zwei Variablen gebraucht. Die eine davon wird für die Logik im Programm selbst gebraucht. Diese Variablen werden am Anfang auf einen Wert gesetzt und dann durch alle Anforderungen überprüft. So wie im Beispiel im letzten Kapitel, gibt es noch vier weitere Passagen, die kontrollieren, ob die Supersection wirklich an sein darf. Damit ich den weiteren Ablauf erklären kann muss ich noch den Variablenverlauf klären. Folgendes Diagramm hilft beim Verständnis.



Wenn nun «isSuperSectionAllowedOn» nach allen Überprüfungen immer noch auf true steht, werden alle «sectionOnRequest» der einzelnen Sektionen auf true gestellt. Danach werden diese Request-Variablen direkt umgeschrieben in die «isSectionOn» Variablen, welche dann übersetzt werden ins «MachineByte». Dieses Byte wird dann dem Arduino geschickt, welcher dann die Relais ansteuert. Dabei muss man auch erwähnen, dass diese Variablen vierfach geführt werden. So gibt es nicht nur für das Öffnen der Sektionen eine Variable «sectionOnRequest» und «isSectionOn», sondern auch das Pendant «sectionOffRequest» und «isSectionOff». Direkt kopiert sind diese Variablen dann auch noch für das Mapping. Wir haben also acht Variablen, die den Status einer Düse darstellen. Dies ist nicht nur für die Übersicht für den Programmierer schlecht. So kann es zu Konflikten in Aussagen kommen. Was passiert, wenn sowohl «SectionOnRequest» als auch «SectionOffRequest» auf true sind? Im Programm habe ich keine dedizierte Antwort auf diese Frage gefunden. Es existieren auch noch viele andere Angaben die ständig hin und her geschrieben werden müssen. Wie gross so ein Block dann plötzlich werden kann, sieht man im folgenden Beispiel:

```
763 // If ALL sections are required on, No buttons are off, within boundary, turn super section on, normal sections off
764 if (tool.isSuperSectionAllowedOn)
765 {
766     for (int j = 0; j < tool.numOfSections; j++)
767     {
768         if (section[j].isMappingOn)
769         {
770             section[j].mappingOffRequest = true;
771             section[j].mappingOnRequest = false;
772             section[j].mappingOffTimer = 0;
773             section[j].mappingOnTimer = 0;
774         }
775         if (section[j].isSectionOn)
776         {
777             section[j].sectionOffRequest = true;
778             section[j].sectionOnRequest = false;
779             section[j].sectionOffTimer = 0;
780             section[j].sectionOnTimer = 0;
781         }
782     }
}
```

Abbildung 9: Hier sieht man wie alle Sektionen entsprechend eingestellt werden und wie viele Variablen dafür umgeschrieben werden müssen

4.5 Signalausgabe

Wie schon im vorherigen Abschnitt erwähnt werden, die Variablen «isSectionOn» respektive «isSectionOff» in ein Bit verarbeitet. Dieses Bit enthält neben den Informationen für die Sektionen auch Informationen für die automatische Lenkung. Diese Bits werden anschliessend über das USB-Kabel an den Arduino geschickt. Dieser entpackt diesen binären Satz dann wieder und befolgt die Anweisungen. In unserem Fall wird die Steuerung ausgelassen und die einzelnen Relais für die einzelnen Sektionen werden geschlossen. Diese Datenleitung kann auch direkt an die SPS weitergeschickt werden. Auf dieser müsste man dafür aber noch ein paar Zeilen Code schreiben, welche das Machinebyte wieder auseinandernehmen und die wichtigen Informationen daraus auslesen und weitergeben.

5 Fazit

Der Code ist sehr unübersichtlich gestaltet. Es können nur kleine Felder geladen werden. Er verwendet für die Beschreibung eines Düsenzustandes acht Variablen und die Berechnungsroutine wird unregelmässig gemacht. Die Berechnungsroutine enthält auch viele Zeilen, die auf dem Gleis nicht gebraucht werden. Durch die unregelmässige Berechnung variiert die Präzision. Um die Präzision zu quantifizieren, dient folgende Tabelle:

Tabelle 3: Quantifizierung Präzision

	Zeitaufwand GPS [ms]	Zeitaufwand Berechnung [ms]	Zeitaufwand gesamt [ms]	Gefahrenere Distanz bei 20 km/h [m]	Gefahrenere Distanz bei 40 km/h [m]
Min	1	8	9	0.05	0.10
Max	10	57	67	0.37	0.744

Beim minimalen Zeitaufwand wird davon ausgegangen, dass das GPS-Signal beim Ausleseversuch vorhanden ist und die Berechnungsroutine anschliessend ununterbrochen berechnen kann. Beim maximalen Zeitaufwand wird davon ausgegangen, dass das GPS-Signal nach einem Ausleseversuch vorhanden ist und die Berechnungsroutine anschliessend unterbrochen wird. Diese Ungenauigkeit macht es auch sehr schwierig, mit der SPS zu kommunizieren.

Mit den bisherigen Informationen kann man bereits eine Entscheidung fassen. Dafür kann man die drei Lösungsprinzipien vom Anfang mit den Ergebnissen vergleichen.

5.1 Lösungsprinzip 1

Grundsätzlich ist die Übernahme des Programmes ohne Veränderung möglich. Aber ich kann dieses Prinzip nicht empfehlen und zwar aus folgenden Gründen:

- **Opensource-Projekt**
Die SBB kann sich nicht sicher sein, wann die Software erneuert wird und welche Art die Neuerungen sind. Für die SBB heisst das, dass man entweder mit der aktuellen Version arbeitet und diese nicht mehr updatet, oder man begibt sich in eine ungewisse Zukunft. Zusätzlich hat man keinen Support für das Programm. Treten Fehler auf, muss man diese selbst beheben.
- **Falscher Fokus der Software**
Das AgOpenGPS-Projekt ist klar auf die Landwirtschaft ausgelegt. Die meisten Optionen im Programm sind für die Gleise uninteressant. In Kombination mit dem Faktor Opensource ist fraglich, ob in naher Zukunft die VRA verbessert wird.
- **Windowsbasiert**
Windows ist ein weiteres System, welches auf dem Zug verbaut werden muss. Das heisst automatisch, dass es eine weitere mögliche Fehlerquelle gibt. Dazu kommt, dass Windows nicht getaktet läuft. Dies kann dazu führen, dass grosse Unsicherheiten entstehen.
- **Nicht getaktet**
Die Berechnungsroutine ist nicht regelmässig. Das kann erhebliche Ungenauigkeiten nach sich ziehen. Auch die Kommunikation mit einer SPS wird so erheblich erschwert.
- **Programm wird nicht sinngemäss ausgeführt**
Damit das Programm auch ohne Veränderung am Quellcode funktioniert, müssen die Ein-/Aus-Befehle absichtlich falsch interpretiert werden auf der SPS. Dies bietet weitere Möglichkeiten für Fehler.
- **Feldgrössen**
Es können nur kleine Felder geladen werden. Das hat zur Folge, dass man eine Strecke in einzelne Teilstrecken aufteilen müsste. Auf den Gleisen ist dies höchst unpraktisch.

Trotz all diesen Nachteilen, muss man erwähnen, dass diese Option vermutlich am wenigsten kostet, zum jetzigen Zeitpunkt. Wenn man jedoch in die Zukunft schaut, kann dieses Prinzip grosse Kosten verursachen. Wenn das Programm sich in eine andere Richtung entwickelt oder man doch noch Verbesserungen machen will, fängt man von null an. Dies würde bedeuten, dass sich eine Person oder Firma einarbeiten müsste, was viel Zeit und somit auch Geld in Anspruch nehmen würde.

5.2 Lösungsprinzip 2

Das zweite Lösungsprinzip ist sehr dehnbar. Hier muss auch die Entscheidung getroffen werden, ob man selbst eine Basis programmieren und dann Codeabschnitte hineinkopieren will oder ob man den bestehenden Code erweitern will. Nach der zweiten Phase ist für mich klar, dass man selbst eine Basis programmieren soll. Der Vorteil ist das Wissen und die Präzision, die man dabei kriegt. Wenn man selbst eine Basis programmiert, kann man auch gezielt Schwachstellen im Code verbessern. Wenn man den Code lediglich erweitert, läuft man in Gefahr, ein grösseres Chaos als bereits vorhanden zu kreieren. Die Vor- und Nachteile dieses Prinzips sind folgende:

- + Aufräumen des Codes
Die Software, die man danach erhält, wird viel übersichtlicher sein. Dies hilft bei weiteren Anpassungen in der Zukunft.
- + Auf Anwendung zugeschnitten
Den Quellcode kann man so viel kleiner machen. Dies hat zur Folge, dass die Software weniger fehleranfällig und optimalerweise auch einfacher zu bedienen ist (nur noch Optionen vorhanden, die auch wichtig sind im Betrieb). Auch die Präzision sollte sich so verbessern, da man so das Problem mit der Hauptroutine behebt.
- Updates
Bei jedem Update muss man den Quellcode von Neuem durchforsten, um die richtigen Codeabschnitte zu finden. Wenn es unglücklich verläuft, wurden die betreffenden Abschnitte nicht verändert und man arbeitete für nichts.
- Zeitaufwand
Damit man etwas am Quellcode verändern oder herauskopieren kann, muss man ihn verstehen. Dies ist eine sehr zeitintensive Angelegenheit. Der Code ist nicht ordentlich geschrieben, was die Aufarbeitung umso komplizierter macht.
- Windowsbasiert
Auch diese Software wird noch auf Windows laufen müssen. Hier gelten die gleichen Nachteile wie bei Lösungsprinzip Nummer eins.

Auch wenn dieses Lösungsprinzip Vorteile hat, empfehle ich dieses Prinzip nicht. Der Aufwand der Aufbereitung übersteigt meiner Meinung nach den Aufwand für die komplette Neu-Programmierung. Der Code ist so verschlungen, dass man sehr lange braucht, bis man den Code überhaupt versteht. Die Teile, die man dann schliesslich herauskopieren könnte, sind aus meiner Sicht einfach und tragen nicht viel zur Zeitverkürzung des Programmieraufwandes bei.

5.3 Lösungsprinzip 3

Das dritte Lösungsprinzip scheint mir die beste Lösung zu sein. Man kann sich durchaus ein wenig am AgOpenGPS orientieren. Aber den Zeitaufwand, um das komplette Programm zu verstehen, erachte ich als überflüssig. Dazu kommen weitere hilfreiche Optionen für die Software. Ein Beispiel dazu ist die Überlegung, die Software direkt auf der SPS zu integrieren. Man würde sich eine Hardware sparen und gleichzeitig die Kommunikationszeit verkleinern, da man keine Kabel mehr hat. Die Vor- und Nachteile dieses Prinzips sind folgende:

- + Angepasst
Die Software macht nur das, was von ihr gewünscht wird. Sie enthält keinen Code, der überflüssig ist.
- + Übersichtlich
Der Quellcode kann übersichtlich geschrieben werden. Dies hilft nicht nur bei späteren Erweiterungen, sondern trägt auch zur Sicherheit bei.
- + Erweiterbar
Wenn man den Code selbst entwickelt hat, weiss man auch, wo die Schnittstellen sind. Mit einer sauberen Dokumentation sollte es dann in der Zukunft keine Probleme bereiten, den Code mit zusätzlichen Funktionen zu erweitern.
- + Plattform offen
Wie oben schon angetönt, sollte man sich die Frage der Plattform stellen. Wenn man den Code selbst erstellt, kann man ihn auch so schreiben, dass er auf einer SPS läuft. Dies würde viele potenzielle Fehlerquellen eliminieren.
- Mehraufwand
Ein eigenes Programm zu erstellen, bedeutet einen grossen Mehraufwand. All die positiven Punkte, die ich erwähnt habe, müssen evaluiert und erstellt werden.
- Entwicklungszeit
Die Entwicklungszeit wird bei einem neuen Programm gross sein. Wie lange so eine Entwicklungszeit ist, kann ich nicht abschätzen.

Dies ist aus meiner Sicht das bevorzugte Lösungsprinzip. Es mag ein grosser Zeitaufwand sein, bis man seine eigene Software hat. Dies zahlt sich jedoch mit der Zeit aus. Wenn man nachträglich das System noch verändern will, spart man hier gegenüber den anderen zwei Lösungsprinzipien enorm Zeit. Dazu kommt, dass die erarbeitete Software gezielt die Funktionen besitzt, die man auch braucht.

6 Dritte Phase

Nach dem Fazit in Kapitel fünf sprach ich mit meinem Auftraggeber. Dieser kann meine Begründungen verstehen und will sich auch nach diesen Empfehlungen richten. Da ich aber kein Informatiker bin, entschieden wir uns, dass ich mich weiter im Lösungsprinzip zwei vertiefe. Die Hoffnung dabei sind die Erkenntnisse, die man eventuell auch für ein neu erstelltes Programm nutzen kann.

Eine grosse Frage besteht noch im Laden der Karte. In der ersten Phase wurde festgestellt, dass bei grossen Karten das Programm einfriert und Windows eine Meldung herausgibt, dass das Programm nicht mehr reagiere. Wenn man dieselbe Karte jedoch auf Google Earth lädt, wird diese schnell und ohne Probleme korrekt angezeigt. Um nun die Stelle zu finden, an welcher das Programm abstürzt, wurde das Programm nochmals im Visual Studio simuliert. Dort wurde versucht eine grosse Karte zu laden.

Zu meiner Überraschung zeigte Visual Studio keine Fehlermeldung an. Nach einiger Zeit wurde die Karte dann tatsächlich geladen und das Programm funktionierte einwandfrei. Es stellt sich heraus, dass Windows selbst eine Warnung auslöste, das Programm aber immer noch am weiterrechnen war. Um herauszufinden, welche Stelle so viel Zeit in Anspruch nimmt, simulierte ich das Programm noch einmal und pausierte im Ladeprozess. Als ich die Zeilen gefunden habe, welche so viel Zeit in Anspruch nehmen, kommentierte ich diese aus und simulierte die Software nochmals. Ohne die auskommentierten Zeilen lud die Software die Karte ohne Probleme. Dies bestätigte mir, dass ich die richtigen Zeilen gefunden habe.

Da wir uns für das dritte Lösungsprinzip entschieden haben, kümmerte es mich nicht gross ob das Programm mit den auskommentierten Zeilen irgendwelche Fehler produziert. Stattdessen analysierte ich den Grund, wieso dass diese Zeilen so viel Zeit in Anspruch nehmen.

6.1 Karten Laden

Wenn eine neue Karte geöffnet wird, werden alle Punkte miteinander verglichen. Dabei versucht das Programm die Umrise so zu legen, dass der Traktor eine ideale Fahrtlinie erhält. Dafür wird auch überprüft, wo das Feld zu eng ist oder wo die Fahrtlinien zu nahe an die Feldgrenzen kommen würden. Für diesen Prozess wird sehr viel Zeit in Anspruch genommen. Um eine Schätzung zu erhalten, wie lange ein Laden einer Karte geht, wurde mit unterschiedlich vielen Punkten die Zeit gemessen. Dabei wurde bei wenigen Punkten 10 mal wiederholt, um sicher zu stellen, dass die Messung kein Zufall ist. Bei grösserer Anzahl reduzierten sich dann die Anzahl Messungen. Dies liegt daran, dass die Messungen mehr Zeit in Anspruch nahmen und dadurch die vorherigen Messungen davon ausgegangen werden konnte, dass nur kleine Abweichungen vorkommen. Die Daten für den folgenden Graph sind im Anhang 12.3 einzusehen.

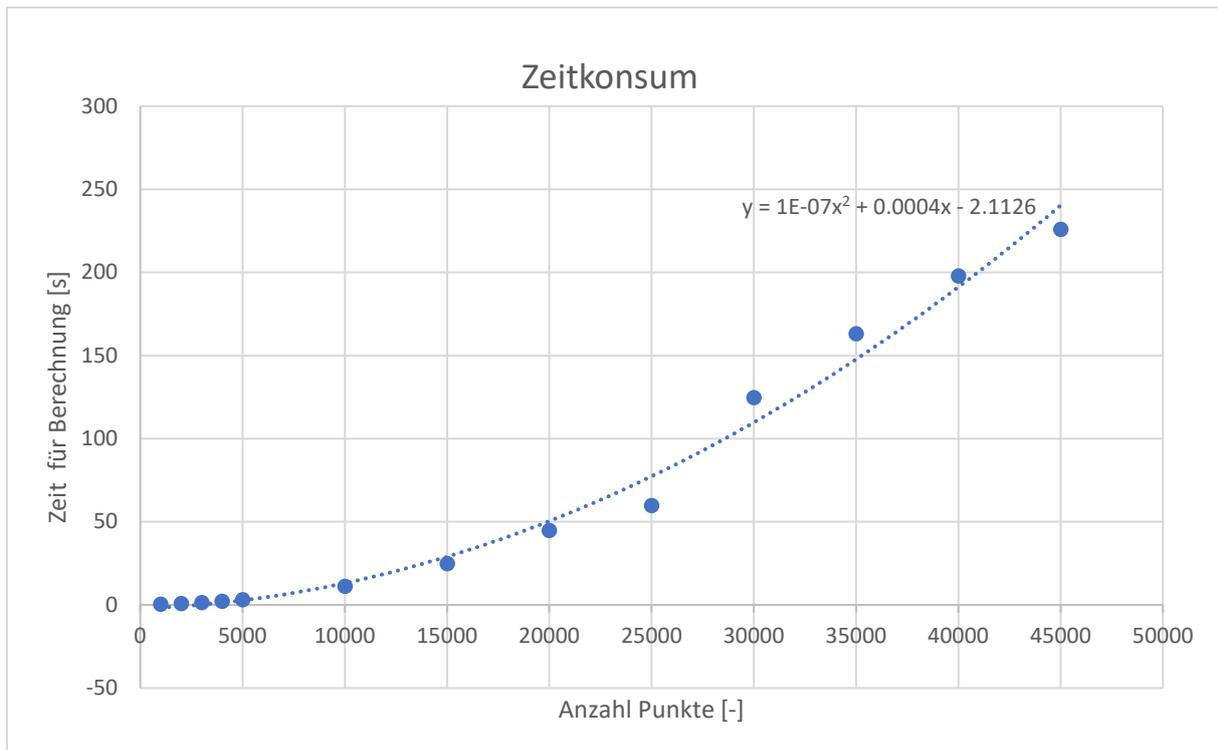


Abbildung 10: Graph Zeitkonsum

Wie man im Graphen sehen kann, kann man den Zeitaufwand mit einem Polynom zweiten Grades annähern. Dieser Test wurde mit einer Karte gemacht, welche von Hand gezeichnet wurde. Wenn man nun ausgeht, dass die Karte von einer AI gezeichnet wird, kann man annehmen, dass eine Karte über einer Million Punkte hat. Schätzt man den Zeitaufwand dafür mit dem Polynom, so kommt man auf eine Ladezeit von rund 27 Stunden. Da das Polynom von Karten entspringt, die maximal 45'000 Punkte haben, fungiert dieses Polynom nur als Schätzung. Wenn man dieses Polynom genauer machen will, müsste man einen Test durchführen mit einer Million Punkten. So könnte auch ausgeschlossen werden, dass neben dem hohen Zeitaufwand noch andere Probleme entstehen. Dies konnte so noch nicht getestet werden, da so grosse Karten zurzeit nicht vorhanden sind.

```

//remove the points too close to boundary
for (int i = 0; i < bndCount; i++)
{
    for (int j = 0; j < lineCount; j++)
    {
        //make sure distance between headland and boundary is not less then width
        distance = glm.Distance(curBnd[i], turnLine[j]);
        if (distance < (totalHeadWidth * 0.99))
        {
            turnLine.RemoveAt(j);
            lineCount = turnLine.Count;
            j = -1;
        }
    }
}

```

Abbildung 11: Screenshot aus Quellcode

Der Grund, wieso der Zeitkonsum nicht linear ansteigt, liegt in der Programmierung. Mit diesen zwei for-Schleifen wird jeder Punkt der Feldgrenze mit jedem Punkt der Polygone im Feld verglichen. Wenn also mehr Polygone oder eine komplexere Feldgrenze gezeichnet wird, wird der Aufwand doppelt erhöht. Dies sind zwei der vier for-Schleifen in diesem Abschnitt, welche alle Punkte durchrechnen. Die anderen zwei rechnen nur mit den Punkten der Feldgrenze und enthalten keine doppelten for-Schleifen.

7 Ausblick

Das empfohlene Lösungsprinzip wird bereits im Fazit der zweiten Phase gemacht. Daher gehe ich in diesem Kapitel davon aus, dass die SBB die Software selbst programmiert. Für den Ausblick kann man also die positiven Punkte aus dem Lösungsprinzip zwei nehmen und einzeln diskutieren.

7.1 Angepasste Software

Man muss sich überlegen, welche Teilfunktionen die Software übernehmen muss. Der Fokus dabei sollte nicht nur bei der Präzision sein. Da bei breiten Pflanzenfeldern nicht alle Ventile gleichzeitig geöffnet werden dürfen, wird dort sowieso nicht schnell gefahren. Ein wichtiger Teil ist deshalb auch der Ausblick nach vorne. Wenn man sieht, dass der nächste Abschnitt keine Pflanzen enthält, kann man schnell fahren und gut Zeit wettmachen.

Wie dieser Ausblick aussehen wird, ist offen. Eine Option wäre das Laden der kartierten Pflanzen auf ein Mobiltelefon oder Tablet mit GPS-Empfang. Da der HWZ ohnehin einen Beifahrer braucht, könnte dieser die Karte lesen und laufend die Informationen an den Lokführer geben. Dies wäre eine schnelle und kostengünstige Variante. Eine andere Option wäre die Integration dieser Information beim Interface auf der SPS. Bei der parallellaufenden Bachelor-Thesis von Roger Brast wurde so ein Interface bereits programmiert. Die Integration darin wäre somit nicht mit grossem Programmieraufwand gekoppelt, wenn man die neue Software auf der SPS laufen lässt.

7.2 Plattform

Die AgOpenGPS-Software läuft auf Windows. Wenn man eine eigene Software programmiert, muss man unbedingt in Betracht ziehen, die Hardware zu wechseln. Wenn man die neu programmierte Software direkt auf der SPS integriert, erhält man wesentliche Vorteile. Die Kommunikationswege werden erheblich verkürzt, wenn man einen gemeinsamen Speicher mit der SPS hat. Die Hardware wird minimiert und mit ihr die möglichen Fehlerquellen. Wenn man trotzdem weiter eine windowsbasierte Software haben will, muss man unbedingt Windows IOT anschauen. Windows IOT ist, wie der Name schon sagt, auf Industrie4.0 ausgerichtet («internet of Things»).

7.3 Schnittstelle

Eine weitere wichtige Überlegung für die Zukunft ist die Schnittstelle zwischen der neuen Software und der SPS. Mit AgOpenGPS wäre die Schnittstelle ein USB-Kabel, welches die SPS mit dem Tablet verbindet. Das Signal wäre das Machinebite, welches lediglich Ein- Aus-Anweisungen gibt. Wenn man die Software auf der SPS integriert, ergeben sich viel mehr Möglichkeiten der Kommunikation.

7.4 Datenbewältigung

Bei AgOpenGPS ist ein Problem das Laden der Karte. Man muss einen effizienten Weg finden, wie man so grosse Datenmengen schnell verarbeiten kann. Dabei muss man sich überlegen, wie man die Erkennung machen will, ob man in einem Feld ist oder nicht. AgOpenGPS macht die Erkennung über Pixel. Es gibt aber auch andere, mathematische, Möglichkeiten, um zu bestimmen, ob ein Punkt in einer Fläche drin ist oder nicht. Diese könnten effizienter sein. Das Problem mit der Datenmenge ist aber dadurch nicht gelöst. Das Problem dabei ist immer die Anzahl der Flächen, die überprüft werden müssen. Um diese Anzahl zu reduzieren, wäre es eine Möglichkeit, eine Art «Roadmap» zu generieren. Diese gibt an, wo auf der Strecke sich die Felder befinden. Mittels Drehgebers am Zug kann man einfach und präzise bestimmen, auf

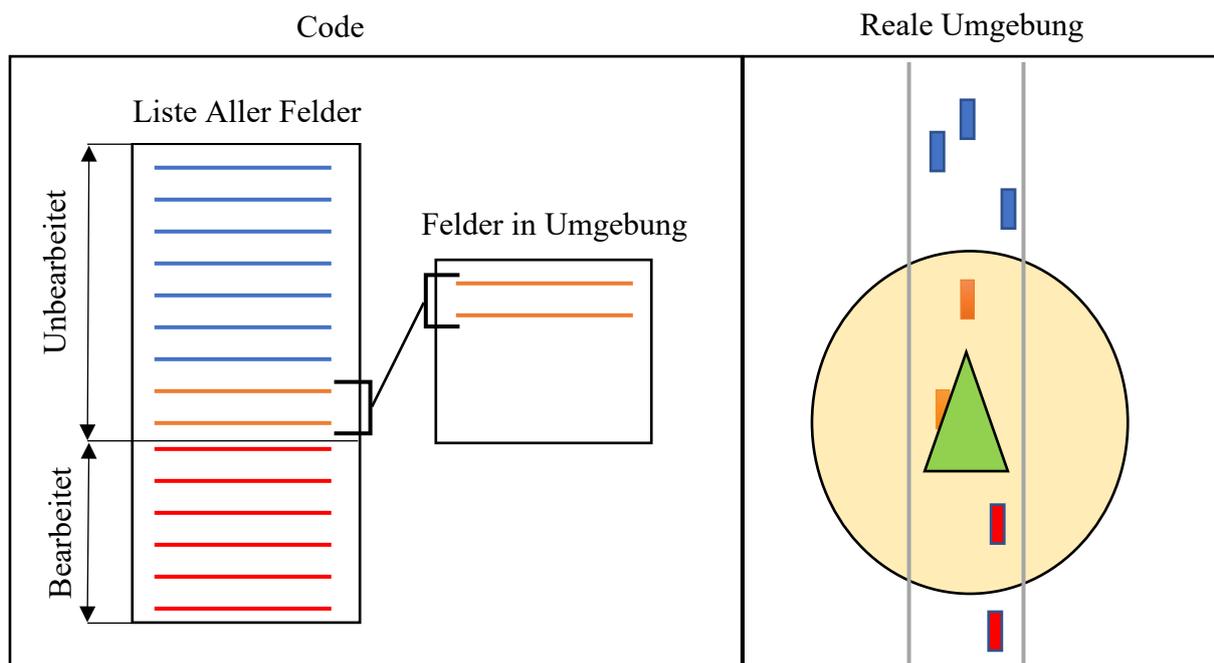


Abbildung 12: Roadmap

welchem Teil der Strecke man sich befindet. So muss der Zug nur die Felder überprüfen, die auch in der Umgebung sind.

Eine Solche Lösung würde auch eine gute Grundlage für Erweiterungen bieten. So könnte man auch noch Informationen zu den Pflanzen mitgeben und die Behandlung entsprechend anpassen.

Diese Lösung wirft aber auch die Frage des Formates der GIS-Dateien auf. AgOpenGPS benötigt die Karten als kml. Es stellt sich die Frage, ob man weiterhin dieses Format brauchen will. Sie ist praktisch, da sie normiert und dadurch universal ist. Wenn man jedoch so ein dediziertes System programmiert, muss man eventuell auch selbst ein Format entwickeln, welches den Ansprüchen gerecht wird und alle Informationen übermitteln kann.

7.5 Sicherheit

In dieser Arbeit wird die Sicherheit des Programmes nicht diskutiert. In der Zukunft wird dies jedoch ein Thema sein. Momentan hat die SBB mit dem HWZ ein Testgerät, welches nicht die Normen von einem normalen Zug erfüllen muss[7]. Sobald man aber ein Serienfahrzeug bauen will, muss man verschiedene Sicherheitsaspekte erfüllen. Zum Beispiel die Überwachung der Umgebung. Man muss sich Gedanken machen, was passiert, wenn ein Arbeiter während der Bearbeitung zu nahe an die Ventile kommt oder der Zug eine Bahnhofsdurchfahrt macht.

8 Reflexion

Diese Arbeit weicht von einer gewöhnlichen Bachelor-Thesis eines Maschinenbauingenieurs ab. Daher war es anfangs schwierig eine gute Systematik zu entwickeln. Der Ablauf mit den verschiedenen Phasen gab mir jedoch schlussendlich eine gute Übersicht. Die grösste Herausforderung war in diesem Projekt das Finden der richtigen Zeilen im Code. Der Quellcode ist so undurchsichtig programmiert, dass ich anfangs nicht wusste, wie ich die richtigen Stellen finden sollte. Mit der Hilfe eines versierten Kollegen und dem Visual Studio konnte ich dann aber einen guten Anfang machen. Schlussendlich glaube ich, dass ich einige wichtige Stellen finden und analysieren konnte. Im Gespräch mit dem Industriepartner erfuhr ich auch einiges über das gesamte Projekt und das Feedback war auch gut. Ich konnte wichtige Fragen klären und dem Gesamtprojekt weiterhelfen. Wie weit eine solche Analyse geht, ist immer schwer vorzustellen. Dies habe ich bei dieser Arbeit gut bemerkt. Es braucht viel Übersicht, damit man auch alle Faktoren betrachtet. Umso schwieriger ist es zu erkennen, welche Themen man noch vertiefter anschauen soll und welche nicht. Ein gutes Beispiel dafür ist das Laden der Karte. In der ersten Phase sah es so aus, als würde das Laden grösserer Karten nicht gehen. Als ich dann den vermeintlichen Absturz der Software genauer untersuchen wollte, musste ich feststellen, dass die Software gar nicht abstürzt. So wurde das Lösungsprinzip eins wieder machbarer als nach Phase eins gedacht.

Auch wenn in der Letzten Phase noch bemerkt wurde, dass das Laden der Karte auch mit grossen Karten funktioniert, empfehle ich der SBB weiterhin die Programmierung eines eigenen Programmes. Auf den ersten Blick sieht die Übernahme der Software verlockend aus. Wenn man jedoch den Quellcode genauer anschaut, findet man einen grossen und unübersichtlichen Code, welcher viele Zusatzfunktionen besitzt, welche die SBB nie brauchen wird.

9 Glossar

A		
AI	artificial intelligence	
G		
GIS	Geoinformationssystem Dieses System wird gebraucht um globale Datenpunkte auszutauschen, darzustellen und zu speichern.	
GPS	Globalpositioningsystem	
H		
HWZ	Heisswasserspritzzug	
K		
kml	Keyhole Markup Language. Ein Dateiformat für geographische Daten.	
L		
Lookahead	Ein Bereich, der sich zwischen Ventilen und einer vordefinierten Grenze befindet. Dieser Bereich befindet sich immer vor den Ventilen.	
M		
Machinebyte	Ein vordefiniertes Format für Daten, welche von der Software auf den Arduino geschickt werden.	
P		
parsen	Beim Parsen werden Datensätze auseinandergeschnitten und zur Weiterverarbeitung in andere Formate geschrieben.	
Q		
Quellcode	Das Programm in seiner ursprünglichen Programmiersprache. Daraus kann das richtige Programm kompiliert werden	
S		
SBB	Schweizerische bundesbahnen	
Serialbuffer	Der Serialbuffer speichert die Daten, die ihm über den Serialport geschickt werden.	
Serialhandler	Der Serialhandler sucht im Serialbuffer nach sinnvollen Datensätzen und schickt diese weiter	
Serialport	Der Serialport ist eine Hardwareschnittstelle zwischen zwei Systemen.	
Supersection	Ein Modus, bei welchem alle Ventile eingeschaltet sind	
V		
VRA	variable-rate application Bezeichnet die gezielte Bearbeitung des Bodens. Im Falle des HWZ bedeutet das, dass man die Pflanzen gezielt bespritzt und nicht das ganze Gleis abspritzt.	
Visual Studio	Ein Programm von Microsoft, um Quellcodes zu schreiben und zu analysieren.	

10 Abbildungsverzeichnis

Abbildung 1 Prinzip Bild des Heisswasserspritzzug.....	3
Abbildung 2:Darstellung Lösungsprinzipien	7
Abbildung 3: Screenshot AgOpenGPS Hauptfenster.....	8
Abbildung 4: Feldgrenzen	10
Abbildung 5: Erwartete Zustandsmaschine.....	11
Abbildung 6: Ablauf Signalempfang	11
Abbildung 7: Ablauf Hauptroutine	13
Abbildung 8: Screenshot Quellcode.....	14
Abbildung 9: Hier sieht man wie alle Sektionen entsprechend eingestellt werden und wie viele Variablen dafür umgeschrieben werden müssen.....	15
Abbildung 10: Graph Zeitkonsum.....	21
Abbildung 11: Screenshot aus Quellcode	22
Abbildung 12: Roadmap	24
Abbildung 13: Screenshot Geräte Einstellungen Konfiguration.....	29
Abbildung 14: Screenshot Geräte Einstellungen Anhängung.....	30
Abbildung 15: Screenshot Geräte Einstellungen Einstellungen	30
Abbildung 16: Screenshot Geräte Einstellungen Teilbreite	31
Abbildung 17: Screenshot Geräte Einstellungen Schalter	31
Abbildung 18: Screenshot Fahrzeugeinstellungen Typ	32
Abbildung 19:: Screenshot Fahrzeugeinstellungen Antenne	32
Abbildung 20: Screenshot Fahrzeugeinstellungen Fahrzeug	33
Abbildung 21: Screenshot Fahrzeugeinstellungen Führungslinien.....	33
Abbildung 22: Koordinaten für Simulator	34
Abbildung 23: Screenshot Lokation Simulationskoordinaten	34
Abbildung 24: Aufschlüsselung NMEA Satz GPGGA.....	35
Abbildung 25: Aufschlüsselung NMEA Satz GPVTG	36
Abbildung 26: Aufschlüsselung NMEA Satz GPRMC	36
Abbildung 27: Aufschlüsselung NMEA Satz GPHDT	36
Abbildung 28: Darstellung Zeitkonsum pro Punkt beim Laden einer Karte	39

11 Tabellenverzeichnis

Tabelle 1: Kurzbeschrieb Funktionen	9
Tabelle 2: Laufzeit Hauptroutine	14
Tabelle 3:Quantifizierung Präzision.....	16
Tabelle 4: Zeitmessungen der Ladezeiten von Karten	39

12 Quellenverzeichnis

- [1] „AgOpenGPS - Farmers from all over the world helping farmers learn and build“, *AgOpenGPS*. <https://agopengps.discourse.group/> (zugegriffen Mai 31, 2021).
- [2] „YouTube“. <https://www.youtube.com/> (zugegriffen Mai 31, 2021).
- [3] „Google Earth“. <https://earth.google.com/web/@47.34090193,8.05241625,426.83384046a,20210.25433547d,30y,0h,0t,0r> (zugegriffen Mai 31, 2021).
- [4] „Welcome to the QGIS project!“ <https://www.qgis.org/en/site/> (zugegriffen Mai 31, 2021).
- [5] „swipos-GIS/GEO“, *Bundesamt für Landestopografie swisstopo*. <https://www.swisstopo.admin.ch/de/geodata/geoservices/swipos/swipos-dienste/swipos-gisgeo.html> (zugegriffen März 29, 2021).
- [6] „simpleRTK2B multiband RTK application board with u-blox ZED-F9P“, *ArduSimple*. <https://www.ardusimple.com/simplertk2b/> (zugegriffen März 30, 2021).
- [7] „Richtlinie Europäisches Parlament“. Zugegriffen: Juni 08, 2021. [Online]. Verfügbar unter: <https://eur-lex.europa.eu/legal-content/DE/TXT/PDF/?uri=CELEX:32006L0042&from=DE>
- [8] „20190303 nmea 0183 talker identifier mnemonics.pdf“. Zugegriffen: Juni 01, 2021. [Online]. Verfügbar unter: <https://www.nmea.org/Assets/20190303%20nmea%200183%20talker%20identifier%20mnemonics.pdf>
- [9] „NMEA.DE: Erklärungen und Hilfen zu NMEA Schnittstellen“. <http://www.nmea.de/nmea0183datensaeetze.html#hdt> (zugegriffen Juni 01, 2021).

13 Anhang

13.1 Grafische Oberflächen AgOpenGPS

In diesem Kapitel werden alle Unteroptionen erklärt, welche für den Betrieb auf dem Gleis relevant sind.

13.1.1 Geräte Einstellungen

Im Kapitel 3.1 sieht man, wie man zu den Geräteeinstellungen kommt.

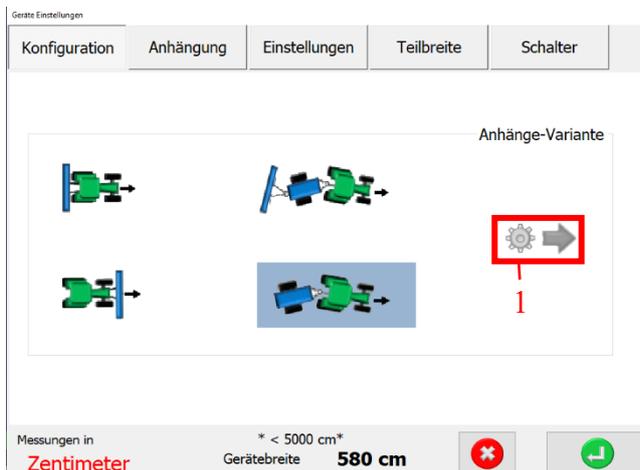


Abbildung 13: Screenshot Geräte Einstellungen Konfiguration

Der erste Tab lässt einen auswählen, wie die Achskonfiguration des Anhängers sein soll. Für den HWZ kommt es darauf an, wo man die GPS-Antenne befestigt. Ist sie auf dem ersten Sprühwagen, macht die erste Option ohne Gelenk Sinn. Ist sie auf der Lokomotive angebracht, so nimmt man am besten die Konfiguration mit einem Gelenk (hier hellblau markiert). Damit man mit der gewählten Variante fortfahren kann, muss man das Zahnrad mit dem Pfeil anklicken (1). Diese Konfiguration ist wichtig in der Kurvenfahrt. Da Programm berechnet so die Lage der Sprüzdüsen, auch wenn der HWZ eine Kurve macht.

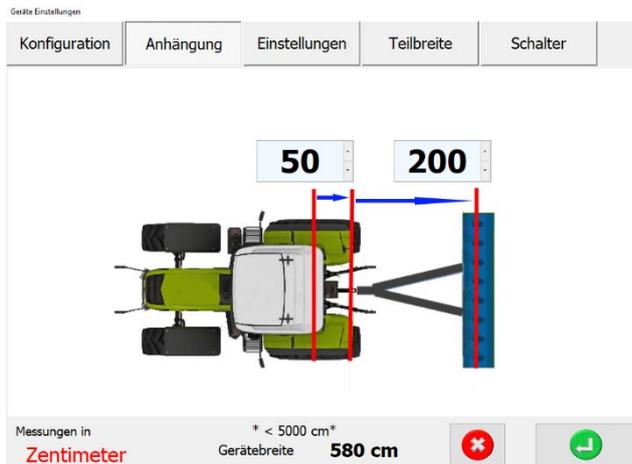


Abbildung 14: Screenshot Geräte Einstellungen Anhängung

Diese Darstellung präsentiert sich, wenn man die Konfiguration mit einem Gelenk wählt. Hier muss man angeben, wie weit das Gelenk von der Hinterachse entfernt ist. Die Hinterachse gilt für das ganze Programm als Fixpunkt. Auch die GPS-Antenne wird auf diese Achse referenziert. Wählt man Die Option ohne Gelenk, kann man nur die Distanz zwischen dem Hinterrad und dem Anhänger verstellen.

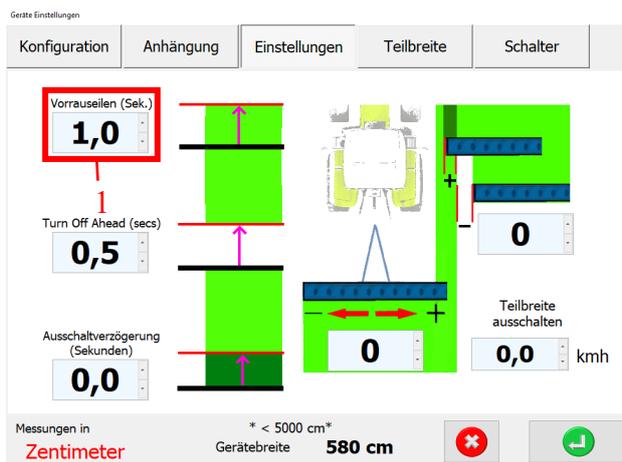


Abbildung 15: Screenshot Geräte Einstellungen Einstellungen

Der nächste Tab befasst sich mit der VRA. Hier kann man angeben, wie die Sektionen reagieren sollen. Dies könnte bei einer schnellen Fahrt nützlich sein. Vor allem die Einstellung «Vorrauseilen» (1) ist interessant. Da die Ventile eine Zeit brauchen, bis sie geöffnet sind, kann man die Öffnung der Ventile bereits vorausschauend machen. Das heisst jedoch nicht, dass man so unlimitiert schnell fahren kann. Das Problem dabei ist die maximale Anzahl an Ventilen, die gleichzeitig geöffnet werden dürfen. Je schneller man fährt, desto höher ist die Wahrscheinlichkeit, dass zu viele Ventile geöffnet werden müssen. Diese Funktion ist also praktisch für die Präzision, macht das System aber nicht schneller.

Der Rest der Einstellungen ist weniger interessant, da man die Ventile nicht länger geöffnet haben will als nötig und bei einem Zug die Überdeckung zweier Bahnen gegeben ist. Auch der Sprühbalken sitzt in der Mitte und muss nicht verschoben werden.

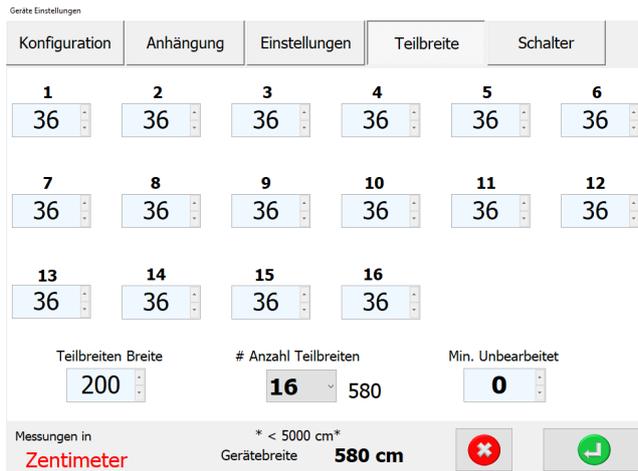


Abbildung 16: Screenshot Geräte Einstellungen Teilbreite

Der nächste Tab lässt einen die Anzahl und die Breite jeder einzelnen Sektion bestimmen. Hier im Beispiel wird davon ausgegangen, dass jedes Ventil gleich breit ist. In der Realität wird dies jedoch nicht der Fall sein. Die Ventile für die äusseren Bereiche sind höher angebracht und daher breiter. Für den Simulationszweck spielt dies keine Rolle.



Abbildung 17: Screenshot Geräte Einstellungen Schalter

Der Letzte Tab beschäftigt sich mit der Arbeitsweise des Relais. Man kann selber einstellen, ob das aktivierte Relais Strom durchlässt oder nicht. Für die SPS macht es keinen Unterschied, da sie beide Signalarten interpretieren kann. Vermutlich ist jedoch «Active Low» für den HWZ besser. So würden im Falle eines Kabelbruchs oder Signalunterbruchs die Ventile geschlossen bleiben.

13.1.2 Fahrzeugeinstellungen

Im Kapitel 3.1 sieht man, wie man zu den Fahrzeugeinstellungen kommt.

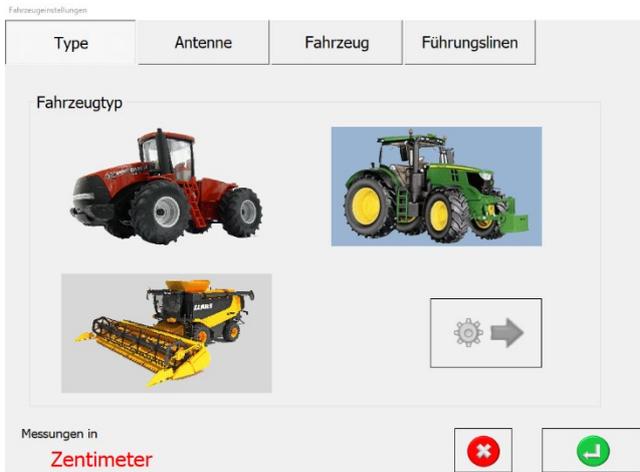


Abbildung 18: Screenshot Fahrzeugeinstellungen Typ

Der erste Tab dient zur Auswahl des Fahrzeugs. Diese Auswahl ist für den HWZ nicht so wichtig. Alle Modelle, die zur Auswahl stehen, können die beiden wichtigen Parameter verändern. Die zwei wichtigen Parameter dabei sind die Distanz zwischen GPS-Antenne und Boden sowie die Distanz zwischen GPS-Antenne und Hinterachse. Der Unterschied zwischen den Modellen ist lediglich der Ort der Steuerachse. Diese Distanz wird nur für die Bahnführung auf dem Feld gebraucht und ist somit nicht wichtig für den HWZ.

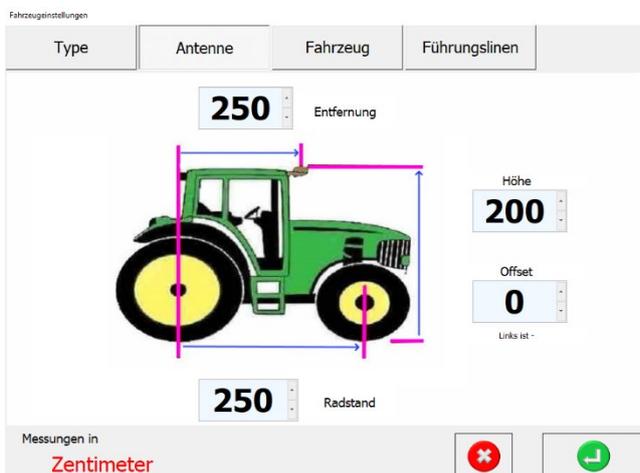


Abbildung 19: Screenshot Fahrzeugeinstellungen Antenne

Der Zweite Tab, hier mit dem Traktor dargestellt, ist wichtig für die Position der GPS-Antenne. Dabei muss die GPS-Antenne bei jeder Konfiguration vor der starren Achse sein. Für den HWZ gibt es daher zwei Möglichkeiten. Bei der einen Variante ist die GPS-Antenne auf dem Spritzwagen angebracht. Bei dieser Variante kann eine virtuelle Achse angenommen werden. Dabei muss auch darauf geachtet werden, dass man bei der Konfiguration einen Anhänger nimmt, der keine Achse besitzt. Der Abstand der stimmen muss, ist der Abstand zwischen GPS-

Antenne und erster Spritzreihe. Bei der zweiten Variante montiert man die GPS-Antenne auf die Lokomotive. Die Starre Achse ist dort beim hinteren Drehgestell in der Mitte. Dafür muss man bei der Konfiguration des Anhängers den wählen, der eine Achse hat. Die zweite Variante ist jedoch wegen diversen Schwierigkeiten nicht empfehlenswert. Das grösste Problem dabei ist die Verkabelung. Wenn der Zug umkehren will, muss der Zug auf die andere Seite der Wagen bewegt werden. Wenn die GPS-Antenne an der Lokomotive angebracht ist, führt das dazu, dass man die Kabel abhängen muss und auf der anderen Seite sehr lange Kabel braucht, die über alle vier Wagen gehen. Durch die längeren Kabel verschlechtert sich dann auch die Präzision. Bei all diesen Nachteilen gibt es aber auch einen Vorteil. Wenn die GPS-Antenne an der Lokomotive angebracht wird, ist die Fahrtrichtung besser anpassbar. So ist die GPS-Antenne immer an der Spitze des HWZ, egal an welcher Seite die Lokomotive angehängt ist.

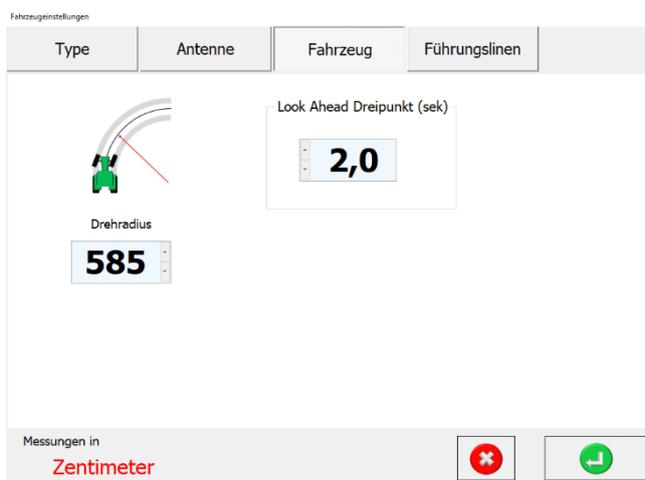


Abbildung 20: Screenshot Fahrzeugeinstellungen Fahrzeug

Der nächste Tab ist für den HWZ nicht relevant. Der Drehradius wird nur für die Bahnplanung verwendet.

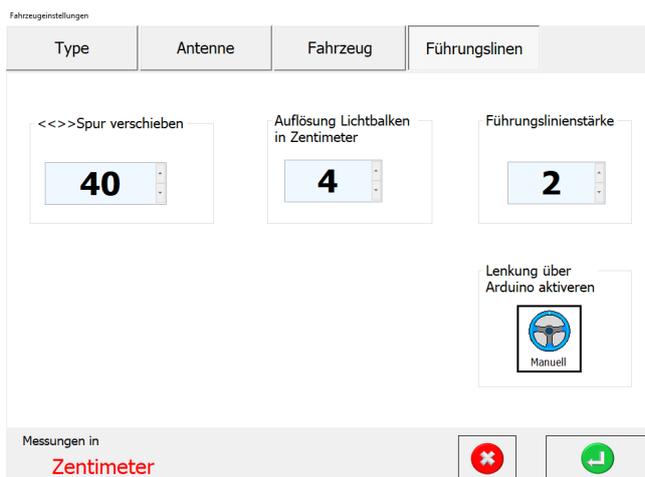


Abbildung 21: Screenshot Fahrzeugeinstellungen Führungslinien

Auch der Letzte Tab ist uninteressant für den HWZ, da er sich nur mit der Bahnplanung beschäftigt.

13.1.3 Simulationskoordinaten

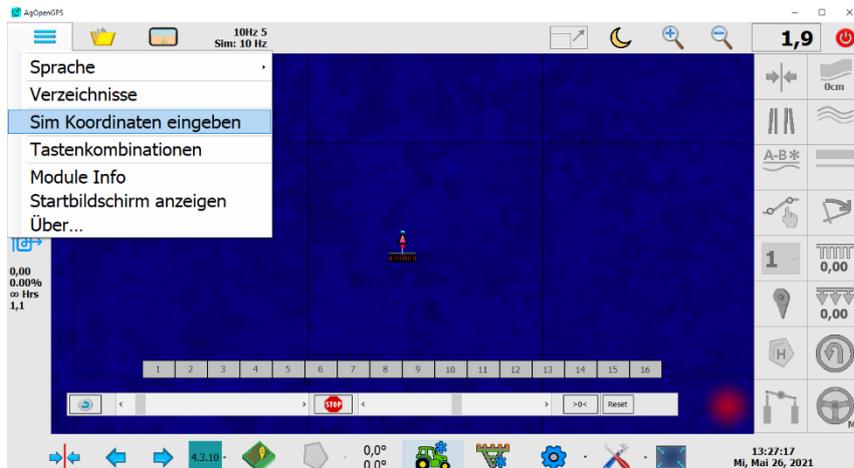


Abbildung 23: Screenshot Lokation Simulationskoordinaten

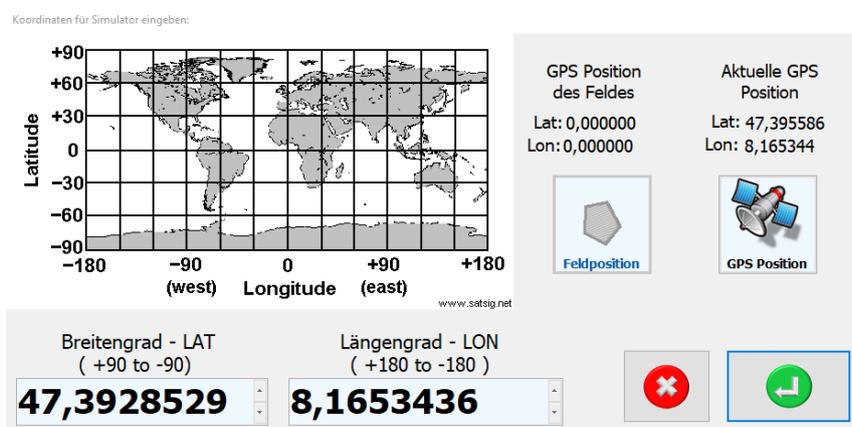


Abbildung 22: Koordinaten für Simulator

Damit AgOpenGPS die Fahrt simulieren kann, muss die Software eine Startposition erhalten. Im ersten Bild ist sichtbar, wo man diese verändern kann. Um den Startpunkt des Vehikels zu definieren, kann man sein KML in einem Editor öffnen. Dort sucht man dann nach `<MultiGeometry>` und findet darunter alle Koordinaten des Feldes. Die ersten Koordinaten überträgt man dann ins AgOpenGPS. So setzt AgOpenGPS das Simulierte Gefährt an diesen Punkt. Nach jeder Veränderung der Startposition muss das Programm neu gestartet werden.

13.2 NMEA Sätze

Die meisten Sätze wurden nach dem Standard NMEA 0183 gesendet. Die NMEA Sätze werden immer mit ihren Namen gestartet. Diese Namen setzen sich aus zwei Teilen zusammen. Der erste Teil definiert den Ursprung. Der zweite Teil, was für Informationen der Satz mit sich führt. Die ersten Teile, die in der AgOpenGPS-Software vorkommen sind:

- GP
GP gibt an, dass das Signal von einem GPS stammt.
- P
P steht für proprietär. Das heisst, dass dieses Format von einer Firma oder Organisation entwickelt wurde und nicht eine allgemeine Definition besitzt.
- GN
GN gibt an, dass das Signal von einem «Global Navigation Satellite System» (kurz GNSS) stammt
- Für PA wurde keine allgemeingültige Erklärung gefunden

All diese Abkürzungen wurden auf nmea.org[8] gefunden. Die folgenden Erklärungen ebenfalls[9] bis auf PAOGI, PTNL, GNTRA und PSTI. Dort wurde die Erklärung aus dem Quellcode genommen. Bei den Erklärungen von nmea.org steht jeweils <CR> und <LF>. Diese signalisieren das Ende des Satzes und starten einen neuen Satz.

13.2.1 GPGGA

```

          1         2         3 4         5 6 7 8 9 10 | 11 12 13 14 15
          |         |         | |         | | | | | | | | | |
$--GGA,hhmmss.ss,llll.ll,a,yyyyy.yy,a,x,xx,x.x,x.x,M,x.x,M,x.x,xxxx*hh<CR><LF>
```

Field Number:

- 1) Universal Time Coordinated (UTC)
- 2) Latitude
- 3) N or S (North or South)
- 4) Longitude
- 5) E or W (East or West)
- 6) GPS Quality Indicator,
0 - fix not available,
1 - GPS fix,
2 - Differential GPS fix
- 7) Number of satellites in view, 00 - 12
- 8) Horizontal Dilution of precision
- 9) Antenna Altitude above/below mean-sea-level (geoid)
- 10) Units of antenna altitude, meters
- 11) Geoidal separation, the difference between the WGS-84 earth ellipsoid and mean-sea-level (geoid), "-" means mean-sea-level below ellipsoid
- 12) Units of geoidal separation, meters
- 13) Age of differential GPS data, time in seconds since last SC104 type 1 or 9 update, null field when DGPS is not used
- 14) Differential reference station ID, 0000-1023
- 15) Checksum

Abbildung 24: Aufschlüsselung NMEA Satz GPGGA

13.2.2 GPVTG

```

      1  2 3  4 5  6 7  8 9
      |  | |  | |  | |  | |
$--VTG,x.x,T,x.x,M,x.x,N,x.x,K*hh<CR><LF>

```

Field Number:

- 1) Track Degrees
- 2) T = True
- 3) Track Degrees
- 4) M = Magnetic
- 5) Speed Knots
- 6) N = Knots
- 7) Speed Kilometers Per Hour
- 8) K = Kilometers Per Hour
- 9) Checksum

Abbildung 25: Aufschlüsselung NMEA Satz GPVTG

13.2.3 GPRMC

```

      1      2 3      4 5      6 7  8  9  10  11 | 12
      |      | |      | |      | |  |  |  |  |  |
$--RMC,hhmmss.ss,A,l111.11,a,yyyyy.yy,a,x.x,x.x,xxxx,x.x,a*hh<CR><LF>

```

Field Number:

- 1) UTC Time
- 2) Status, V = Navigation receiver warning, P = Precise
- 3) Latitude
- 4) N or S
- 5) Longitude
- 6) E or W
- 7) Speed over ground, knots
- 8) Track made good, degrees true
- 9) Date, ddmmyy
- 10) Magnetic Variation, degrees
- 11) E or W
- 12) Checksum

Abbildung 26: Aufschlüsselung NMEA Satz GPRMC

13.2.4 GPHDT

```

      1  2 3
      |  | |
$--HDT,x.x,T*hh<CR><LF>

```

Field Number:

- 1) Heading Degrees, true
- 2) T = True
- 3) Checksum

Abbildung 27: Aufschlüsselung NMEA Satz GPHDT

13.2.5 PAOGI

\$PAOGI,1,2,3,4,5,6,7,8,9,10,11,12,13

- (1 , 2) Fix taken at 1219 UTC
- (3 , 4) Latitude deg N
- (5, 6) Longitude deg E
- (7) Fix quality: 0 = invalid
 - 1 = GPS fix (SPS)
 - 2 = DGPS fix
 - 3 = PPS fix
 - 4 = Real Time Kinematic
 - 5 = Float RTK
 - 6 = estimated (dead reckoning) (2.3 feature)
 - 7 = Manual input mode
 - 8 = Simulation mode
- (8) Number of satellites being tracked
- (9) Horizontal dilution of position
- (10, 11) Altitude, Meters, above mean sea level
- (12) time in seconds since last DGPS update

- (13) Checksum

13.2.6 PTNL

\$PTNL,1,2,3,4,5,6,7,8,9,10,11,12

- 0 Message ID \$PTNL,AVR
- 1 UTC of vector fix
- 2 Yaw angle in degrees
- 3 Yaw
- 4 Tilt angle in degrees
- 5 Tilt
- 6 Roll angle, in degrees
- 7 Roll
- 8 Range in meters
- 9 GPS quality indicator:
 - 0: Fix not available or invalid
 - 1: Autonomous GPS fix
 - 2: Differential carrier phase solution RTK (Float)
 - 3: Differential carrier phase solution RTK (Fix)
 - 4: Differential code-based solution, DGPS
- 10 PDOP
- 11 Number of satellites used in solution
- 12 Checksum

13.2.7GNTRA

Hier wurde keine Beschreibung hinzugefügt. Bekannt aus dem Quellcode ist lediglich, dass dieses Format die Richtung (Heading) und die Rotation um die Fahrtrichtung (Roll) liefern.

13.2.8 PSTI

\$PSTI,032,033010.000,111219,A,R,-4.968,-10.817,-1.849,12.046,204.67,,,,,*39

- (1) 032 Baseline Data indicator
- (2) UTC time hhmss.sss
- (3) UTC date ddmmyy
- (4) Status:
 - V = Void
 - A = Active
- (5) Mode Indicator:
 - F = Float RTK
 - R = fixed RTK
- (6) East-projection of baseline, meters
- (7) North-projection of baseline, meters
- (8) Up-projection of baseline, meters
- (9) Baseline length, meters
- (10) Baseline course: angle between baseline vector and north direction, degrees
- (11) - (15) Reserved
- (16) Checksum

13.3 Datensatz Zeitmessungen Karten Laden

Tabelle 4: Zeitmessungen der Ladezeiten von Karten

Anzahl Punkte Berechnet	1000	2000	3000	4000	5000	10000	15000	20000	25000	30000
Dauer [ms]	509	867	1401	2165	3145	11207	25227	46312	72601	124066
	538	846	1407	2145	3140	11211	24573	44848	7218	125921
	509	834	1414	2164	3183	11281	24859	44365	72881	124070
	515	863	1496	2164	3153	11191	24700	44145	73602	
	522	835	1425	2174	3134	11271	24995	44281	73390	
	519	832								
	508	837								
	509	834								
	512	839								
	520	838								
Durchschnitt[ms]:	516.1	842.5	1428.6	2162.4	3151	11232.2	24870.8	44790.2	59938.4	124685.667
Durchschnitt[s]	0.5161	0.8425	1.4286	2.1624	3.151	11.2322	24.8708	44.7902	59.9384	124.685667
Durchschnitt[min]	0.00860167	0.01404167	0.02381	0.03604	0.05251667	0.18720333	0.41451333	0.74650333	0.99897333	2.07809444
Durchschnittliche Zeit pro Punkt	0.5161	0.42125	0.4762	0.5406	0.6302	1.12322	1.65805333	2.23951	2.397536	4.15618889

Anzahl Punkte Berechnet	40000	45000
Dauer [ms]	200010	226674
	195048	225321
	198982	
Durchschnitt[ms]:	198013.333	225997.5
Durchschnitt[s]	198.013333	225.9975
Durchschnitt[min]	3.3002222	3.766625
Durchschnittliche Zeit pro Punkt	4.95033333	5.02216667

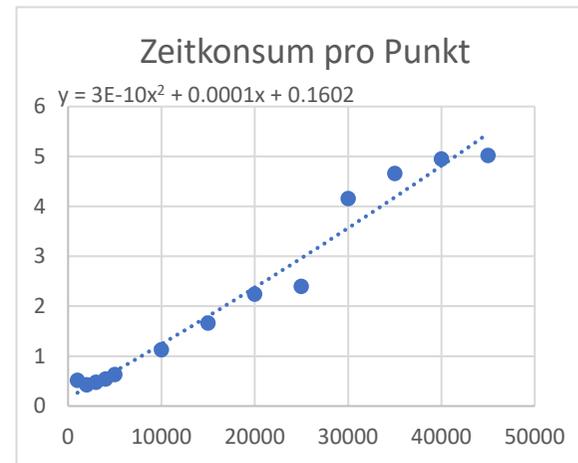


Abbildung 28: Darstellung Zeitkonsum pro Punkt beim Laden einer Karte