## Einleitung

## Mechanik

## Elektronik

## Software

## Abstract

## Zusammenfassung

Es soll eine multiplattformfähige Software (Linux, macOS, Windows) entwickelt werden, die die Daten des Prüfstands für den Benutzer grafisch ausgibt. Die Bedienung sollte der des alten Programms ähneln, ein Diagramm soll wie gehabt nach der abgeschlossenen Messung angezeigt werden, das ausgedruckt und abgespeichert werden kann.

Die Software für den µC soll alle Sensordaten einlesen und über ein eigenes Protokoll an die GUI-Applikation übermitteln.

## Aufgabenstellung

Ziel ist es, eine Software für den Benutzer zur Bedienung des Prüfstandes zu schreiben. Diese soll die Messung ausführen und automatisiert steuern, d.h. der Benutzer soll ab dem Zeitpunkt der Messung nur mehr das Motorrad bedienen müssen. Nach abgeschlossener Messung sollen die Messdaten verarbeitet und in einem Diagramm ausgegeben werden.

Der µC soll die Messwerte der Sensoren auswerten und auf Anfrage der GUI-Applikation an diese übertragen. Das System muss echtzeitfähig sein und das Intervall zwischen den Messpunkten soll sich auf ein Minimum beschränken.

## Zeitplan

## Variantenstudien

## GUI-Applikation

## Plattform

Folgende Varianten habe ich in Betracht gezogen:

* App für Smartphone & Tablet
* Stationärer Touch-Screen mit angeschlossenem Raspberry Pi
* Applikation für PC

## Smartphone & Tablet

Mobile Geräte werden immer beliebter und leistungsstärker, eine Variante besteht im Entwickeln einer App für Android und iOS.

*Vorteile:*

* Mobil
* Schnelle Inbetriebnahme

*Nachteile:*

* Es müssten zwei Apps geschrieben werden (Android und iOS)
* iOS AppStore-Veröffentlichung kostenpflichtig

## Raspberry Pi

Eine weitere Möglichkeit besteht in der Montage eines stationären Touchscreens, der über einen Raspberry Pi das Programm ausführt.

*Vorteile:*

* Einfache Handhabung für den User
* Schnelle Inbetriebnahme

*Nachteile:*

* Begrenzte Leistung
* Zusätzliche Kosten des Systems

## PC

*Vorteile:*

* Hohe Performance
* Großer Monitor

*Nachteile:*

* Eventuell langsame Inbetriebnahme
* Nicht handlich

## Programmiersprache

Im Falle einer App für Smartphone und Tablet standen Java (für Android), Objective-C und Swift (für iOS) zur Auswahl. Für iOS wäre meine Wahl auf Swift gefallen, allerdings hätte dies eine hohe Einarbeitungszeit erfordert, da diese Sprache in der Schule nicht gelehrt wurde. Zusätzlich besteht das Problem, dass Swift noch recht jung ist und es daher noch relativ schwierig ist, für Probleme im Internet Lösungen zu finden.

Für eine PC-Applikation standen C, C++ und Java zur Verfügung. C wäre nicht geeignet, da diese Sprache nicht objektorientiert ist. Java hat den Vorteil, dass das Programm multiplattformfähig ist, sprich auf den drei wichtigsten Betriebssystemen – Linux, macOS und Windows – ausführbar sein wird. Zusätzlich gestaltet sich das Designen der GUI mit Hilfe des Editors für Java Swing in der Netbeans IDE als relativ einfach.

## µC-Software

## Entwicklungsboard

## Anforderungen an den µC

|  |  |
| --- | --- |
| Sensor | Benötigte Peripherie |
| Motordrehzahl | Digitaler Eingang (ext. Interrupt) |
| Walzendrehzahl | Digitaler Eingang (ext. Interrupt) |
| Motortemperatur (Thermoelement) | Analoger Eingang |
| Abgastemperatur (Thermoelement) | Analoger Eingang |
| Umgebungswerte (BMP-180) | I2C oder SPI |

Tabelle 1: Anforderungen an den µC

Zusätzlich wird noch mindestens ein UART zur Kommunikation mit dem PC benötigt. Ein optionaler weiterer UART-Port würde als Debugging-Schnittstelle verwendet werden.

## Arduino MEGA & Arduino DUE

*Vorteile:*

* Hohe Rechenleistung
* Großer Speicher
* 4x UART
* 2x I2C
* CAN-Bus (nur DUE)

*Nachteile:*

* Großer Platzbedarf
* Teuer

Abbildung 1: Arduino MEGA & Arduino DUE, Quelle: Eigene Fotografie

* 3,3V Pegel durch ARM-Architektur (nur DUE)

## Arduino UNO & Arduino NANO

*Vorteile:*

* Klein
* Günstig
* µC tauschbar (nur UNO, THT-Edition)

*Nachteile:*

* Nur ein UART
* Kleinerer Speicher

Abbildung 2: Arduino UNO & Arduino NANO, Quelle: Eigene Fotografie

* Geringere Performance

## Kommunikationsmethoden

## Ethernet

Mit den sogenannten „Ethernet-Shields“ oder den speziellen Arduino Ethernet Boards ist es möglich mit einem Arduino Daten über das Netzwerk zu übertragen. Dadurch spart man sich das Verbinden mit einem USB-Kabel, es setzt aber eine Ethernet-Leitung zur Elektronik und eine Verbindung des Computers ins Netzwerk voraus.

Abbildung 3: Arduino Ethernet, Quelle: Eigene Fotografie

## UART

Deutlich einfacher lässt sich die Kommunikation mit UART lösen. Für Java gibt es Bibliotheken, mit denen das Realisieren der Kommunikation einfach vonstatten geht. Auch ist man bei einer UART-verbindung nicht an die Netzwerkgeschwindigkeit gekoppelt, bzw. im Falle von WiFi an die Signalstärke.

## Programmiersprachen

Die verbreitetsten Programmiersprachen zur µC Programmierung sind C und Assembler.

Eine weitere Option besteht mit der „Arduino-Language“, die sich als C++ mit der eingebundenen *Arduino.h*-Library versteht und das Programmieren von µCs deutlich vereinfachen soll.

## Zusammenfassung

Als Plattform für die PC-Applikation wurde der PC gewählt, da er die höchste Leistung bietet und im Gegensatz zur mobilen Lösung nur eine Anwendung geschrieben werden muss.

Als Sprache dient Java um die Multiplattformfähigkeit zu gewährleisten. Ein weiterer Vorteil: Sollten Einplatinencomputer wie der Raspberry Pi zukünftig in der Lage sein die erforderliche Leistung zu bringen, wird es möglich sein, das Java-Programm auf diesen auszuführen.

Die Wahl des Entwicklungsboard ist auf den Arduino UNO Rev.3 gefallen, da die Spezifikationen des darin verbauten ATmega328p für unsere Zwecke ausreicht. Der NANO, der den selben µC verbaut hat, wurde nicht verwendet, da der Formfaktor des UNO besser zu handeln ist.

Als Kommunikationsmethode wurde UART gewählt, da die einwandfreie Kommunikation über das Netzwerk von der Geschwindigkeit, bzw im Falle von W-LAN von der Signalstärke abhängt. Unser Programm setzt aber eine sehr schnelle Kommunikation voraus. Außerdem würde ein solches System keine weiteren Vorteile bringen, da sich im Normalfall Prüfer und Prüfstand im selben Raum befinden.

Programmiert wird der µC mit C++ und der Arduino.h-Library, da die Programmierung der GUI-Applikation sehr aufwendig werden wird und ich hier Zeit sparen möchte. Dies bietet auch den Vorteil, dass Bibliotheken von Sensor-Herstellern (zB von Adafruit für den BMP-180) eingebunden werden können, die die Programmierung noch weiter vereinfachen.

## PC-Applikation

## UML-Diagramm (vereinfacht)

Zur Übersichtlichkeit wurden JFrame-Objekte (JButton, JMenu, etc) entfernt, wie auch die von der abstrakten Klasse *Request* abgeleiteten Klassen. In der Klasse *Config* wurden die Getter und Setter entfernt.

## GUI-Klassen

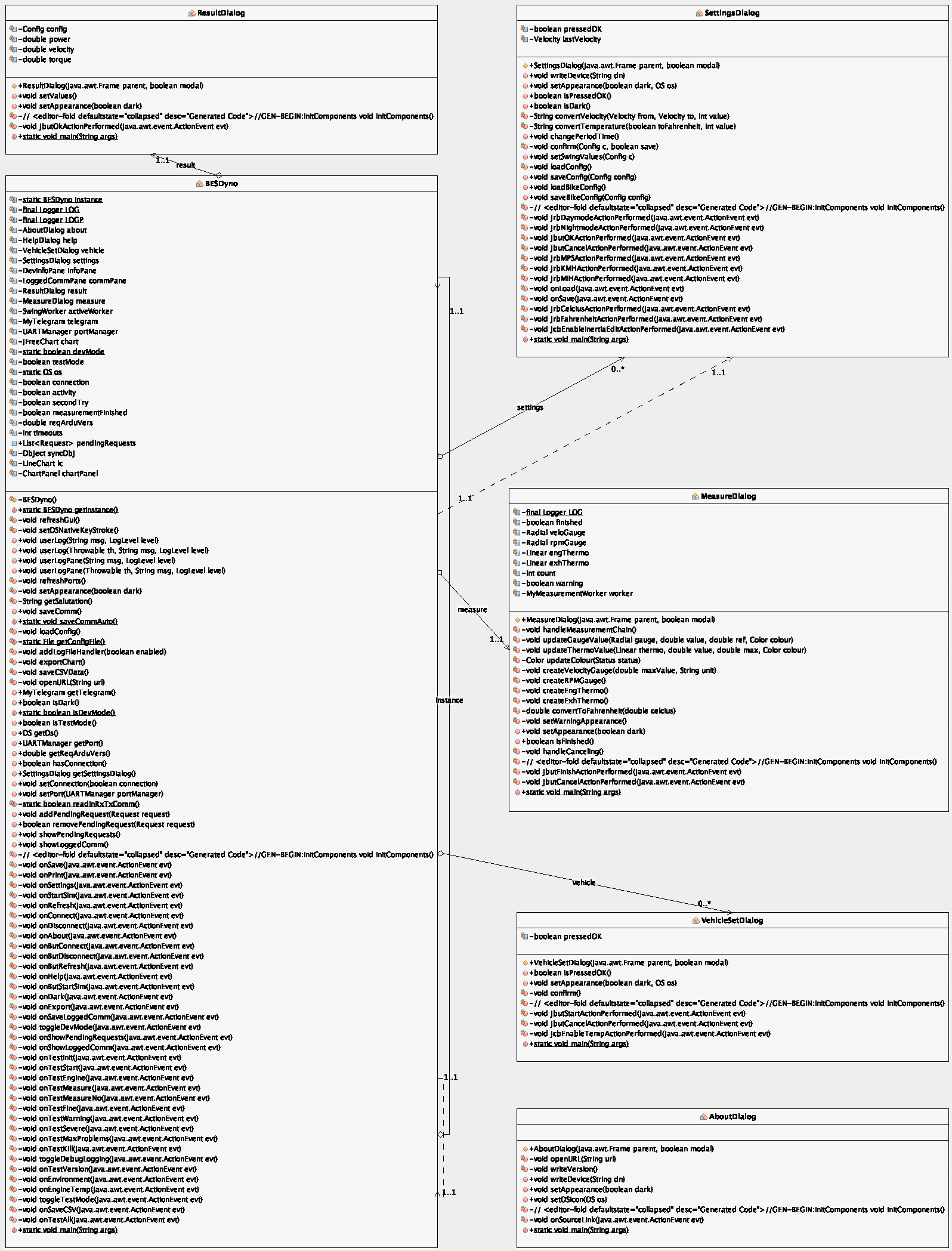


Abbildung 4: UML-Diagramm der GUI-Klassen, Quelle: Generiert mit easyUML

## Datenerhaltungsklassen

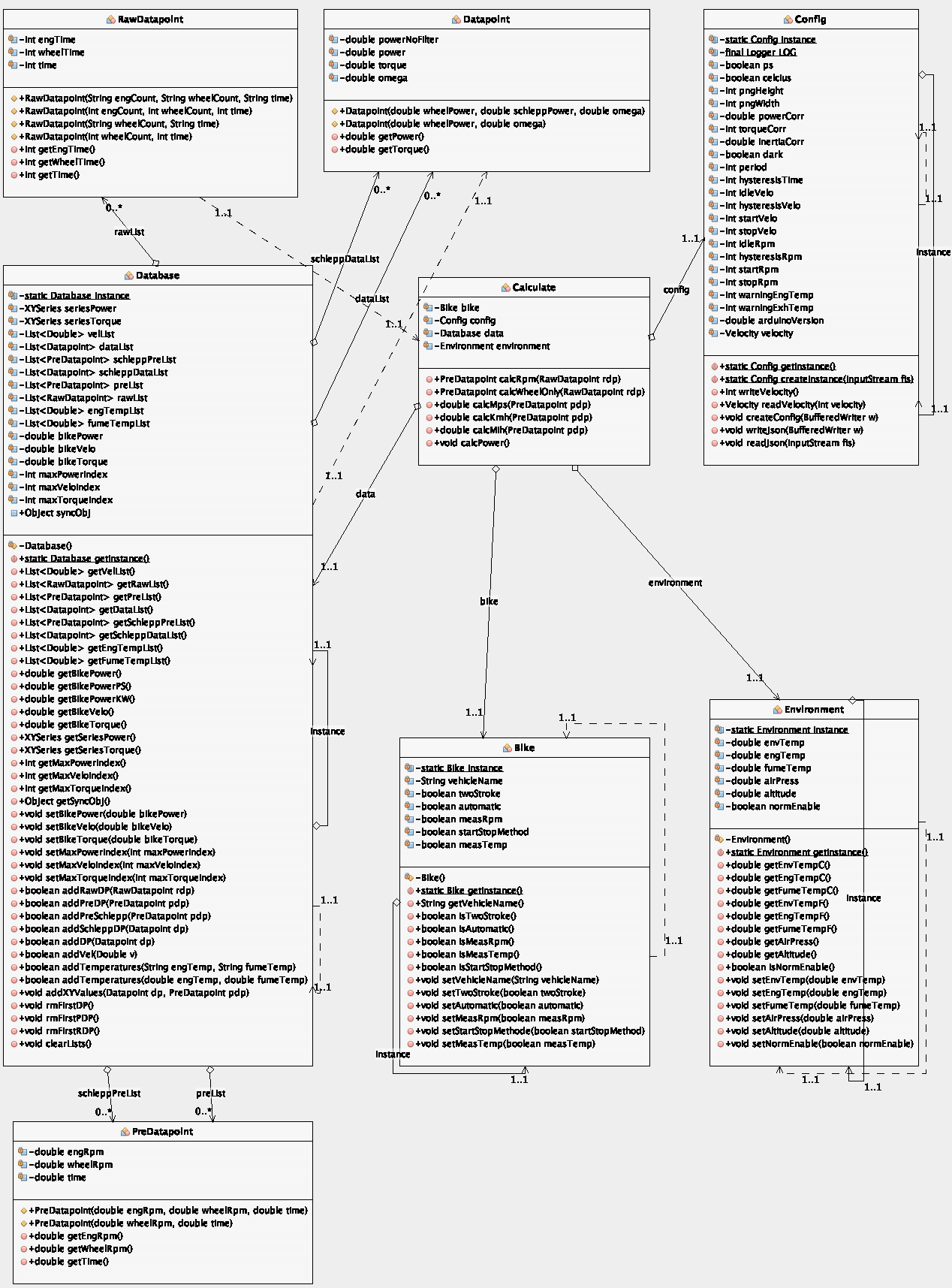


Abbildung 5: UML-Diagramm der Datenerhaltungsklassen, Quelle: Generiert mit easyUML

## Klassen zur Kommunikation und Berechnung

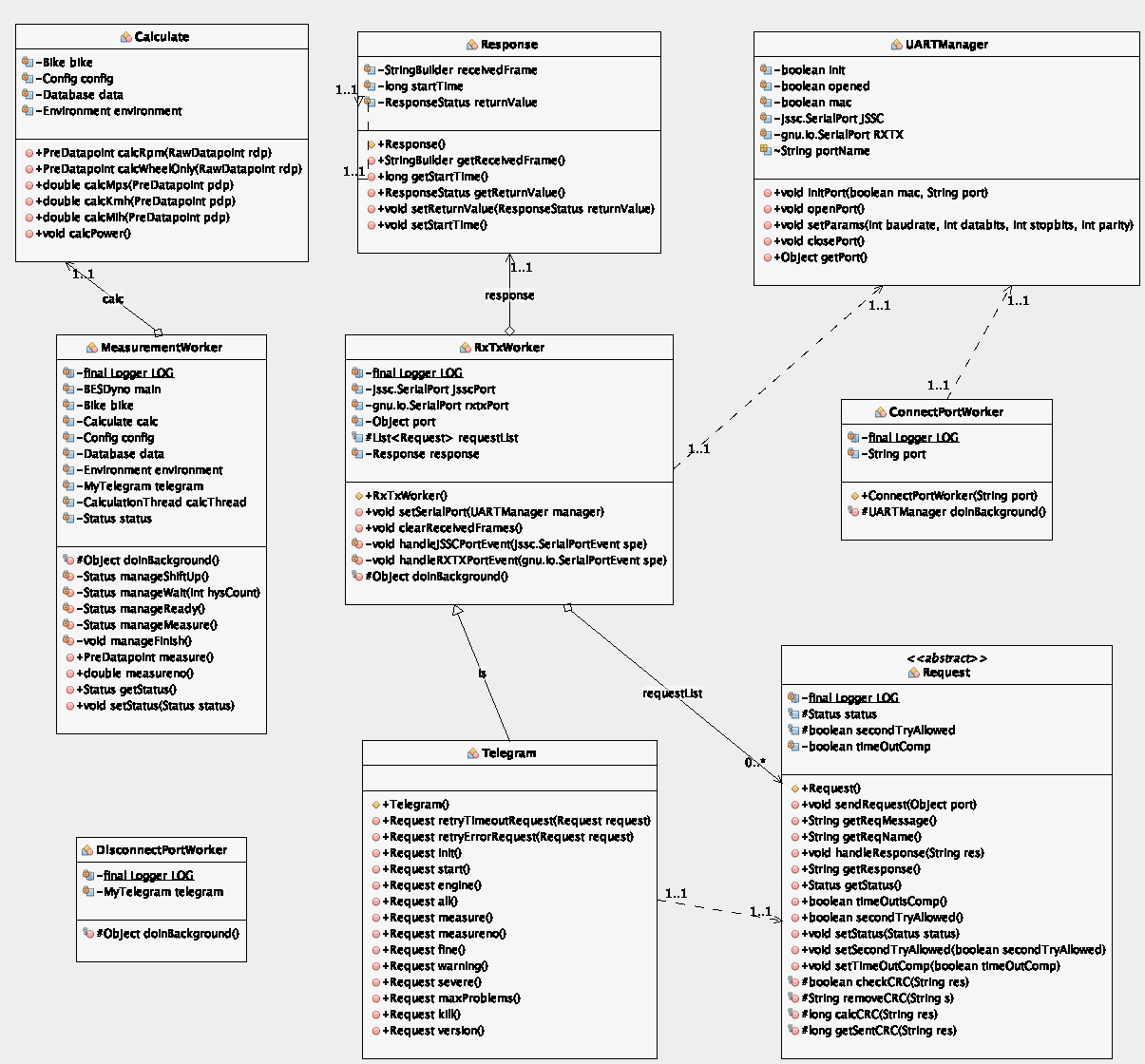


Abbildung 6: UML-Diagramm der Klassen für Kommunikation und Berechnung, Quelle: Generiert mit easyUML

## Graphical User Interface

## About



Abbildung 7: AboutDialog, Quelle: Screenshot

Jedes Programm sollte einen About-Dialog besitzen, in dem allgemeine Informationen über das Programm, rechtliches und Copyright-Informationen Die jLabels wurden mit HTML-Tags ausgestattet, die eine Formatierung der Labes erlauben. Am Anfang wird mit Tag <html> geöffnet und am Ende mit </html> geschlossen. Mit <b> lässt sich eine Textstelle fett darstellen und mit <br> fügt man einen Zeilenumbruch ein.

Der jButton „Quellcode“ öffnet im Standardbrowser einen Link zum GitHub-Repository des Projekts.

## Hauptfenster

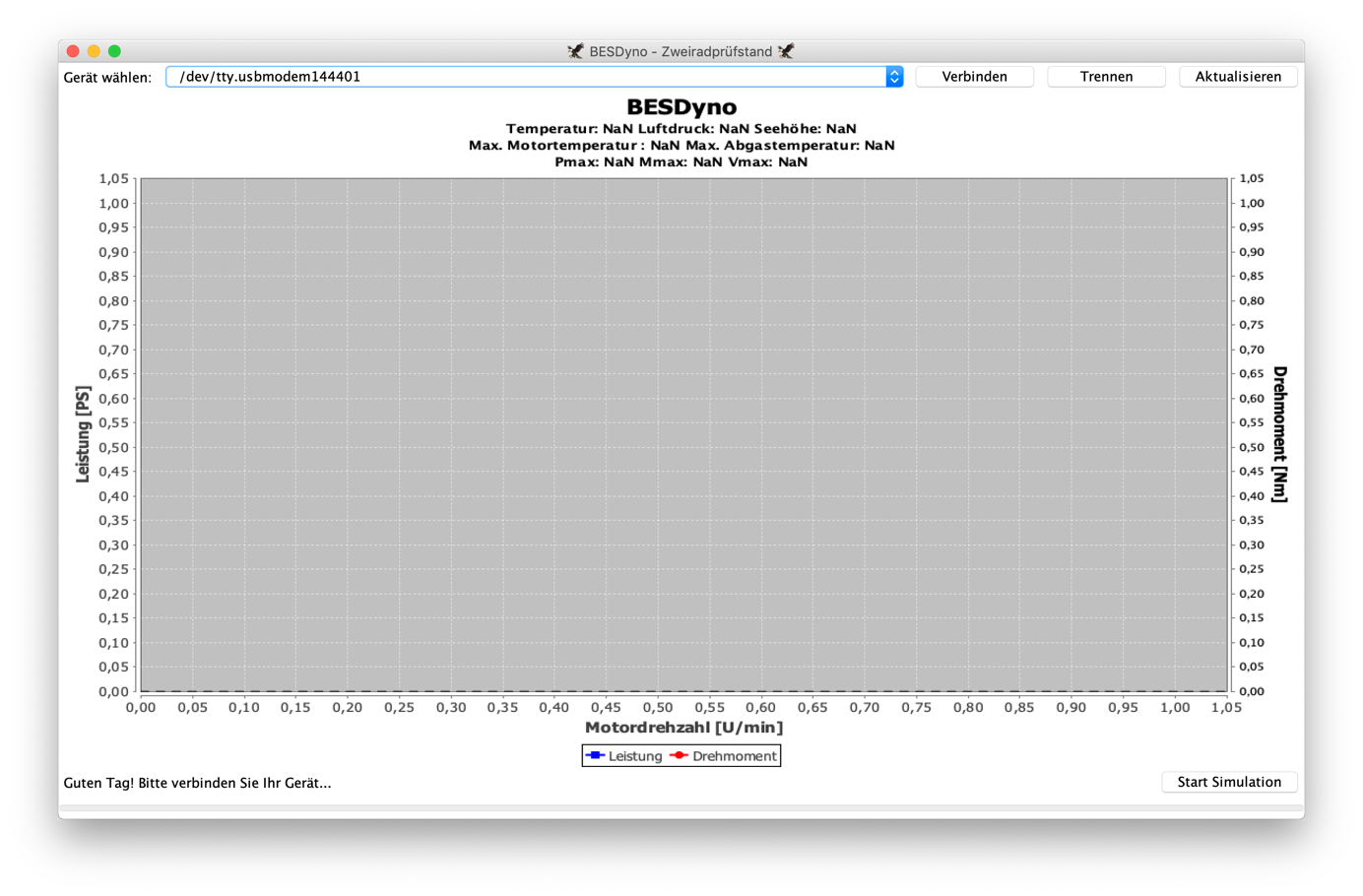


Abbildung 8: Main-GUI, Quelle: Screenshot

Wenn das Programm startet, wird dieses Fenster als erstes angezeigt.

In der oberen Leiste kann man sich mit seinem Endgerät verbinden und unten wird ein Status-Text in einem jTextField angezeigt. Die eigentliche Bedienung findet über das Menü in der Titelleiste statt, die den Vorteil bietet, dass mit Tastaturkürzeln gearbeitet werden kann.

In der Mitte wird beim Start ein leeres Diagramm angezeigt, dass nach Abschluss der Messung mit Daten gefüllt wird.

## Einstellungen

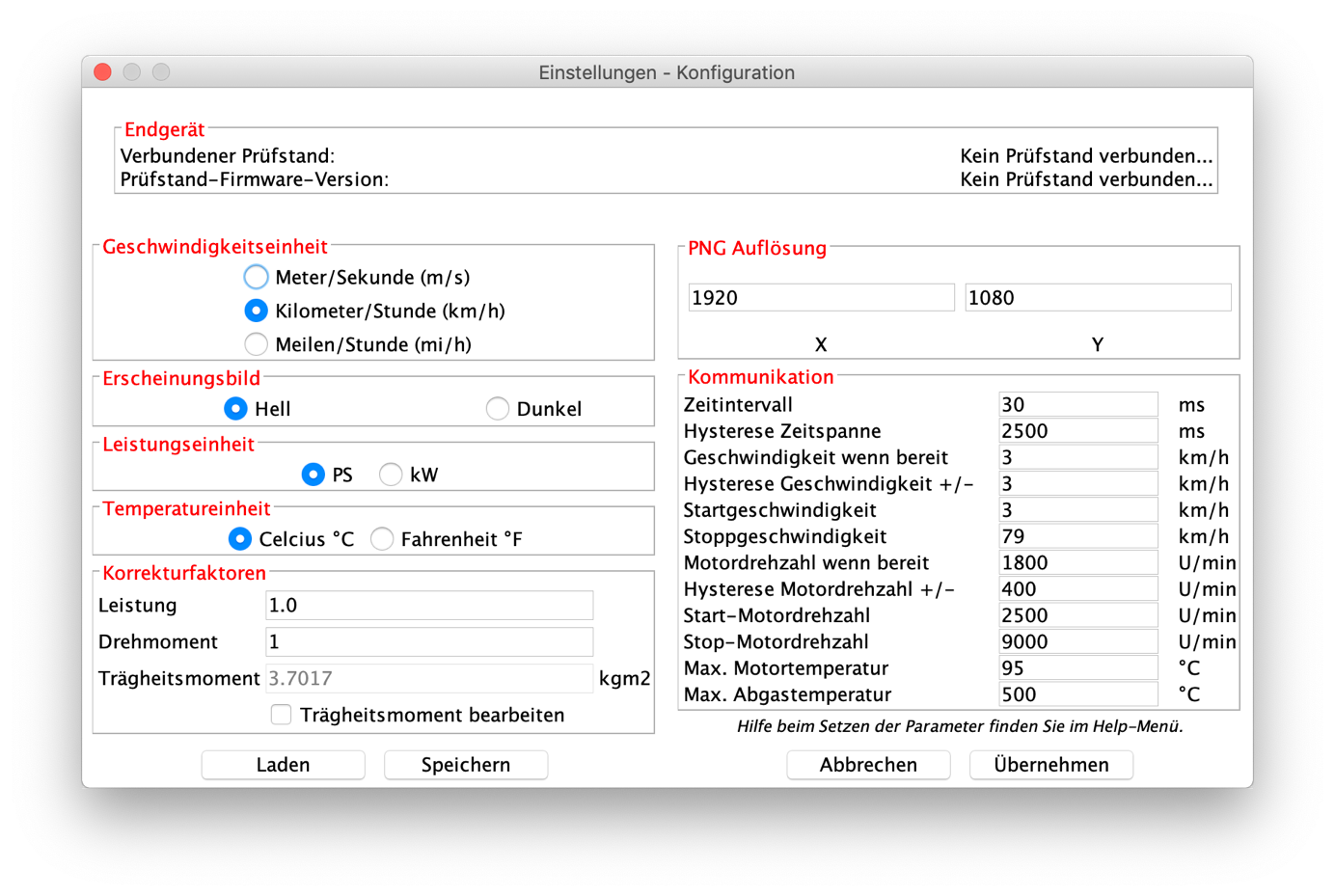


Abbildung 9: SettingsDialog, Quelle: Screenshot

*Im Einstellungsfenster werden für die Messung wichtige Parameter festgelegt:*

|  |  |
| --- | --- |
| Parameter | Funktion |
| Zeitintervall | Zeit, die zwischen zwei Messpunkten vergehen soll. Bei leistungsstarken Rechnern kann sie sehr klein sein. |
| Hysterese Zeitspanne | Gibt an, wie lange das Einpendeln bei einer bestimmten Drehzahl dauern soll. |
| Geschwindigkeit wenn bereit | Bei dieser Geschwindigkeit pendelt sich das Programm ein. (Leerlaufgeschwindigkeit) |
| Hysterese Geschwindigkeit | Einpendeln: Leerlaufgeschwindigkeit ± Hysterese Geschwindigkeit |
| Startgeschwindigkeit | Ab dieser Geschwindigkeit werden Daten für das Diagramm erfasst. |
| Stoppgeschwindigkeit | Ab dieser Geschwindigkeit stoppt die Messung. |
| Motordrehzahl wenn bereit | Bei dieser Drehzahl pendelt sich das Programm ein. (Leerlaufdrehzahl) |
| Hysterese Motordrehzahl | Einpendeln: Leerlaufdrehzahl ± Hysterese Motordrehzahl |
| Start-Motordrehzahl | Ab dieser Drehzahl werden Daten für das Diagramm erfasst. |
| Stop-Motordrehzahl | Ab dieser Drehzahl stoppt die Messung. |
| Max. Motortemperatur | Überschreitet während der Messung der Motor diesen Wert, gibt es eine Warnung. |
| Max. Abgastemperatur | Überschreitet während der Messung der Krümmer diesen Wert, gibt es eine Warnung. |

Tabelle 2: Einstellungs-Parameter

*Anmerkung: Wenn die Motordrehzahl erfasst wird, bezieht sich das Programm nur auf die eingegeben Drehzahlen.*

## Mess-Fenster

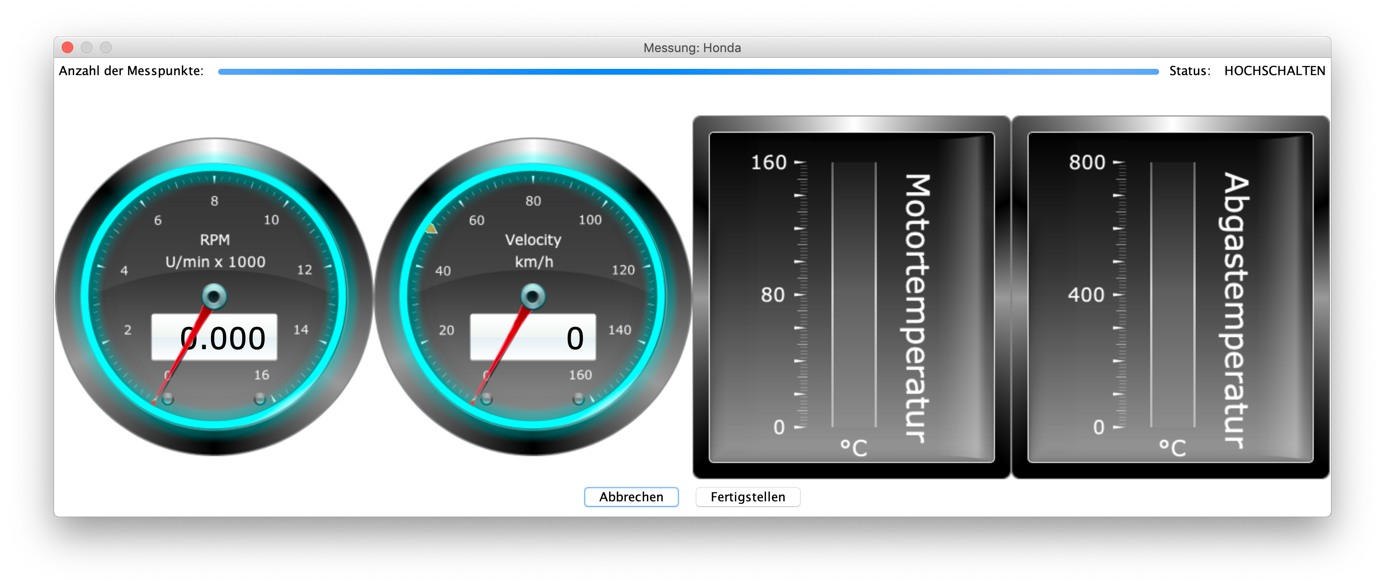


Abbildung 10: MeasureDialog mit Temperaturmessung, Quelle: Screenshot



Abbildung 11: MeasureDialog ohne Temperaturmessung, Quelle: Screenshot

Bevor man zu diesem Fenster gelangt, muss noch der Name des Motorrads, die Taktanzahl des Motors und ob man Motordrehzahl und die Temperaturen messen möchte, eingegeben werden.

Danach gelangt man zum MeasureDialog der die aktuelle Drehzahl des Motors und die aktuelle Geschwindigkeit des Walze anzeigt. Die Aufgabe des Nutzers wird textuell über den Tachos und farblich (Blau – im Bild zu sehen, Gelb, Grün, Rot) ausgegeben.

Wenn die Temperatur währen des gesamten Messvorgangs gemessen wird, überprüft das Programm, ob sie die im Einstellungsdialogfenster eingegebenen Werte übersteigt. Ist dies der Fall, färbt sich als Warnung die gesamte Benutzeroberfläche rot.

Nach abgeschlossener Messung wird ein Dialogfenster mit den Maximalwerten des Motorrads angezeigt. Wenn dieses geschlossen wird, gelangt man wieder zurück zum Hauptfenster und das Diagramm wird mit den gemessenen Werten gefüllt.

Die Tachometer/Thermometer passen sich je nach Einheit (m/s, mi/h, km/h bzw. °C, °F) in ihrer Skalierung an.

## Menü in der Titelleiste

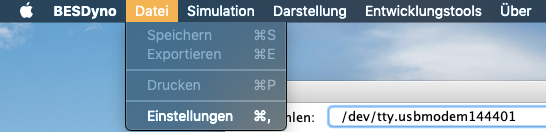


Abbildung 12: Datei-Menü, Quelle: Screenshot

Über den Menü-Punkt „Datei“ können die Werte als CSV-Datei gespeichert, das Diagramm exportiert oder gedruckt, sowie das Einstellungsdialogfenster aufgerufen werden.



Abbildung 13: Simulation-Menü, Quelle: Screenshot

Über den Menü-Punkt „Simulation“ kann die Simulation gestartet und die Umweltdaten bzw. Motorradtemperaturen aktualisiert werden. Unter dem jSeperator wird die Verbindung mit dem Endgerät gesteuert.



Abbildung 14: Darstellung-Menü, Quelle: Screenshot

Im Menü „Darstellung“ gibt es nur den Punkt „Dark-Mode“ mit dem die gesamte Oberfläche auf ein dunkles Theme geschaltet wird. Die Hintergründe werden Dunkelgrau, der Text wird weiß.

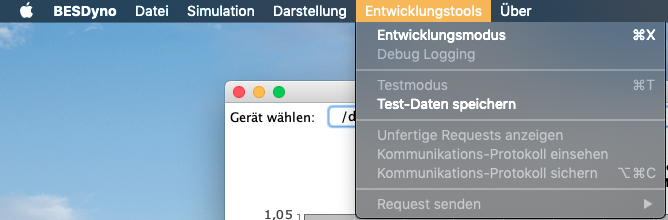


Abbildung 15: Entwicklungstools-Menü, Quelle: Screenshot

Im Menü-Punkt „Entwicklungstools“ finden sich Debugging-Tools, die sich während der Entwicklung des Programms als sehr nützlich erwiesen haben.

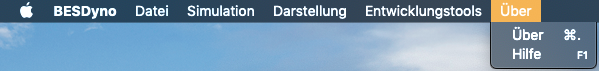


Abbildung 16: Über-Menü, Quelle: Screenshot

Im Menü-Punkt „Über“ findet sich das ‚Über das Programm‘-Dialogfenster, sowie eine Hilfe, die als Online-Handbuch realisiert wird.

## SteelSeries (Tachometer & Thermometer)

Für die Anzeige der Tachometer und Thermometer im MeasureDialog habe ich nach einer Bibliothek gesucht, die einfach zu bedienen und optisch zufriedenstellen ist.

Fündig geworden bin ich auf dem Blog von Gerrit Grunwald: <https://harmoniccode.blogspot.com>.

Er hat die SteelSeries-Library für JavaFX, Java Swing und JavaScript ins Leben gerufen.

GitHub-Repository: <https://github.com/HanSolo/SteelSeries-Swing>

Da die Dokumentation zu dieser Bibliothek auf dem Blog schwer zu finden war (es handelt sich nur um Blog-Einträge zu verschiedensten Zeitpunkten), habe ich eine Website gesucht, die diese Bibliothek verwendet und den JavaScript-Quelltext eingesehen, um zu verstehen, wie diese Bibliothek funktioniert:

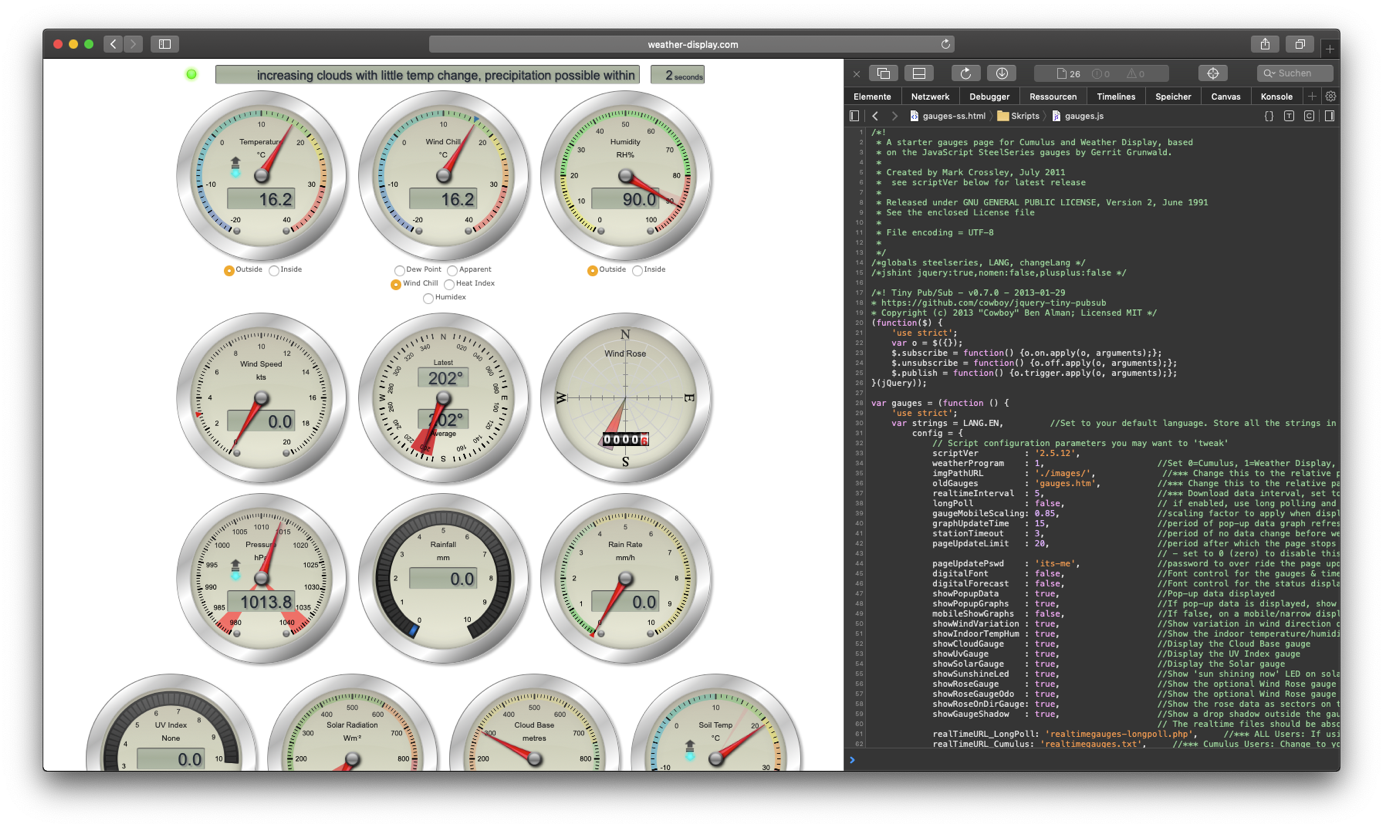


Abbildung 17: <http://www.weather-display.com/windy/gb/gauges-ss.html>, Abgerufen am 19.02.2019, Quelle: Screenshot

Herausgefunden habe ich die Aufgaben folgender Methoden:

Bsp: private final Radial gauge = new Radial();

|  |  |
| --- | --- |
| Methode | Beschreibung |
| gauge.setTitle() | Title des Gauge |
| gauge.setUnitString() | Textfeld für die physikalische Einheit |
| gauge.setFrameDesign() | Wählt aus einer enum das Design aus |
| gauge.setLedVisible() | Aktiviert/Deaktiviert „LED“ |
| gauge.setGlowVisible() | Aktiviert/Deaktiviert „glow“ um den Gauge |
| gauge.setGlowColor() | Farbe des „Glowings“ |
| gauge.setKnobStyle() | Stil des „Knopfes“ in der Mitte aus enum |
| gauge.setKnobType() | Weitere Anpassung des „Knopfes“ |
| gauge.setMaxValue() | Maximalwert der Skala |
| gauge.setMajorTickSpacing() | Abstand zwischen den großen, beschrifteten Strichen |
| gauge.setMinorTickSpacing() | Abstand zwischen den kleinen Strichen |
| gauge.setTickmarkColor() | Farbe der Striche |
| gauge.setTicklabelOrientation() | Orientierung der Beschriftung (Horizontal, Radial, etc) |
| gauge.setLabelNumberFormat() | Format der Beschriftung (Normal, Scientific, Prozent, etc) |
| gauge.setThreasholdColor() | Farbe des benutzerdefinierten Markers |
| gauge.setThreasholdType() | Design des Markers (Pfeil oder Dreieck) |
| gauge.setThreasholdVisible() | Aktiviert/Deaktiviert Marker |
| gauge.setValue() | Neuer Wert wird ausgegeben |
| gauge.setValueAnimated() | Neuer Wert wird ausgegeben und vom aktuellen Stand animiert „angefahren“ |
| gauge.setLcdValue() | Neuer Wert für die Digitalanzeige am Gauge |
| setThreashold() | Neuer Wert für den Marker |

Tabelle 3: Methoden der Bibliothek "SteelSeries"

Über diese Methoden lassen sich die anzeigen personalisieren.

Die Bibliothek verfügt über verschiedenste Klassen für die verschiedenen Anzeigen, verwendet habe ich Radial() für die Tachometer und Linear() für die Thermometer.

## JFreeChart (Leistungsdiagramm)

Um das Leistungsdiagramm zu erstellen, wurde die Bibliothek JFreeChart verwendet. Das „Free“ in JFreeChart ist allerdings nur relativ, da die Bibliothek zwar gratis verwendet werden darf, man aber keine Code-Beispiele erhält, diese erhalten nur zahlende Kunden. Aus diesem Grund habe ich das Diagramm meines Vorgängers übernommen und an mein Programm angepasst.

Die folgende Dokumentation der Methoden stammt von Levin Messing aus der Diplomarbeit xy:[[1]](#footnote-1)

|  |  |
| --- | --- |
| Methode | Beschreibung |
| ChartFactory.createXYLineChart() | Erstellt ein JFreeChart Objekt. Legende, Dataset, Orientierung und Überschrift werden gesetzt. |
| XYPlot.setDataset() | Fügt ein Dataset dem Plot eines JFreeChart-Objektes hinzu. |
| XYPlot.setRangeAxis() | Fügt eine Y-Achse dem Diagramm hinzu. |
| XYPlot.mapDatasetToRangeAxis() | Weist einem Dataset die zu verwendende Y-Achse zu. |
| XYPlot.setRenderer() | Setzt den Renderer, der für das Design zuständig ist. |
| XYPlot.addRangeMarker() | Fügt einen Marker hinzu, der bestimmte Werte markiert. |
| Dataset.addSeries() | Fügt ein Series Objekt dem Dataset Objekt hinzu. |
| ValueMarker.setStroke() | Setzt das Design eines Markers |
| ValueMarker.setLabelTextAnchor() | Setzt die Position vom Text des Markers |
| ValueMarker.setValue() | Setzt den Wert den der Marker hervorheben soll. |
| XYLineAndShapeRenderer.setSeriesStroke() | Setzt das Design der Linie. |
| JFreeChart.addSubtitle() | Fügt einen Untertitel hinzu. |

Tabelle 4: JFreeChart XYLineChartMethoden, Quelle: Levin Messing, Diplomarbeit xy

## JSON (Lesen und Schreiben der Config-File)

Um Daten im Allgemeinen und Konfigurationsdaten im Speziellen zu speichern, wird gerne das JSON-Format verwendet. Dies bietet den Vorteil, dass es auch für den Menschen lesbar ist, da es auf dem Key-Value-Prinzip basiert.

Leider ist die offizielle Bibliothek javax.json nur in der Java Enterprise Edition verfügbar und muss nachinstalliert werden. Zusätzlich zur javax.json.jar wird noch die javax.json-api-*versionsnummer*.jar benötigt.

|  |  |
| --- | --- |
| Methode | Beschreibung |
| JsonObjectBuilder.add(key, value) | Fügt dem JSON-Objekt einen Wert hinzu: Variable „key“ ist ein String und value kann jeder Datentyp sein. |
| JsonObjectBuilder.build() | Gibt ein JSON-Objekt zurück. |
| JsonReader.readObject() | Liest ein JSON-Objekt aus einem String oder einer Datei ein. |
| JsonObject.getBoolean(key) | Gibt den hinterlegten Boolean-Wert von „key“ zurück. |
| JsonObject.getInt(key) | Gibt den hinterlegten Integer-Wert von „key“ zurück. |
| JsonObject.getJsonNumber(key) | Wird für double benötigt: Gibt JSON-Number zurück. |
| JsonNumber.doubleValue() | Konvertiert JSON-Number in double |

Tabelle 5: javax.json Methoden

Als Datei geschrieben wird mittels eines BufferdWriter w: w.write(jsonObj.toString());

Eingelesen wird mit einem FileInputStream fis: Json.createReader(fis);

## Kommunikation

## Protokoll

Für die Kommunikation zwischen dem µC und der PC-Applikation wurde ein Protokoll definiert. Es sieht sich als einfaches Request/Response-System bzw. basiert auf dem Master/Slave-Prinzip. Der Master (PC) muss eine Anfrage stellen um eine Antwort vom Slave (µC) zu erhalten. Das Protokoll ist verbindungslos und die Datenübertragung erfolgt textuell.

## Aufbau

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Startzeichen | Daten | Trennzeichen | Prüfsumme | Stoppzeichen |
| „:“ | data#data#data | „>“ | CRC32 | „;“ |

Tabelle 6: Aufbau des Protokolls

Wenn die Response aus mehreren Daten besteht, wird zwischen den Daten ein Hash „#“ als Trennzeichen verwendet. Die CRC32-Prüfsumme wird über die Daten generiert, ohne Berücksichtigung der Steuerzeichen („:“, „>“, „;“), mit Ausnahme des Hash („#“), der sich als Teil der Daten sieht.

Beispiel: Request „ALL“:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Startzeichen | Daten | Trennzeichen | Prüfsumme | Stoppzeichen |
| „:“ | 0#0#0.00#0.00#800144 | „>“ | 51141376 | „;“ |

Tabelle 7: Beispiel des Protokoll-Aufbaus

Weitere Informationen zu den Daten gibt es unter dem Punkt *4.6.6.1.3 Requests & Responses*.

## Ablauf

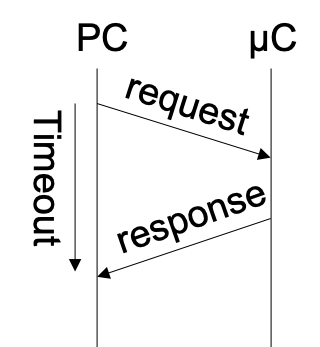


Abbildung 18: Ablauf des Protokolls, Quelle: Eigene Zeichnung

Der PC sendet eine Request aus und speichert den Zeitpunkt des Versendens. Der µC antwortet mit der gewünschten Response. Sollte die Response nicht in einer vorgegebenen Zeit eintreffen, wird bei manchen Requests die Anfrage erneut gestellt, bei anderen einfach verworfen. Die Länge des Timeouts unterscheidet sich auch von Request zu Request.

|  |  |  |
| --- | --- | --- |
| Request | Erneute Anfrage (Timeout) | Länge des Timeouts (ms) |
| INIT | ja | 5000 |
| VERSION | ja | 5000 |
| START | ja | 1000 |
| ENGINE | ja | 1000 |
| KILL | nein | 1000 |
| ALL | nein | 1000 |
| MEASURE | nein | 1000 |
| MEASURENO | nein | 1000 |
| FINE | nein | 1000 |
| WARNING | nein | 1000 |
| SEVERE | nein | 1000 |
| MAXPROBLEMS | nein | 1000 |

Abbildung 19: Timeout-Handling der Requests

Auf eine weitere Überprüfungsmaßnahme außer dem CRC – wie zB das Idle-RQ-Verfahren – wurde verzichtet, da es nicht notwendig ist. Fehlerhafte Pakete werden einfach verworfen, bzw. nicht ankommende Pakete werden ignoriert und das Programm geht zum nächsten Messpunkt.

## Requests & Responses

|  |  |  |  |
| --- | --- | --- | --- |
| Request | Request-String | Response (Data) | Aufgabe am µC |
| INIT | i | BESDyno | - |
| VERSION | p | Versionsnummer | - |
| START | s | Umgebungstemperatur und -luftdruck | Auslesen der Werte des BMP180 |
| ENGINE | e | Motor- und Abgastemperatur | Auslesen der Thermoelemente am ADC. |
| KILL | k | „KILL“ | Rücksetzen des Timers |
| ALL | a | Motor- und Walzendrehzahl,  Motor- und Abgastemperatur und Zeit | Messen der Periodendauer an D2 & D3, Auslesen von A0 & A1 (Thermoelemente) |
| MEASURE | m | Motor- und Walzendrehzahl, Zeit | Messen der Periodendauer an D2 & D3 |
| MEASURENO | n | Walzendrehzahl, Zeit | Messen der Periodendauer an D3 |
| FINE | f | „FINE“ | Setzt grüne LED |
| WARNING | w | „WARNING“ | Setzt gelbe LED |
| SEVERE | v | „SEVERE“ | Setzt rote LED |
| MAXPROBLEMS | x | „MAXPROBLEMS“ | Setzt gelbe und rote LED |
| *DEBUG* | *d* | *Für den Menschen lesbaren Text im Terminal aller Messwerte* | *Alles wird gemessen* |

Tabelle 8: Arten der Requests & Responses

|  |  |
| --- | --- |
| Request | Response-Beispiel (mit gleichen Werten) |
| INIT | :BESDyno>CRC; |
| VERSION | :1.00>CRC; |
| START | :22.88#9898.00#420.00>CRC; |
| ENGINE | :55.32#439.21>CRC; |
| KILL | :KILL>CRC; |
| ALL | :5350#1220#55.32#439.21#800144>CRC; |
| MEASURE | :5350#1220#800144>CRC; |
| MEASURENO | :1220#800144>CRC; |
| FINE | :FINE>CRC; |
| WARNING | :WARNING>CRC; |
| SEVERE | :SEVERE>CRC; |
| MAXPROBLEMS | :MAXPROBLRMS>CRC; |
| DEBUG | Temperatur: 22.88  Luftdruck: 9898.00  Höhenmeter: 420.00  A0 Motor: 55.32  A1 Abgas: 439.21  D2 Motor: 5350  D3 Walze: 1220  Zeit (µs): 800144 |

Tabelle 9: Beispiele für Responses

## Prüfsumme

## Behandlung von Fehlern

## Datenerfassung und -sicherung

## Serielle Schnittstelle

## JSSC

## RxTxComm

## Messvorgang

## Messkette

## Berechnungen

## Live-Anzeigen

## Numerische Simulation

## Tabellarische Berechnung

## Kompensation der Verlustleistungen

## Kompensation der Leistung nach DIN70020

## Multithreading

## SwingWorker allgemein

## RxTxWorker

## MeasurementWorker

## Maßnahmen zur einwandfreien Plattformunabhängigkeit

## Bibliotheken für die serielle Kommunikation

## OS native Look & Feel

## Dateinamen

## Tastaturkürzel

## Nennenswerte Features der Software

## Einheiten

## Zwei Messmethoden

## Exportieren und Laden von Config-Files

## Analoge Anzeigen der Bibliothek SteelSeries

## Live-Anzeige der Motor- und Abgastemperatur

## Dark-Mode

## Entwicklungs- und Testmodus

## Kommunikationsprotokoll

## CSV-Dateien

## Aufgetretene Probleme und Verbesserungsmöglichkeiten

## Singleton

## Kommunikation (Timeout)

## Messvorgang

## Berechnung

## JFreeChart

## SteelSeries

## µC-Programmierung

## Mikrocontroller

## Peripherie

## UART

## ADC

## I2C

## Messungen

## Thermoelemente: Motor- & Abgastemperaturen

## BMP180: Umgebungstemperatur & -luftdruck

## Drehzahlen

## Variante 1: Zählen der Impulse

## Variante 2: Messen der Periodendauer

## CRC-Berechnung

## Status-LEDs, optische Rückmeldung an den User

## Wichtigste Funktionen der Arduino.h

## micros();

## attachInterrupt();

## pulseInLong();

## pinMode();

## analogRead();

## Aufgetretene Probleme und Verbesserungsmöglichkeiten

## Anhang

1. Quelle: Levin Messing, Diplomarbeit xy [↑](#footnote-ref-1)