

[Robin Mantz]

[robin.mantz@einstein-motorsport.com]

Entscheidungs-dokumentation

[Telemetrie Software]

Einstein Motorsport

Inhalt

[1. Motivation 2](#_Toc527621137)

[1.1 Pro 2](#_Toc527621138)

[1.2 Contra 2](#_Toc527621139)

[2. Reglement 3](#_Toc527621140)

[3. Varianten 4](#_Toc527621141)

[3.1 C++ 4](#_Toc527621142)

[3.1.1 Pro 4](#_Toc527621143)

[3.1.2 Contra 4](#_Toc527621144)

[3.2 C# / Visual C# 4](#_Toc527621145)

[3.2.1 WPF vs. Winforms 4](#_Toc527621146)

[3.2.2 Pro 4](#_Toc527621147)

[3.2.3 Contra 4](#_Toc527621148)

[3.3 Python 4](#_Toc527621149)

[3.4 Kaufen 4](#_Toc527621150)

[3.4.1 Pro 4](#_Toc527621151)

[3.4.2 Contra 4](#_Toc527621152)

[3.5 C++ und JavaScript 5](#_Toc527621153)

[3.5.1 Konzept 5](#_Toc527621154)

[3.5.2 Pro 5](#_Toc527621155)

[3.5.3 Contra 5](#_Toc527621156)

[3.5.4 Bibliotheken 5](#_Toc527621157)

[4. Berechnungen 7](#_Toc527621158)

[4.1 Datenmengenabschätzung „Offline“ 7](#_Toc527621159)

[4.2 Datenmengenabschätzung „Online“ 7](#_Toc527621160)

[5. Tests 8](#_Toc527621161)

[5.1 Telemetrie Version 0.1 8](#_Toc527621162)

[5.2 [Hier kommen die aktuellen Tests zum Konzeptphase AL19] 9](#_Toc527621163)

[6. Auswertung der Berechnungen / Tests 10](#_Toc527621164)

[7. Entscheidung 11](#_Toc527621165)

# Motivation

Die Telemetrie soll einerseits „Live“ und „Offline“ zur Verfügung stehen. Die ursprüngliche Motivation hinter der Telemetrie kam durch den kapitalen Motorschaden des AL17 auf dem Hockenheimring während des Endurance. Durch eine Liveüberwachung der wichtigen Motordaten am Streckenrand und einem vorhandenen Fahrerfunk hätte man das Fahrzeug frühzeitig anhalten können und den Motor „retten“.

Zum AL14 wurde schon einmal eine Telemetrie entwickelt, die über WLAN und eine Handy-App funktionierte. In den Jahren 2008 und 2010 wurden schon Studienarbeiten zu diesem Thema durchgeführt. 2012 gab es eine Studienarbeit zu der Handy-App. Die bisherigen Systeme haben entweder nicht, oder nicht zufriedenstellend funktioniert.

Bei Einstein war ein sehr großes Problem der Vergangenheit immer die kurze Testphase. Es wurde auch schon ein direkter Zusammenhang zwischen der Länge der Testphase und dem Ergebnis auf den Events nachgewiesen. Die „Live“ Telemetrie soll primär die benötigten Testzeiten verringern bzw. effizienter machen und das Auto vor möglichen Totalausfällen schützen. Die „Offline“ Telemetrie soll zur Datenauswertung genutzt werden, in gleicher Art, wie WinDarab bisher genutzt wurde.

## Pro

* Verringerung der Testzeiten durch ein schnelleres Auswerten der Daten und ebenfalls ein effizienteres Auswerten, falls man dem Programm zur Datenauswertung genau auf den Testzweck maßgeschneiderte Auswertealgorithmen zur Verfügung stellt.
* Überwachung des Fahrzeugzustands nicht nur durch den Fahrer, der nicht dauerhaft alle Systemkritischen Variablen überwachen kann und auch kaum sinnvolle Zusammenhänge erkennen kann, sondern auch vom Streckenrand, wo man sich nur auf die Daten konzentrieren kann und zur Not über den Fahrerfunk eingreifen kann.
* Bessere „Offline“ Auswertung der Testdaten, durch ein eigens anpassbares Programm, das genau die geforderten Daten direkt in einer sinnvollen Visualisierung darstellen und ggf. schon verrechnen kann. Generell auch um mehr Spielraum in der Auswertung der Daten zu haben, anstatt sich an die Regeln der kostenlosen Version von WinDarab zu halten.
* Es existiert kein direkter Zeitdruck, da das Auto auch unabhängig von der Telemetrie fahren kann und somit bei Scheitern des Projekts nicht das Auto betroffen ist.
* Womöglich auch ins Internet „streambar“, sodass Alumnis und Mitglieder, die gerade nicht da sind auch die Daten verfolgen können.

## Contra

* Es ist ein hoher Engineering Aufwand nötig, durch die Notwendigkeit eigene Hardware und Software zu entwickeln.
* Die Telemetrie wird Platz im Auto und Gewicht beanspruchen. Sie soll vor dem front hoop platziert werden und die Platine allein ohne Antenne wiegt 49,5g. [Hier noch ein geschätzter Gesamtwert also mit Kabel, Antenne und Platine]
* Die Entwicklung einer Platine und der Kauf eines Funkmoduls sind natürlich mit Kosten verbunden, die womöglich über Sponsoring gedeckt werden sollten.

# Reglement

Im Reglement gibt es keine Regeln, die die Telemetrie betreffen würden.

# Varianten

## C++

### Pro

* Microcontroller wird in C programmiert => halbwegs einheitliche Sprachen
* Schnelligkeit, wenn professionell programmiert wird
* Vorhandene Kenntnisse bei allen Mitgliedern durch Elektrotechnik Studium HS Ulm
* Individualisierungsmöglichkeiten am höchsten durch assemblernähere Bibliotheken

### Contra

* Sehr großer Aufwand um gewünschte Ziele zu erreichen, durch große Individualisierungsmöglichkeiten und assemblernahe Programmierung
* Keine professionelle Programmierung vorhanden durch fehlende Informatiker und fehlende Erfahrung der Mitglieder
* Irrelevant, ob Microcontroller gleich programmiert ist, da die Projekte durch die Funkstrecke getrennt sind und nichts miteinander zu tun haben
* „Aussterbende“ Sprache, was GUI Anwendungen angeht und für Datenbearbeitung sehr kompliziert um es effizient zu gestalten
* Fehlende Bibliotheken und nicht „Industriestandart“ Entwicklungsumgebungen

## C#

### WPF vs. Winforms

<https://wpf-tutorial.com/about-wpf/wpf-vs-winforms/> [Ausarbeiten]

### Pro

* Industriestandart
* Vorhandene Bibliotheken
* „Höhere Sprache“ also weniger Zeitintensive und „Einfachere“ Programmierung
* Mehr Support im Internet in jeder Hinsicht

### Contra

* Zeitaufwand zum Einlesen bzw. Auswählen und Testen der neuen Bibliothek bzw. Sprache
* Grundsätzlich Ressourcenaufwändigere Programmierung als C++
* Zeitaufwand durch „Umschreiben“ der bisher vorhandenen Software

## Python

Es ist zu überlegen ob man die Datenberechnungen mit Python umsetzt, falls man die Visualisierung der Daten über JavaScript und Websockets laufen lässt, um komplett Plattformunabhängig zu sein. Da wir zum Zeitpunkt des AL19 einen erfahrenen Python Programmierer im Team haben, könnte sich das womöglich anbieten.

### Pro

### Contra

## Kaufen

### Beispiele

<https://www.plex-tuning.com/>

<https://www.race-technology.com/za/racing/products/data-loggers/data-loggers_rt-live>

### Pro

* Man hätte ein fertiges System, das einwandfrei funktioniert ohne Aufwand

### Contra

* Alle kaufbaren vernünftigen Systeme sind sehr teuer
* Man hat immer noch nicht mehr Freiraum für eigens programmierte Auswertungen als bisher mit WinDarab für die „Offline“ Telemetrie
* Vermutlich nicht so gewichtsoptimiert, wie eine eigens entwickelte es sein könnte

## C# und JavaScript

### Konzept

Das Fahrzeug sendet die Daten in Echtzeit an den Server über Funk, der ein richtiger Server oder auch ein Client mit Server im Hintergrund sein kann. Andere Clients können über http eine Website vom Server laden, der dann eine WebSocket-Verbindung aufbaut, um die Daten in Echtzeit an den Client zu schicken. Der Server stellt die Daten mit einer C#-Anwendung bereit, auf den Clients erfolgt die Ausgabe mit JavaScript im Webbrowser als Rich Internet Application. Ob der Server nun ein Laptop am Streckenrand werden wird, oder ein Raspberry Pi mit Powerbank, der mittig an der Stecke liegt, muss in den späteren Reichweitetests festgestellt werden. Grundsätzlich ist beides möglich.

### Pro

* Viele Clients können Daten von einem Gerät abrufen
* Nur eine Antenne nötig, restliche Kommunikation über herkömmliche Techniken
* Nur ein Gerät benötigt Sichtkontakt
* Plattformunabhängig bei Clients (Mac, Linux, Windows, Android)
* Es muss nur auf dem Server Software installiert werden
* Gut designbar, macht optisch was her
* Leichter an mehrere Entwickler aufteilbar als rein C#
* Aktuell schneller zu entwickeln als C#

### Contra

* Single-Point-of-Failure bei Serverausfall
* Eventuell nur auf Live-Telemetrie anwendbar
* Eventuelle Performance-Einbußen

### Bibliotheken

* D3.js Data Driven Documents
  + Bibliothek, um im GUI performante und gut anpassbare Graphen zu generieren.
  + <https://d3js.org/>
  + Lizenz privat und kommerziell frei nutzbar <https://github.com/d3/d3/blob/master/LICENSE>
* SuperWebSocket
  + Bibliothek, um auf dem Server über kurze Latenzzeiten Daten mit Hilfe von WebSocket an den Client schicken zu können. .NET hat nativ hier auch schon Möglichkeiten, daher kann darauf eventuell verzichtet werden, jedoch bietet sich die Bibliothek gerade am Anfang an, da sie schnell einsatzbereit ist und einfach zu verstehen.
  + <https://archive.codeplex.com/?p=superwebsocket>
  + Lizenz privat und kommerziell frei nutzbar <https://github.com/kerryjiang/SuperWebSocket/blob/master/LICENSE.TXT>

# Berechnungen

## Datenmengenabschätzung „Offline“

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| Anzahl der Variablen | Lograte in ms |  | 1s | 1min | 10min | 30min | 60min |
| 200 | 20 | Anzahl der entstandenen Datenpakete nach: | 10000 | 600000 | 6000000 | 18000000 | 36000000 |

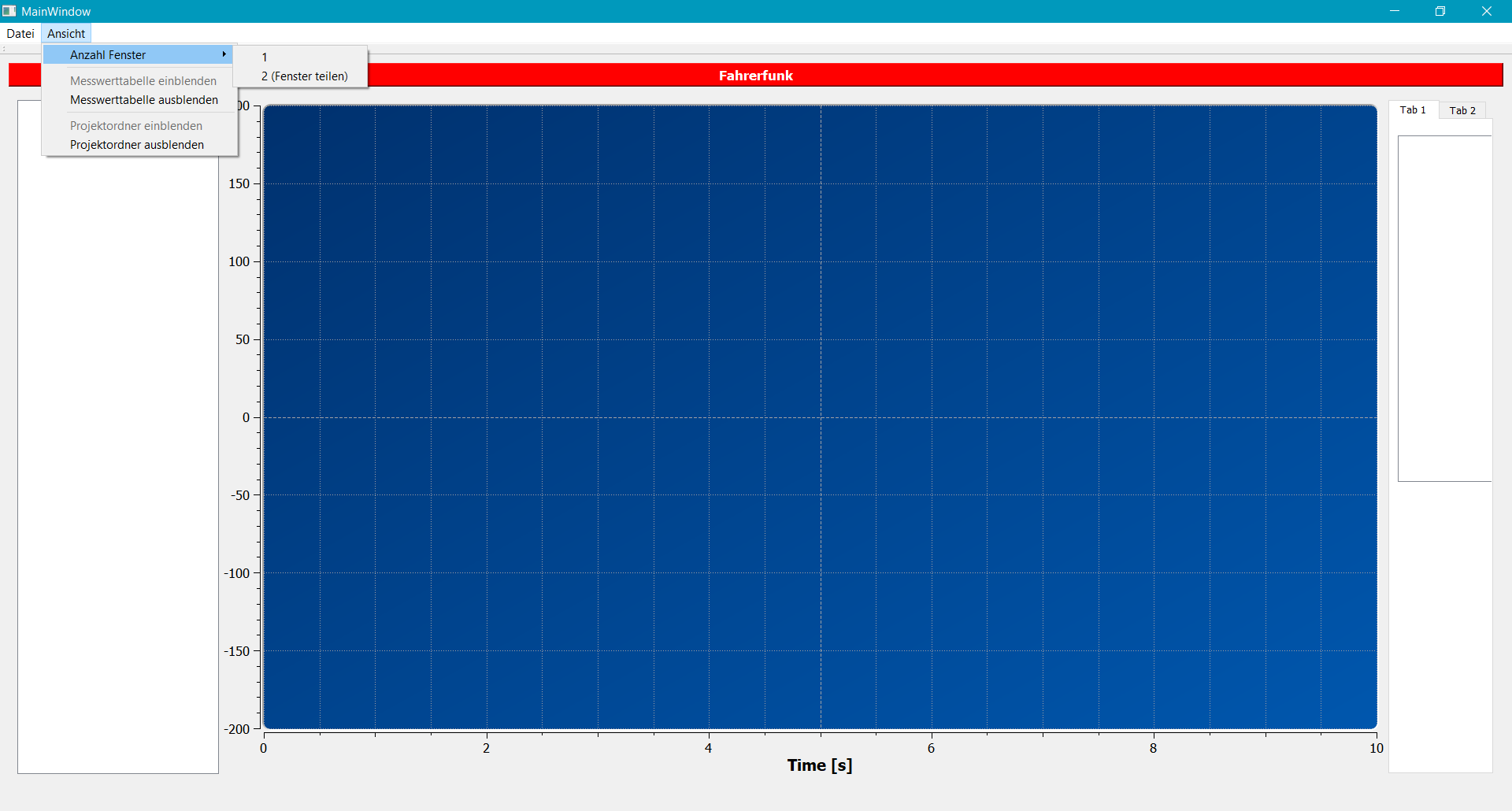
## Datenmengenabschätzung „Online“

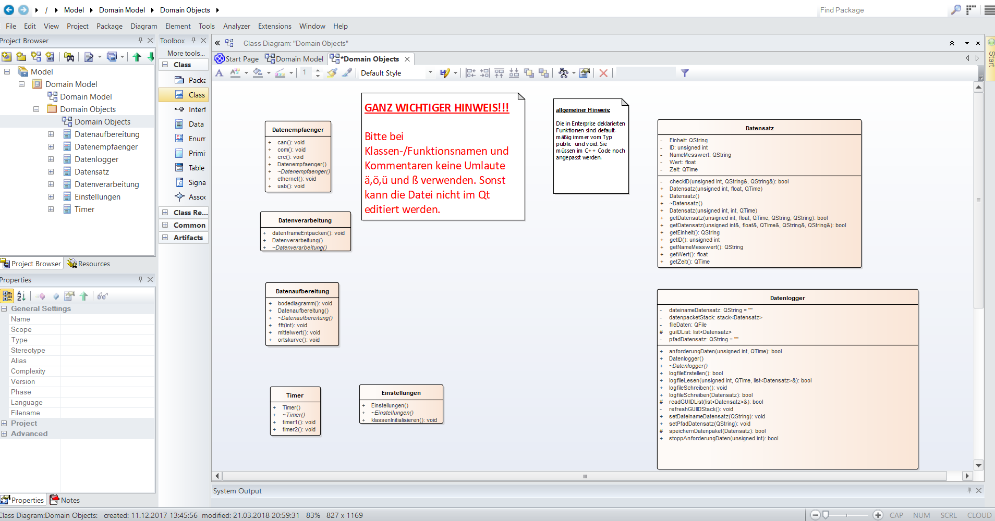
|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| Anzahl der Variablen | Datenrate in ms |  | 1s | 1min | 10min | 30min | 60min |
| 10 | 100 | Anzahl der entstandenen Datenpakete nach: | 100 | 6000 | 60000 | 180000 | 360000 |

# Tests

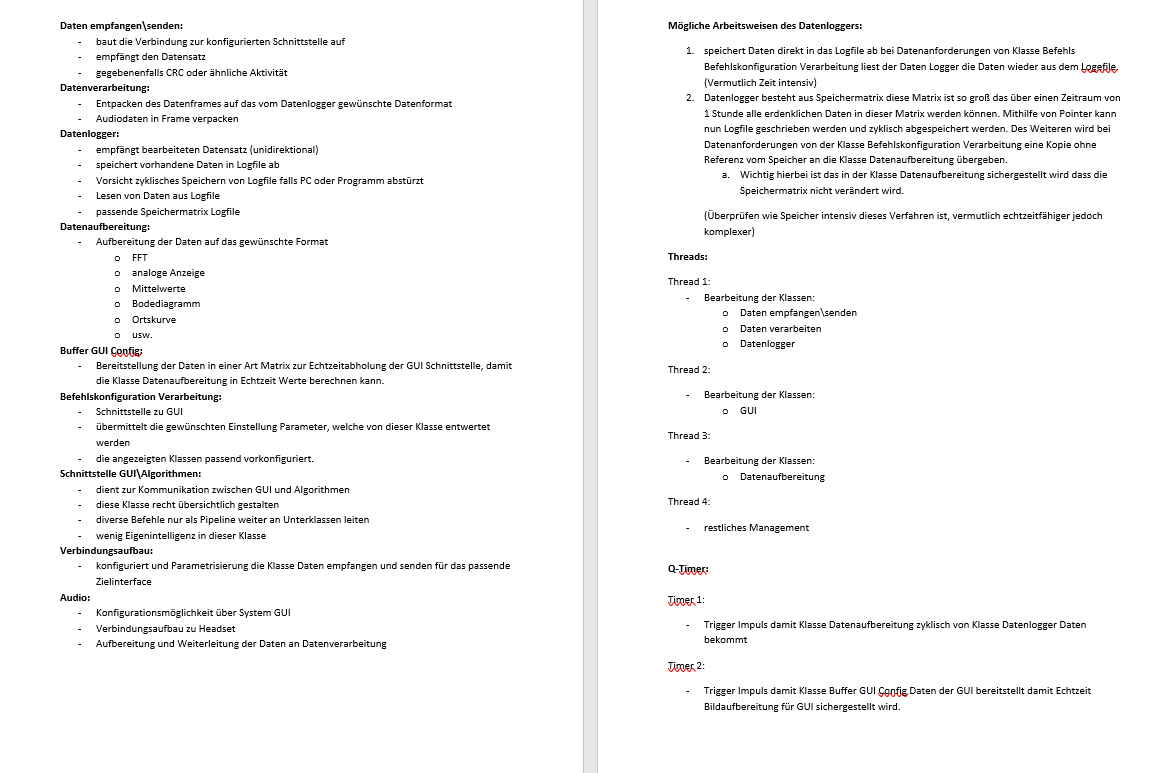
## Telemetrie Version 0.1

Die Software für die Telemetrie wurde mit dem Al18 schon in einer ersten Version mit C++ entwickelt. Die zugehörige Hardware, mit einem STM32F4 Prozessor von ST und einem AMB8826 868MHz Long Range Funkmodul von Amber Wireless, wurde ebenfalls mit dem AL18 fertig gestellt, jedoch noch nicht getestet. Die Software wurde hauptsächlich von Alexander Stenke und Robin Mantz geschrieben. Dabei wurde schnell klar, dass man mit den amateurhaften Programmierskills aus der Vorlesung schnell an die Grenzen von C++ stoßen würde. Es wurde QT als Programmierumgebung gewählt und versucht, die Visualisierung mit der Qwt Bibliothek zu realisieren. Mit QT war man ebenfalls nicht wirklich zufrieden.



Es wurde versucht eine Klassenstruktur mit Hilfe von „Enterprise Architekt“ zu erstellen, was anfangs auch so funktioniert hatte, dann aber nicht mehr weitergeführt wurde. Das Tool ist sehr nützlich beim Erstellen von größeren Softwarestrukturen. Wie wir Enterprise Architekt in der V2 des AL19 verwenden werden bzw. ob wir es überhaupt verwenden werden, ist noch ungeklärt. Die Bilder sollten dieser Dokumentation beiliegen, für näheres Interesse.

Das erste Ergebnis der Klassenstruktur in Worten ist dem nachfolgenden Bild zu entnehmen. Dies war jedoch nur **vorläufig**.



## WPF

### LiveCharts

Eine vielversprechende Bibliothek, die vermutlich auch verwendet werden kann, um mit Wpf (Windows Presentation Foundation) Graphen für unseren Gebrauch darzustellen.

<https://lvcharts.net/>

[Bilder und Eindrücke der ersten LiveCharts Tests]

# Auswertung der Berechnungen / Tests

# Entscheidung