Grafische Benutzeroberfläche – Florian Boemmel

1. Generelles

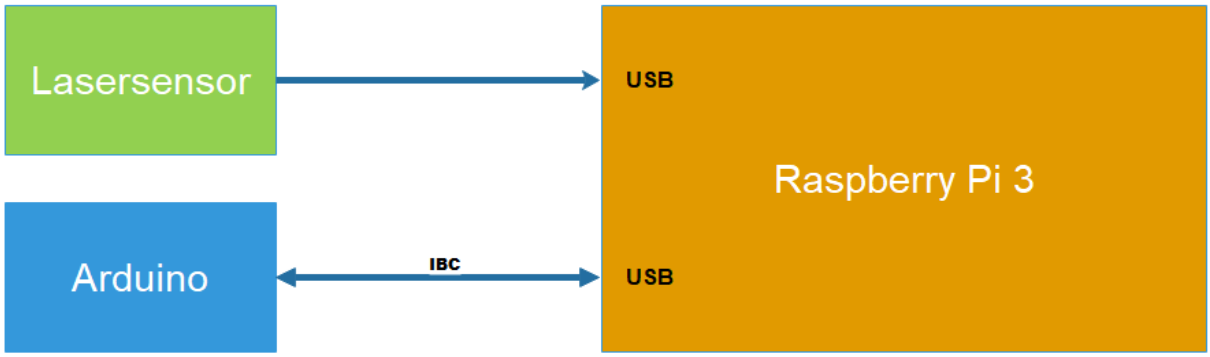
Die grafische Benutzeroberfläche (im folgenden GUI bezeichnet) stellt, abstrakt dargestellt, dass Bindungsglied zwischen Benutzer und dem Fahrzeug da. Über diese, soll die Steuerung des Fahrzeugs erfolgen.

Die GUI soll unter den Aspekten der Skalierbarkeit und der einfachen Erweiterung durch andere Projektmitglieder entwickelt werden. Aus diesem Grund, einigte sich das Projektteam darauf, dass alle Module auf dem Raspberry Pi 3 Model B (im folgenden Pi bezeichnet) in C++ entwickelt werden.

Module stellen hierbei externe Klassen da. Diese werden unabhängig von der GUI entwickelt und müssen anschließend in die GUI eingebunden werden.

Folgende Module existieren:

- Lasersensor
- IBC



2. Entwicklungsumgebung

Während der Anfangsphase des Projekts stand die Zielplattform der GUI noch nicht eindeutig fest. Konkret bedeutete dies, dass das Team zwischen einer Desktop-Anwendung und einer Touch-Anwendung direkt auf dem Pi oder einer App schwankte. Jedoch ist gerade die Zielplattform ein ausschlaggebender Faktor, um das passende GUI-Toolkit auszuwählen.

Auf der Basis der unbekannten Zielplattform, sei es eine Desktop-Anwendung unter Linux / Windows / OSX oder eine Mobile-Anwendung, recherchierte ich über mögliche GUI-Toolkits. Das C++ basierende GUI-Toolkit Qt stach dabei vermehrt heraus. Demnach ist es möglich, auf der Basis eines Projekts alle Zielplattformen zu bedienen.

Qt bietet zusätzlich die Möglichkeit in gewissen Umfang plattformunabhängig zu entwickeln. Konkret bedeutet dies, dass Layout und plattformunabhängige Logik auf jedem Betriebssystem entwickelt und getestet werden kann. Lediglich betriebssystemspezifische Logik kann nur auf dem dazugehörigen Rechner getestet werden. Eine Kompilierung ist jedoch per Cross-Kompilierung möglich.

Weiterhin basiert Qt auf C++. Somit können auf C++ basierende Module ohne weitere Probleme in das Projekt eingefügt werden und erfüllt somit die Voraussetzung des Teams, alle Module auf dem Pi in C++ zu entwickeln.

Ein weiterer Vorteil in Qt liegt in der enorm großen Community und dem einfachen Zugang zu sehr detaillierten Dokumentationen und Beispielen. Qt stellt außerdem auf den gängigsten Plattformen Ihre eigene IDE(QtCreator) mit einem Designer zur Verfügung.

Das Team einigte sich schlussendlich auf eine Touch-Anwendung direkt auf dem Pi. Dazu wurde ein Touchdisplay der Größe 3.2 Zoll direkt auf dem Pi angebracht. Der Pi bietet eine große Auswahl an Möglichkeiten, jedoch ist seine Rechenleistung begrenzt und für einige Tätigkeiten, wie z.B. eine umfangreiche GUI direkt auf ihn zu programmieren eher ungeeignet.

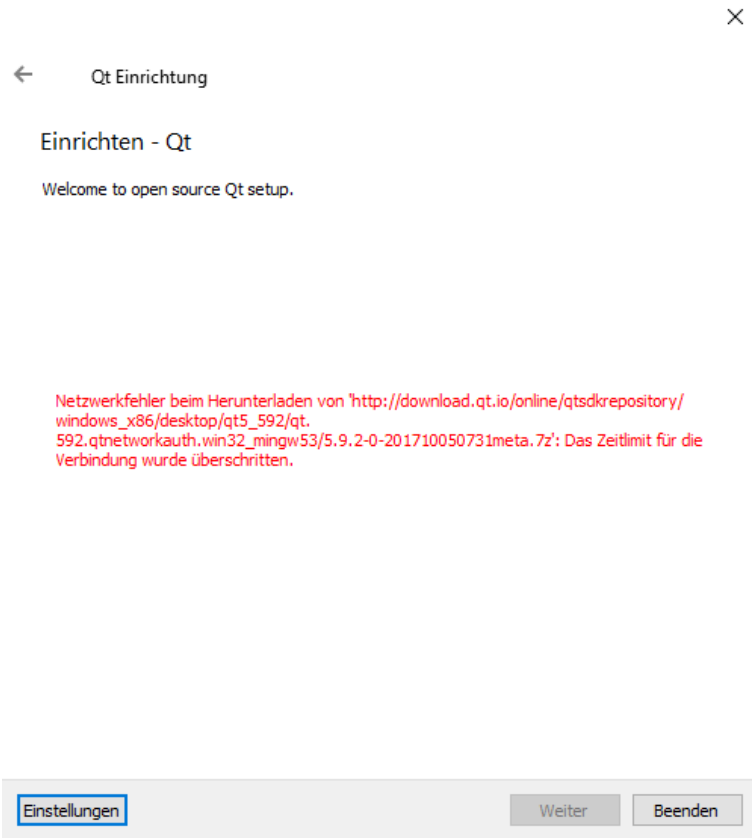
Aus den bereits aufgeführten Gründen wird die GUI in der Sprache C++ mit dem GUI-Toolkit Qt5 realisiert. Dabei möchte ich noch kurz anfügen, dass ich bis dato noch nichts mit Qt gemacht habe und generell noch keine GUI geschrieben habe.

3. Installation & Einrichtung von QtCreator

Qt ist in zwei Versionen verfügbar. Zu einem Open Source und Commercial. Die Unterschiede können unter der Download-Seite eingesehen werden. Ich verwende die kostenlose Open Source Version.

I. Installation unter Windows

Ich möchte hier ausführlicher auf die Installation unter Windows eingehen, da es ein entscheidendes Problem dabei gab. Dies ist auch unter OSX zu beobachten. Downloadet man die Installationdatei über die Downloadseite und führt diese aus, kann es passieren, dass folgende Fehlermeldung während der Installation auftritt:



Auch eine erneute Ausführung der Installation führte zum gleichen Ergebnis. Die einzige Lösung hierfür war es, anstatt der Online-Installation, eine Offline-Installation durchzuführen. Für die Offline-Installation muss zunächst unter der schwer zu findenden Offline-Downloadseite die gewünschte Version gewählt werden und anschließend die Plattform. Danach sollte die Installation reibungslos funktionieren.

Ein letzter wichtiger Punkt ist das Auswählen der zu installierenden Pakete. Unter Windows reicht die von Qt standartmäßig ausgewählten Pakete. Jedoch sollte unter dem Punkt Tools MinGW 5.* ausgewählt werden, falls dieser noch

nicht zuvor installiert wurde. Dieser stellt den Standard Compiler unter Windows dar.

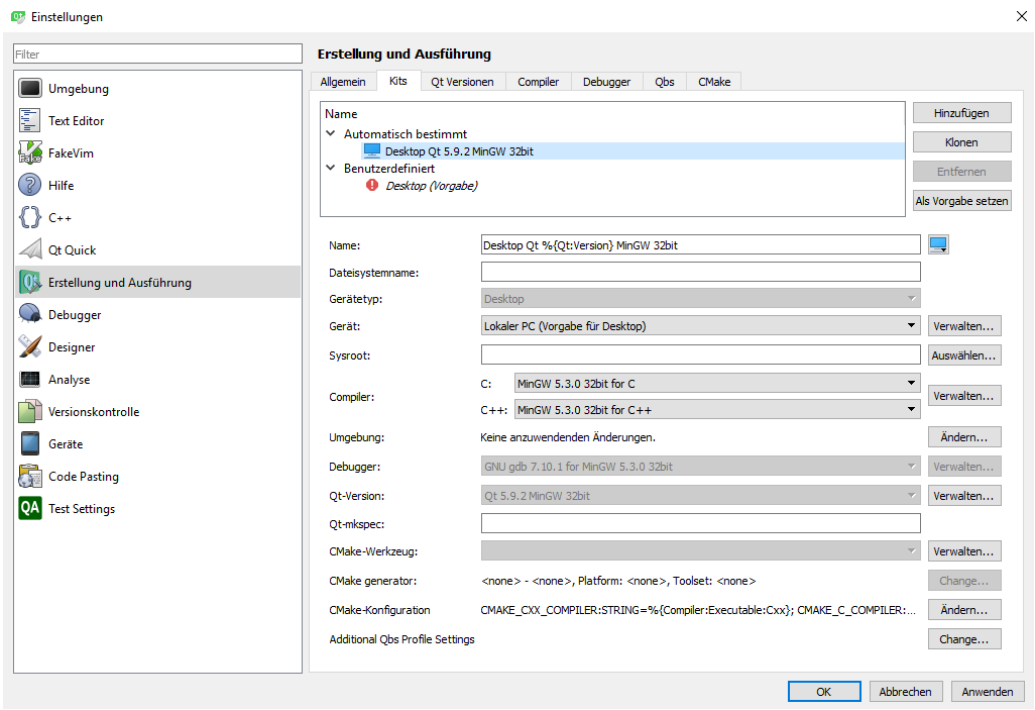
II. Installation unter Linux (Raspberry Pi)

Unter Linux gestaltet sich die Installation etwas einfacher. Dazu müssen lediglich folgenden Kommandos im Terminal ausgeführt werden:

```
sudo apt-get update
sudo apt-get dist-upgrade
sudo apt-get install qt5-default

sudo apt-get install qtcreator

sudo apt-get install libqt5serialport5
sudo apt-get install libqt5serialport5-dev
```

III. Einrichten

Einige erstmalige Einstellungen sind für einen korrekten Kompiliervorgang nötig. Dazu muss ein Kit (Bezeichnung von Qt) eingerichtet werden, falls dies nicht automatisch geschieht. Auch im Falle eines Fehlers beim Kompiliervorgang kann es am nicht korrekt eingestellten Kit liegen.



Dazu muss unter dem Punkt Compiler ein C- sowie C++-Compiler eingestellt sein. Ist das Kit, in diesem Fall „Desktop“ unter dem Reiter Automatisch bestimmt, nicht rot oder gelb markiert, ist das Kit erfolgreich eingestellt

worden und eine Kompilierung ist nun möglich. Die Einrichtung unter Linux ist äquivalent. Lediglich die Compiler sind verschieden.

IV. Cross-Kompilierung vs. „Copy And Paste“

Um eine GUI auf einer spezifischen Zielplattform ausführen zu können, muss diese mit einem für die Zielplattform geeigneten Compiler übersetzt werden. Qt bietet die Möglichkeit einer Cross-Kompilierung an.

Jedoch gestaltete sich die Einrichtung eines Cross-Compilers als schwierig. Bereits bei der Beschaffung der richtigen Source-Dateien, passend, für den Pi stellte sich heraus, dass dies ohne externe Tools nicht möglich war. Weiterhin konnte die Dokumentation von Qt mir auch nicht entscheidend weiterhelfen.

Ich probierte aus diesem Grund ein einfaches „Copy and Paste“ aus. Genauer beschrieben, entwickelte ich ein minimales Beispiel auf meinen Windows Computer und pushte dieses anschließend auf GIT und pullte dieses auf den Pi. Unerwartet konnte das Projekt ohne Probleme auf dem Pi geöffnet werden und übersetzt werden.

Ich entschied mich von nun an, die GUI auf einem stärkeren Rechner zu entwickeln und anschließende Tests direkt auf dem Pi durchzuführen. Dies erwies sich im Verlauf des Projekts als vorteilhaft. Einige Einschränkungen gab es jedoch trotzdem.

Plattformspezifische Funktionalitäten mussten entweder auskommentiert oder per Compiler-Schalter deaktiviert werden. Auch nach der Integrierung des Protokolls von Robert war das Problem der plattformspezifischen Funktionalitäten stets gegenwärtig, da in diesem Linux Systemaufrufe getätigt werden. Daraufhin setzte ich ein virtuelles Ubuntu auf, um weiterhin, ohne weitere Compiler Schalter, kompilieren zu können und somit die Entwicklung etwas angenehmer zu gestalten.

4. Anforderungen

/G0101/ **Automatischer Start der Benutzeroberfläche:** Verbindet der Benutzer das Fahrzeug mit einer von Ihm gewählten Stromquelle, bootet der Raspberry Pi direkt in die Benutzeroberfläche des Fahrzeugs und verhindert so eine falsche Bedienmöglichkeit des Fahrzeugs.

/G0102/ **Initialisierung des Fahrzeugs:** Der Benutzer kann über einen Button das Fahrzeug initialisieren. Das bedeutet im konkreten Fall, dass zunächst ein Serieller Port geöffnet wird und das Inter Board Protocoll (IBC) gestartet wird. Weiterführende Steuerungsmöglichkeiten dürfen dem Benutzer zu diesem Zeitpunkt nicht zu Verfügung stehen.

/G0103/ **Moduswahl:** Der Benutzer hat die Möglichkeit zwischen zwei Betriebsmodi auszuwählen:

- Uhrsteuerung
- Controllersteuerung

Zusätzlich muss der Benutzer, ohne einen Modus auszuwählen, die Möglichkeit erhalten, sich die aktuellen Sensorwerte ansehen zu können.

/G0104/ **Neustart der Benutzeroberfläche:** Der Benutzer muss über ein Menü die Möglichkeit erhalten, die Benutzeroberfläche neu zu starten. Dies ist insbesondere bei Verbindungsproblemen zum Mikrocontroller unabdingbar.

/G0105/ **Beenden des Systems:** Der Benutzer muss über ein Menü die Möglichkeit erhalten, die Benutzeroberfläche sowie den Raspberry Pi ordnungsgemäß herunterfahren zu können.

/G0106/ **Uhrsteuerung:** Wählt der Benutzer den Modus Uhrsteuerung, muss diesem zunächst eine kurze Anleitung dargestellt werden, wie er die Uhren anzulegen hat. Hat der Benutzer diese Information verstanden, muss er diese bestätigen. Nach der positiven Bestätigung, muss dem Benutzer die Steuerung anhand von Bildern und Animationen verständlich erklärt werden. Zudem muss der Benutzer über einen Button die Möglichkeit gegeben werden, den Raumscan zu starten (/G0111/).

/G0107/ **Controllersteuerung**: Wählt der Benutzer den Modus Controllersteuerung, wird dieser aufgefordert, den Controller griffbereit zu halten. Hat der Benutzer diese Information verstanden, muss er diese bestätigen. Nach der positiven Bestätigung, muss dem Benutzer die Steuerung anhand von Bildern und Animationen verständlich erklärt werden. Zudem muss der Benutzer über einen Button die Möglichkeit gegeben werden, den Raumscan zu starten (/G0111/).

/G0108/ **Navigation:** Der Benutzer muss jederzeit die Möglichkeit erhalten, zur Moduswahl (/G0103/) zurückzukehren und einen anderen Modus wählen zu können. Dabei darf die Navigationstiefe in einem Modus keine Rolle spielen.

/G0109/ **Darstellung der Sensorwerte:** Dem Benutzer muss nach der Wahl, sich die Sensorwerte anzeigen zu lassen, eine Übersicht der vorhandenen Sensoren und deren aktuellen Werten dargestellt werden.

/G0110/ **Fehleranzeige:** Dem Benutzer muss eine Fehleranzeige bereitgestellt werden. Diese muss unabhängig von allen Darstellungen und Benutzereingaben jederzeit gut sichtbar sein. Weiterhin müssen dem Benutzer spezifische Details über einem Fehlerfall dargestellt werden.

/G0111/ **Raumscan**: Der Benutzer muss die Möglichkeit erhalten, nach der Wahl eines Modi, den Raumscan zu initialisieren. Während der Raumscan aktiv ist, müssen dem Benutzer die Sensordaten dargestellt werden (/G0109).

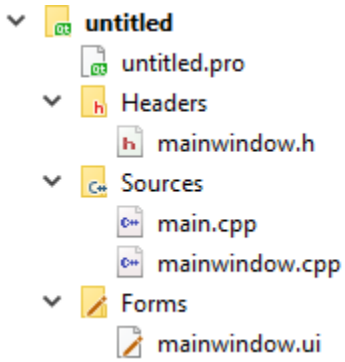
5. Umsetzung

I. Projekterstellung

Die im Rahmen des Projekts entwickelte GUI, ist eine Qt-Widget-Anwendung. Diese kann wie folgt generiert werden:

1. Neues Projekt
2. Anwendungen → Qt-Widget-Anwendung
3. Namen für das Projekt vergeben und einen Pfad auswählen
4. Das passende Kit auswählen (Falls keins vorhanden ist, muss eins, wie unter Punkt 3 (Einrichten), erzeugt werden)
5. Nun kann der Klassenname für das Hauptfenster sowie die Basisklasse eingestellt werden. Dabei wählt man bei der Basisklasse QMainWindow und einen gewünschten Namen für die Klasse und deren Quelldateien.
6. Schließlich kann noch eine Versionsverwaltung eingestellt werden.

Nun ist das Projekt angelegt und enthält folgende Dateien:



- .pro: Durch ein qmake wird aus diesem ein Makefile
- mainwindow.h: Die Headerdatei des Hauptfensters
- mainwindow.cpp: Die Sourcedatei des Hauptfensters
- mainwindow.ui: Die Formulardatei des Hauptfensters
- main.cpp: Die Main der GUI-Anwendung

Im Folgenden wird auf die main.cpp genauer eingegangen, um das Grundprinzip von Qt besser zu verstehen:

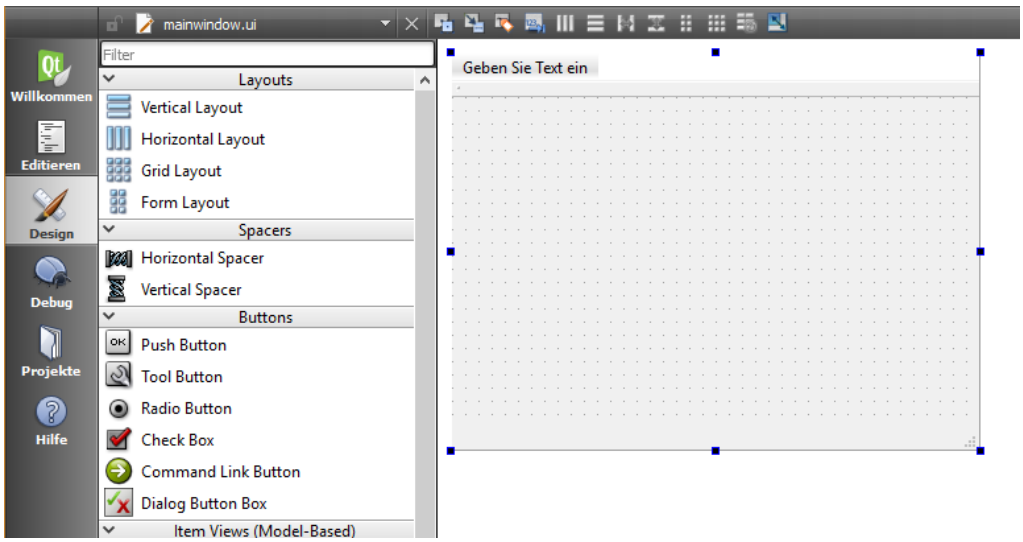
```cpp
#include "mainwindow.h"
#include <QApplication>

int main(int argc, char *argv[])
{
    QApplication a(argc, argv);
    MainWindow w;
    w.show();

    return a.exec();
}
```

Der Ablauf einer Qt-Widgets-Anwendung ist denkbar simpel. Zunächst wird eine Instanz von QApplication erzeugt.

Anschließend wird eine Instanz des Hauptfensters erstellt und mit der Methode show() wird dieses anschließend angezeigt. Würde man show() nicht aufrufen, würde die Anwendung dennoch ausgeführt werden, nur ohne Fenster.

Das letzte Kommando übergibt die Kontrolle des Programms. Die Kontrolle bedeutet hier, dass der Aufruf a.exec() Qt anweist, auf Events zu hören. Ohne diese Anweisung, wären keine Benutzerinteraktionen möglich. a.exec() wird erst beendet, wenn die Qt-Anwendung beendet wird. Somit wird das return erst nach der Beendigung der Qt-Anwendung ausgeführt.

Weiterhin ist eine genauere Betrachtung der Formulardatei des Hauptfensters ratsam:



Öffnet man diese, wird automatisch der integrierte QT-Designer (im Folgenden als Designer bezeichnet) aufgerufen. Dort kann das Hauptfenster geändert werden und Kontroll- sowie Layoutelemente hinzugefügt werden. Formulardateien sind XML-Basiert und können im Qt-Creator nur über den Designer geändert werden.

II. Designer vs. Code

Zu Begin jedes Projekts, stellt sich die Frage, sollte man den Designer verwenden oder nicht? Dabei gehen die Meinungen sehr weit auseinander.

Gerade bei großen Projekten wird der Designer oft empfohlen und bei kleineren die codebasierende Lösung. Jedoch gibt es auch dort andere Meinungen.

Auch ich überleget lange, welche Herangehensweise ich nutzen möchte für unser Projekt. Es folgten einige Versuche der beiden Möglichkeiten. Ich entschied mich schlussendlich für eine reine Implementierung des Designs und damit gegen die Verwendung des Designers.

III. Layout der GUI

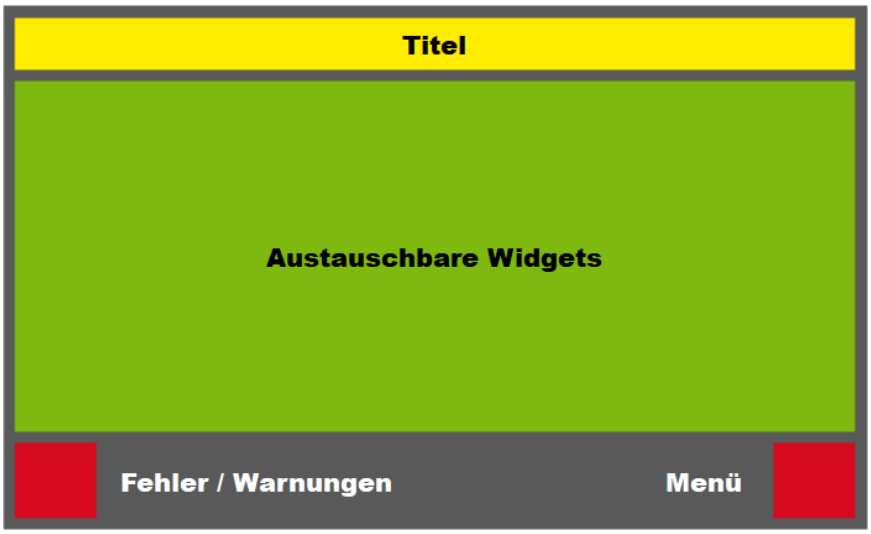
Zunächst musste ich mir ein Grundkonzept des Layouts der GUI überlegen. Wir verwendeten ein 3.2 Zoll großes Touchdisplay (im Folgenden als Display bezeichnet) direkt auf dem Pi. Demnach schränkte bereits das Display das Konzept drastisch ein:

- Die erkannten Eingaben sind ungenau und werden nicht immer gleich erkannt
- Die Größe des Displays ist für die geforderten Anforderungen nicht optimal

Aus dem ersten Punkt folgte die Einschränkung, dass Kontrollelemente eine ausreichende Größe besitzen und gegen eine doppelte Betätigung abgesichert werden müssen.

Aus dem zweiten Punkt folgte die Einschränkung, dass Informationen kompakt dargestellt werden müssen. Dies gilt auch für Kontrollelemente.

Der daraus entstandene Entwurf des Layouts:



Dabei stellt der Titel ein einfacher Schriftzug mit dem Text „StarCar“ dar.

Durch das Konzept der Austauschbaren Widgets wird die Einschränkung der kompakten Informationsdarstellung erfüllt. Im Detail bedeutet dies, das die dargestellten Elemente ein sogenanntes Masterlayout bilden, dass sich nicht ändert und nur die wichtigsten Elemente jederzeit erreichbar sind. Wie der Zugang zu der Fehleranzeige sowie dem Menü.

Weiterhin wurden den wichtigsten Kontrollelementen eine ausreichende Größe zugewiesen und somit wird der Einschränkung der ausreichenden Größe erfüllt. Alle weiteren Kontrollelemente müssen im späteren Entwicklungsprozess an die verfügbare Größe der Austauschbaren Widgets angepasst werden.

IV. Masterlayout

Zu Beginn wurde das Masterlayout entwickelt. Dabei wurde zunächst der Fokus auf die Implementierung des zuvor entworfenen Masterlayouts gelegt. Dabei erstellte ich drei Methoden:

- generateLayout()
- generateStyle()
- setupConnects()

In der ersten Methode wird das Layout des Fensters folgendermaßen generiert:

```
void HomeWindow::generateLayout(){

    centralVBox         = new QVBoxLayout(ui->centralWidget);
    hBox1               = new QHBoxLayout();
    lblHeadline         = new QLabel();
    pButtonAlert        = new QPushButton(QIcon();
    pButtonExit         = new QPushButton(QIcon();

    mainStackedWidget   = new QStackedWidget();

    centralVBox->addSpacing(8);
    centralVBox->addWidget(lblHeadline,0,Qt::AlignHCenter);
    centralVBox->addSpacing(12);
    centralVBox->addWidget(mainStackedWidget);
    centralVBox->addLayout(hBox1);

    hBox1->addSpacing(5);
    hBox1->addWidget(pButtonAlert);
    hBox1->addSpacing(200);
    hBox1->addWidget(pButtonExit);
    hBox1->addSpacing(5);

    centralVBox->addSpacing(5);
}
```

Qt bietet für die Strukturierung von Elementen „Layouts“ an. Ich verwendete QVBoxLayouts und QHBoxLayouts. Dabei stellen diese zu einem eine Box dar, in die Elemente vertikal und zum anderen horizontal angeordnet werden.

In die vertikale Box wird zunächst der Titel in Form eines Labels eingefügt und anschließend ein QStackedWidget. Ein QStackedWidget ist ein Container und seine Hauptfunktionalität besteht darin, dass Widgets in diesem gestapelt werden können. Schließlich wird eine horizontale Box eingefügt. Diese beinhaltet einen Button für die Fehleranzeige und einen für das Menü.

Zusammengefasst bilden diese Elemente das Masterlayout. Alle späteren Widgets werden nur im QStackedWidget angezeigt.

In der zweiten Methode wird das erzeugte Layout und die hinzugefügten Kontrollelemente gestylt:

```
void HomeWindow::generateStyle(){

/*******************************Margins*****************************************/

    ui->centralWidget->setContentsMargins(0,0,0,0);
    centralVBox->setContentsMargins(0,0,0,0);

/*******************************StyleSheets*************************************/

    this->setStyleSheet("QWidget{"
                        "background-color: #2b2b2b;}");

/*****************************Windowstyle***************************************/

    this->setFixedSize(320,240);
    setWindowFlags(Qt::FramelessWindowHint);

/*****************************Button & Label************************************/

    pButtonExit->setIconSize(QSize(32,32));
    pButtonExit->resize(32,32);

    pButtonAlert->setIconSize(QSize(32,32));
    pButtonAlert->resize(32,32);

    lblHeadline->setText("StarCar");

    lblHeadline->setStyleSheet("QLabel{"
                               "color: yellow;"
                               "font-family: TimesNewRoman;"
                               "font-style: normal;"
                               "font-size: 15pt;"
                               "font-weight: bold;}");

    pButtonExit->setStyleSheet("QPushButton{"
                               "border-radius:  10px;"
                               "border-width:   3px;"
                               "border-color:   black"
                               "border-style:    solid}");

    pButtonAlert->setStyleSheet("QPushButton{"
                                "border-radius:  10px;"
                                "border-width:   5px;"
                                "border-color:   black"
                                "border-style:    solid}");
}
```

Im ersten Schritt werden die Abstände des Masterlayouts gesetzt. Nach den beiden ersten Kommandos werden die Ränder entfernt und somit der gesamte Platz ausgenutzt. Standartmäßig werden 8 Pixel Rand automatisch gesetzt.

Ein ganz wichtiger Punkt ist es, wenn man nicht mit dem Designer arbeitet, dass man die Fenstergröße explizit festlegt. In meinem Fall gleich der Größe des Displays auf dem Pi. Weiterhin muss der Rand des Fensters entfernt werden, da es später im Vollbildmodus laufen soll und der Rand nicht benötigt wird.

Arbeitet man mit oder ohne den Designer, muss man Änderungen am Aussehen von Elementen explizit über das StyleSheet tätigen. Diese sind sehr ähnlich der CSS Syntax. Beispiele hierfür findet man unter der StyleSheet-Dokumentation von Qt.

In der dritten Methode werden die unter Qt als Signal & Slots bezeichneten Verbindungen eingerichtet:

```
void HomeWindow::setupConnects(){

    connect(pButtonExit, SIGNAL(clicked(bool)), this, SLOT(slotShowExitWidget()));
    connect(pButtonAlert, SIGNAL(clicked(bool)), this , SLOT(slotShowAlertWidget()));
}
```

Signals & Slots sind eines der Schlüsselfunktionen von Qt. Diese sind ein Mechanismus von Qt, wie sich verschiedene GUI-Elemente oder Aktionen unterhalten können. Jemand sendet ein Signal aus und ein anderer empfängt dieses. Ein Signal kann z.B. beim Drücken eines Buttons ausgesendet werden. Ein oder mehrere Empfänger, die so genannten Slots, empfangen das Signal und rufen daraufhin eine entsprechende Funktion auf.

Konkret auf dieses Projekt angewandt, wird zunächst der Menübutton mit dem Event „wurde geklickt“ mit der Klasse HomeWindow verknüpft. Bei einem Klickevent wird ab nun der Slot slotShowExitWidget() ausgeführt.

```
void HomeWindow::slotShowExitWidget(){

    exitWidget = new ExitWidget(this);

    connect(exitWidget, SIGNAL(removeWindowformStack()), this, SLOT(removeActiveWidget()));
    addWidgetToMainStackWidget(exitWidget);
}
```

In diesem Slot wird zunächst eine neue Instanz vom Typ ExitWidget erstellt.

Anschließend wird eine weitere Verbindung erstellt. Diese vereinfacht gesagt, löst bei einem Signal removeWindowfromStack den Slot removeActiveWidget aus und entfernt das exitWidget aus dem QStackWidget. Hier sieht man den enormen Vorteil von Signals & Slots. Ein einfaches Signal aus dem neuen Widget informiert das Masterlayout dieses wieder zu entfernen.

```
emit removeWindowformStack();
```

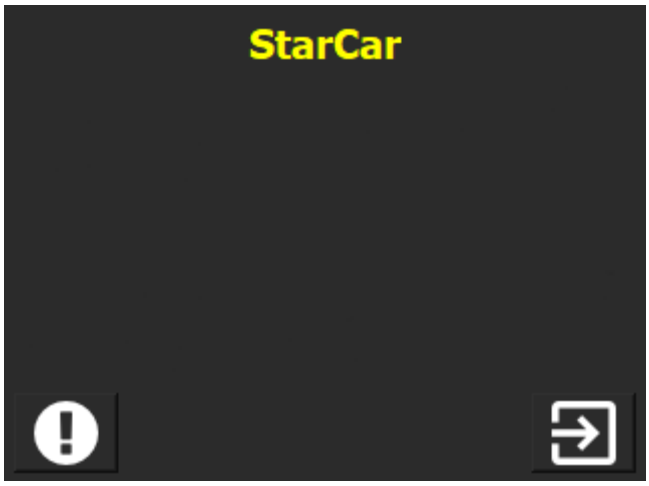
Dazu wird das Schlüsselwort emit verwendet. Auch hier sehr gut erkennbar, wie einfach ein Signal geschickt werden kann aus dem neuen Widget.

Schließlich wird das zuvor erzeugte ExitWidget in das QStackWidget eingefügt und angezeigt.

Äquivalent dazu ist die Vorgehensweise bei dem Button für die Fehleranzeige.

Wie sich im späteren Verlauf der Entwicklung zeigte, erwies sich das Konzept erst das Layout zu genieren, dies zu stylen und schließlich die Verbindungen einzurichten als robust und wird von allen Widgets verwendet.

Das Masterlayout nach der Implementierung:



V. StartWidget

Wird die GUI gestartet wird zunächst wie eben beschreiben das Masterlayout generiert. Anschließend wird automatisch das erste Widget in das Feld der Austauschbaren Widgets geladen.