**Dokumentation**

**Projekt MiniQ 2WD**

*Semesterprojekt Embedded Systems*

Fachhochschule Nordwestschweiz

Studiengang Systemtechnik

Klasse 7Sbb

Dozent: Daniel Binggeli

Projektteam: Alex Gessler

Bruno Hürzeler

Brugg-Windisch, Freitag, 12. Januar 2018, V 1.0000.00

# Inhalt

[Inhalt 2](#_Toc503556737)

[Änderungen 3](#_Toc503556738)

[Abbildungen 3](#_Toc503556739)

[Tabellen 3](#_Toc503556740)

[Formeln 3](#_Toc503556741)

[1 Aufgabenstellung 4](#_Toc503556742)

[1.1 Rahmenbedingungen 4](#_Toc503556743)

[1.2 Projektidee 4](#_Toc503556744)

[1.3 Vorgehen 5](#_Toc503556745)

[1.4 Funktionsbeschreibung 6](#_Toc503556746)

[2 Entwurf 7](#_Toc503556747)

[2.1 Modularisierung 7](#_Toc503556748)

[2.2 Modulhierarchie 9](#_Toc503556749)

[2.3 Schnittstellendefinition 10](#_Toc503556750)

[3 Testspezifikation 17](#_Toc503556751)

[3.1 Testspezifikationen der Module in «functions.cpp» 17](#_Toc503556752)

[3.2 Globale Testspezifikationen des Gesamtsystems 18](#_Toc503556753)

[3.3 Testspezifikationen Systemanforderungen 20](#_Toc503556754)

[4 Abnahmetests 21](#_Toc503556755)

[4.1 Abnahmetest der Module in «functions.cpp» 21](#_Toc503556756)

[4.2 Abnahmetest der Systemanforderungen 27](#_Toc503556757)

[4.3 Abnahmetest des Gesamtsystems 31](#_Toc503556758)

[4.4 Geplante Optimierungen 33](#_Toc503556759)

[5 Ehrlichkeitserklärung 33](#_Toc503556760)

[6 Anhang 34](#_Toc503556761)

[6.1 Messdaten 34](#_Toc503556762)

[6.2 Code für Messungen 35](#_Toc503556763)

[6.3 Code 36](#_Toc503556764)

## Änderungen

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Index** | **Kapitel Verweis** | **Dok. Rev.** | **Datum / Wer** | **Zweck** |
| 1 | Alle | V00.0000.00 | 10.11.17 / Hub | Erstellung des Dokumentes |
| 2 | 1.4 | V00.0001.01 | 14.11.17 / Gea | Funktionsbeschreibung angepasst |
| 3 | 2 | V00.0001.02 | 21.11.17 / Hub | Softwarekonzept erstellt |
| 4 | 3 | V00.0010.01 | 04.12.18 / Gea | Testspezifikationen erstellt |
| 5 | 4 | V00.0010.02 | 01.02.18 / Gea Hue | Abnahmetests durchgeführt und eingefügt |
| 6 | 5+6 | V00.0100.01 | 12.01.18 / Gea Hue | Anhang eingefügt |
| 7 | Alle | V1.0000.00 | 12.01.18 / Gea Hue | Fertigstellung des Dokumentes |

Tabelle : Änderungen

## Abbildungen

[Abbildung 1: Funktion MiniQ als Kompass 4](#_Toc503556765)

[Abbildung 2: V-Diagramm 5](file:///C:\Users\bruno\Dropbox\ebs\00_Projekt%20MiniQ2WD\MiniQCompasMachine\Dokumente\01_Final\03_Gessler_Hürzeler_komplett.docx#_Toc503556766)

[Abbildung 3: MiniQ mit Positionsnummern Abbildung 4: Winkelwerte und LED Farben 6](#_Toc503556767)

[Abbildung 5: Benutzergesteuerte Bedienung 8](#_Toc503556768)

[Abbildung 6: Modulhierarchie 9](#_Toc503556769)

[Abbildung 7: Strukturzugriff im initUp 12](#_Toc503556770)

[Abbildung 8: Statusanzeige der RGB-LED 13](#_Toc503556771)

[Abbildung 9: Display-Treiber 15](#_Toc503556772)

[Abbildung 10: Zeitmessung Anforderung 9 27](#_Toc503556773)

[Abbildung 11: Messung Zykluszeit 29](#_Toc503556774)

## Tabellen

[Tabelle 1: Änderungen 3](#_Toc503556775)

[Tabelle 2: Testspezifikationen der Methoden in «functions.cpp» 17](#_Toc503556776)

[Tabelle 3: Globale Testspezifikationen des Gesamtsystems 19](#_Toc503556777)

[Tabelle 4: Testspezifikationen Systemanforderungen 20](#_Toc503556778)

## Formeln

[Formel 1: Geschwindigkeitsberechnung 12](#_Toc503556779)

[Formel 2: Spannung der Fotodioden 14](#_Toc503556780)

[Formel 3: Refresh-Rate Display 30](#_Toc503556781)

# Aufgabenstellung

## Rahmenbedingungen

Im Modul Embedded Systems werden in Form einer Projektarbeit die verschiedenen Schritte eines Entwicklungsprojekts durchgearbeitet. Der Auftrag ist das Programmieren eines MiniQ 2WD (Arduino Leonardo Plattform) nach einer selbstgewählten, in diesem Dokument definierten Aufgabenstellung. Die Hardware und Sensorik stehen bereits durch den MiniQ 2WD zur Verfügung. In diesem Projekt soll das Engineering von Embedded System Projekten simuliert und die praktischen Kenntnisse vertieft werden.

Detaillierte Informationen zur Aufgabenstellung und dessen Rahmenbedingungen sind im Dokument «PA\_Aufgabenstellung\_mit\_Notizen.pdf» festgehalten.

## Projektidee

Der MiniQ-Roboter soll als Kompass verwendet werden können. Gemäss Aufgabenstellung werden zwei Aufgaben gefordert weshalb der MiniQ erstens in einem manuellen Modus und zweitens in einem automatischen Modus betrieben werden soll.

Im manuellen Modus soll der MiniQ von Hand gedreht bis der gewünschte Winkel, gemessen mit dem 3-Achsen Kompass HMC5883L, auf dem LCD Display angezeigt wird. Als zweiten Sensor wird zusätzlich die aktuell gemessene Lichtstärke in [V] auf dem LCD angezeigt. Die integrierte LED wird als Aktor definiert und soll grün leuchten, wenn der MiniQ mit dem LCD-Display gegen Norden zeigt.

Im automatischen Modus soll sich der MiniQ selbständig mit seinen beiden Motoren als Aktoren nach Norden ausrichten. Im automatischen Modus soll ebenfalls der aktuell gemessene Winkel auf dem LCD-Display angezeigt werden und die LED grün leuchten, wenn der MiniQ nach Norden ausgerichtet ist.

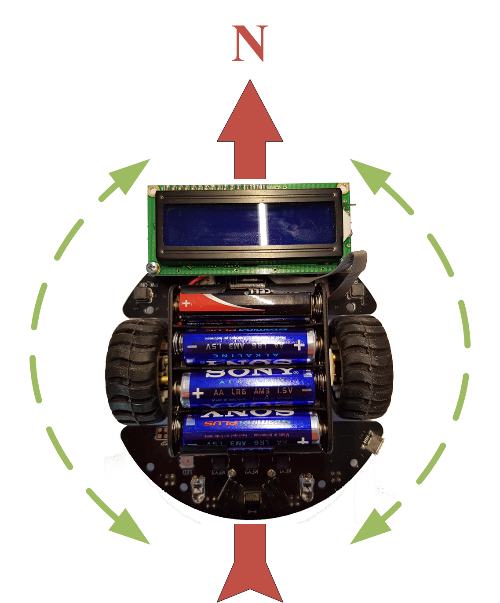


Abbildung : Funktion MiniQ als Kompass

## Vorgehen

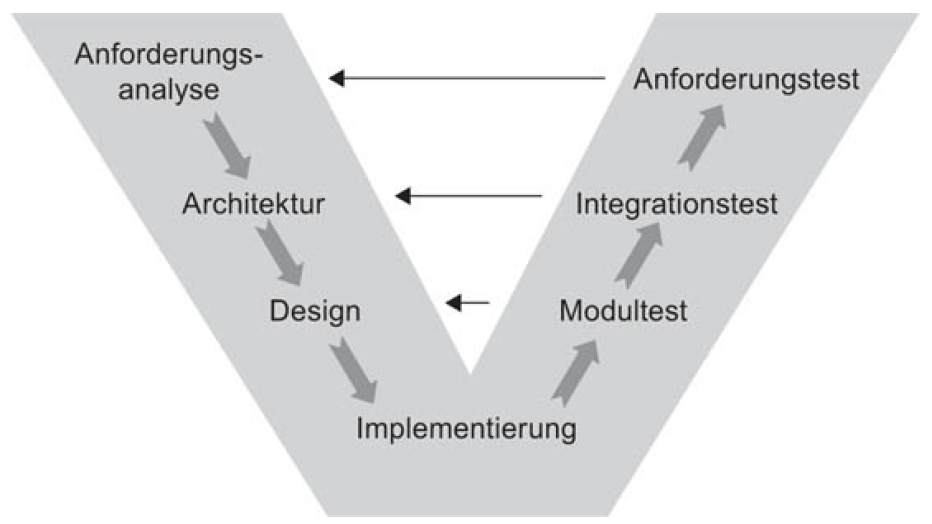
Das allgemeine Vorgehen für diese Projektarbeit ist in Abbildung 2 ersichtlich.

Abbildung : V-Diagramm

In dieser Dokumentation sind die 4 abzugebenden Dokumente vereint:

1. Architektur-Dokument
2. Design-Dokument
3. Testspezifikation und Testprotokoll
4. Programm-Code

### Implementierung

Die Implementierung erfolgt mittels folgenden Tools:

* Microsoft Visual Studio Enterprise 2017, *Version 15.5.3*
* Arduino Extension for Visual Studio: Visual Micro *V1711.19*
* GitHub

Das GitHub wird als Versionenkontrolle verwendet, um möglichst effizient im Team arbeiten zu können.

Das GitHub kann unter folgender URL abgerufen und ggf. heruntergeladen / geklont werden:

<https://github.com/fhnwAlex/fhnwEbs.git>

## Funktionsbeschreibung

Der MiniQ wird zu Beginn auf eine ebene Fläche gelegt. Über den «Power-On»-Knopf [**1**] (vgl. Abbildung 3) wird der MiniQ eingeschaltet. Auf dem LCD-Display [**2**] wird man aufgefordert über die Taste 1 [**3**] die Kompasskalibration zu starten. Wird die Taste 1 [**3**] gedrückt läuft die Kompasskalibration diese dauert 20 Sekunden. Während der Kalibration wird dessen Status auf dem LCD-Display [**2**] angezeigt und das integrierte LED [**4**] blinkt weiss. Die erste 10 Sekunden fährt der MiniQ im Uhrzeigersinn und die letzten im Gegenuhrzeigersinn auf der Stelle. Ist die Kalibration beendet werden auf dem LCD-Display [**2**] die beiden Modi (manuell und automatisch) zur Auswahl angezeigt. Mit der Taste 1 [**3**] wird der manuelle Modus und mit der Taste 2 [**5**] der automatische Modus aktiviert.

Im manuellen Modus wird der aktuelle Winkel mit dem 3-Achsen Kompass HMC5883L [**6**] gemessen und auf dem LCD-Display [**2**] angezeigt. Als Richtung gilt das LCD-Display [**2**] der Winkel wird gemäss Abbildung 4 anzeigt. Der MiniQ kann von Hand dann in die gewünschte Position gebracht werden. Des Weiteren wird über die beiden Fotodioden [**7**] die aktuelle Lichtstärke in Volt auf dem LCD-Display [**2**] angezeigt. Das integrierte LED [**4**] wechselt seine Farbe von rot, gelb und grün gemäss Abbildung 4. Mit der Taste 3 [**8**] wird der Modus abgebrochen und man kehrt in das Auswahlmenu der beiden Modi zurück.

Im automatischen Modus wird ebenfalls der aktuelle Winkel mit dem 3-Achsen Kompass HMC5883L [**6**] gemessen und auf dem LCD-Display [**2**] angezeigt. Der MiniQ richtet sich selbständig mit seinen beiden Motoren [**9**] nach Norden aus. Liegt der Winkel zwischen 180° und 350° dreht der MiniQ im Uhrzeigersinn, liegt der Winkel zwischen 10° und 180° im Gegenuhrzeigersinn. Das integrierte LED [**4**] wechselt seine Farbe von rot, gelb und grün gemäss Abbildung 4. Mit der Taste 3 [**8**] wird der Modus abgebrochen und man kehrt in das Auswahlmenu der beiden Modi zurück.

Der MiniQ wird über Batterien [**10**] oder ein langes USB-Kabel [**11**] versorgt.

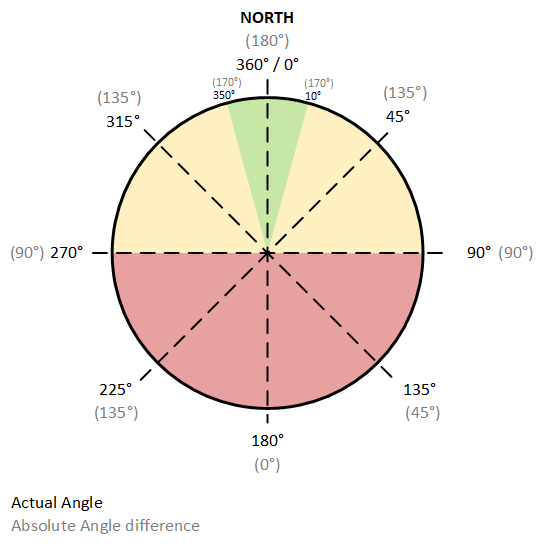
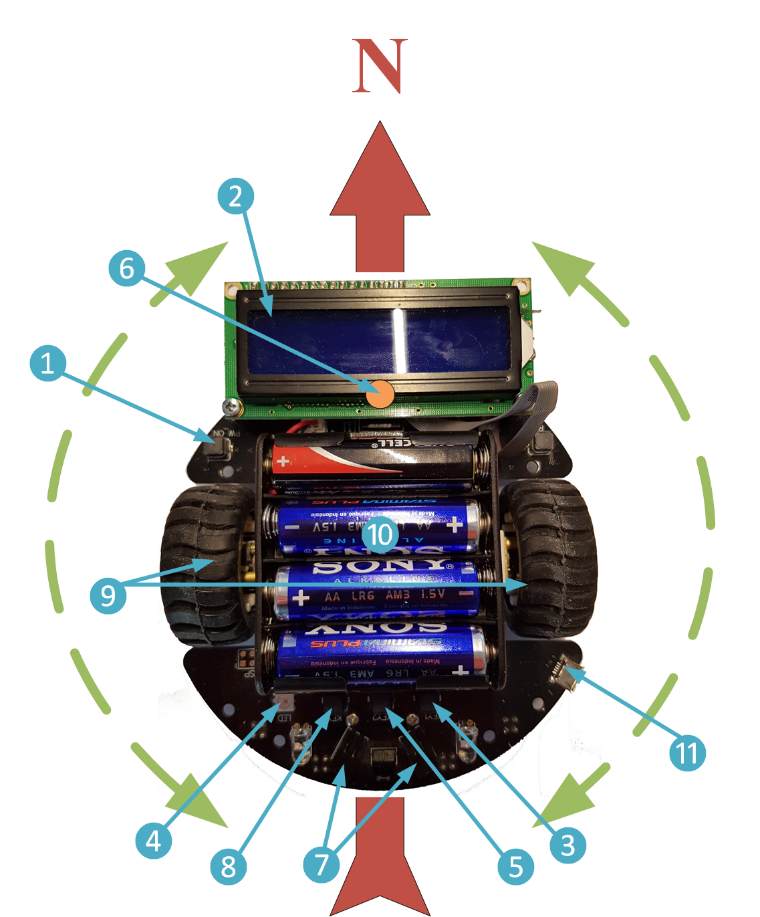


Abbildung : MiniQ mit Positionsnummern Abbildung : Winkelwerte und LED Farben

# Entwurf

## Modularisierung

Um die erforderlichen Aufgaben erfolgreich absolvieren zu können, muss die Software-Struktur modularisiert werden. Bei unserer Applikation haben wir uns für eine möglichst flache Hierarchische Modularisierung entschieden. Ein wesentlicher Grund für diesen Entscheid liegt in der Übersichtlichkeit der Gesamtstruktur. Aus diesem Entscheid folgt, dass ein Modul (Functions) implementiert wird, in welchem alle Methoden erarbeitet werden.

Unserer Ansicht nach ist so die Software viel besser les- und interpretierbar. Dazu hinaus wäre eine stark modularisierte Struktur über den Anforderungen der Aufgabenstellung.

### Hauptprogramm

Das Hauptprogramm dient als Betriebssystem für das Embedded System. In unserem Hauptprogramm werden alle Methoden, welche für das Absolvieren der Aufgaben notwendig sind, zyklisch aufgerufen.

### Menügeführte Benutzerschnittstelle

Die Benutzerschnittstelle besteht aus den Tasten KEY1, KEY2 und KEY3. Ausserdem dient ein 2x16 Zeichen LCD zur Darstellung von aktuellen Informationen.

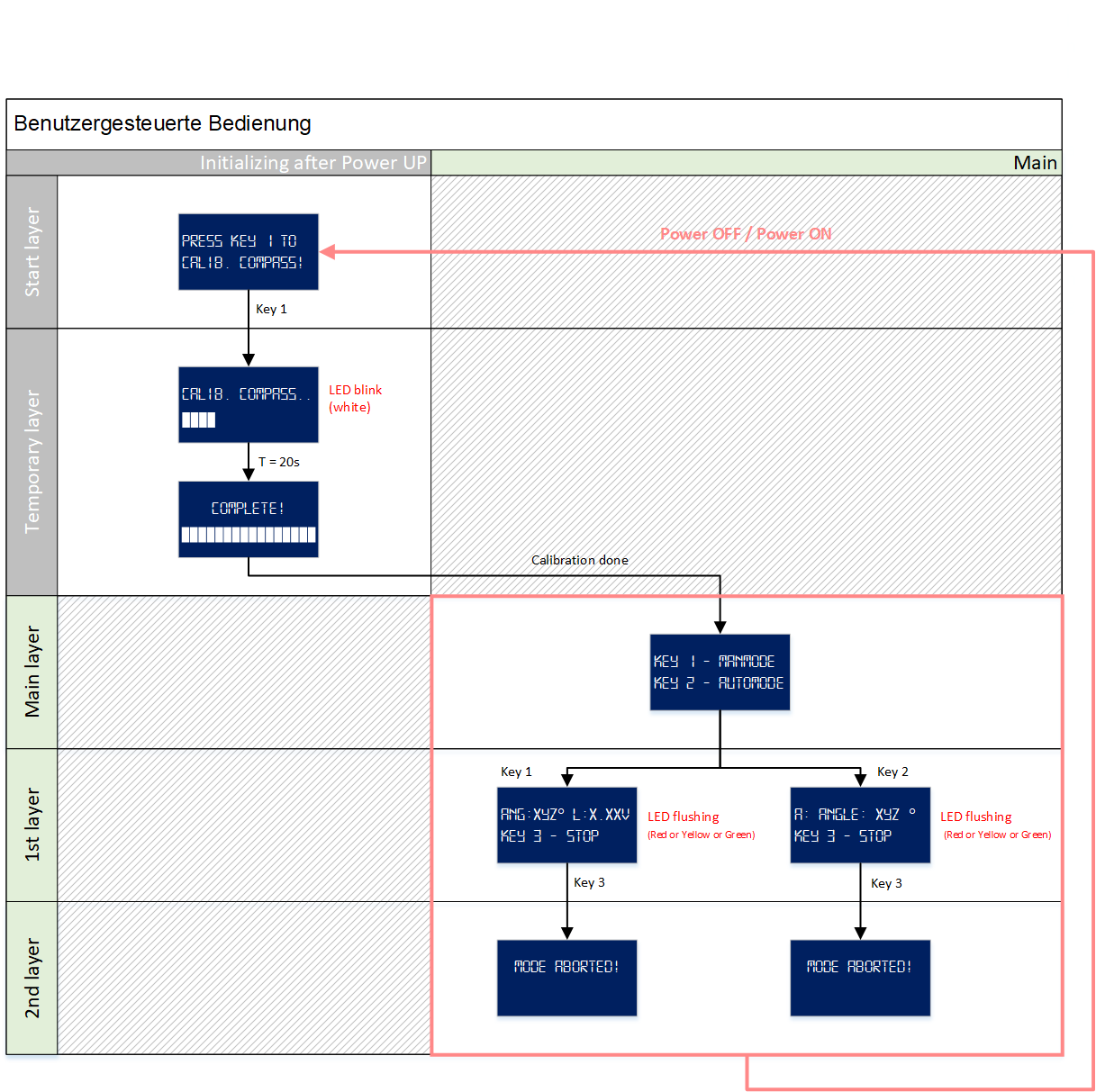
Zur übersichtlichen Darstellung ist das Menü sehr einfach gehalten. Die Menüführung ist wie folgt aufgebaut:

Abbildung : Benutzergesteuerte Bedienung

Zu Beginn muss bei jedem Neustart des MiniQ-Roboters der Kompass neu kalibriert werden. Dies führt dazu, dass das Menü in zwei Bereiche aufgeteilt wird. Im Bereich Initialisierung wird der Benutzer dazu aufgefordert durch das Drücken der KEY1-Taste den Kompass zu kalibrieren. Nach dem Drücken wird dem Benutzer der aktuelle Status der Kalibration laufend angezeigt, bis jene nach 20 Sekunden beendet ist. Danach wird dem Benutzer die Hauptebene im zweiten Bereich (Main) visualisiert. Dort kann ausgewählt werden, ob die Applikation im manuell- (KEY1) oder im automatischen Modus (KEY2) gestartet werden soll. Im manuellen Modus wird dem Benutzer der aktuelle Winkel des MiniQ-Roboters zu Norden und die aktuelle Spannung der Fotodioden angezeigt. Im automatischen Modus wird nur der aktuelle Winkel des MiniQ-Roboters zu Norden visualisiert. In beiden Modis werden zusätzliche Bedienmöglichkeiten (KEY3) angezeigt. Wird die KEY3-Taste betätigt so wird dem Benutzer gezeigt, dass der aktuelle Fahrmodus abgebrochen ist und das Menü gelangt wieder auf die Hauptebene zurück.

### Initialisierung

Dieses Programm ist dazu da, alle Hardware-Komponente, jegliche Pins und verschiedene Speicherzugriffe zu initialisieren.

### Kompass

Der Kompass wird für die Winkelermittlung verwendet. Weiter wird die Kalibration des Kompasses in diesem Modul realisiert. Ohne diese Kalibrierung kann die Applikation nicht verwendet und gestartet werden. Des Weiteren kann die Applikation während des Kalibration Prozesses **nicht** abgebrochen werden.

### Motorsteuerung

Die Motorsteuerung steuert die Motoren anhand des Winkels an. Die Geschwindigkeit ist abhängig von der Winkeldifferenz zwischen der aktuellen Position und Norden (0°).

### RGB-LED

Das RGB-LED zeigt sowohl im manuellen- wie auch im automatischen Modus den Winkelbereich an in welchem sich der MiniQ-Roboter befindet. Weiter wird das RGB-LED während dem Kalibrierungsprozess benötigt.

### Benutzerinteraktion

Der Benutzer kann über die Tasten **KEY1 bis KEY3** den Ablauf des MiniQ-Programms beeinflussen. In regelmässigen Abständen liest das Programm „*UI*“ mithilfe des Unterprogrammes „*Keys*“ die Benutzerinteraktionen aus.

Über das LCD-Display erhält der Benutzer entweder Informationen über den aktuellen Winkel oder der zusätzlichen Lichtintensität, welche in Spannung umgerechnet wird:

Um zum einen die Hierarchie möglichst flach zu halten und zum anderen der Übersichtlichkeitshalber werden alle Methoden in einer Bibliothek (functions) implementiert. Diese wird danach ins Hauptprogramm „*ebs2WDMiniQ.ino*“ eingebunden.

## Modulhierarchie

Abbildung : Modulhierarchie

## Schnittstellendefinition

### ebs2WDMiniQ

**Modul:**

Dieses Programm ist das Hauptprogramm und übernimmt folgende Aufgaben:

**Methoden:**

Im Main befinden sich die Standard Arduino Funktionen:

|  |
| --- |
| **void setup()** |

Die Setup Funktion wird einmal zu Beginn (Nach Powerup oder nach Reset) ausgeführt und dient der Initialisierung.

|  |
| --- |
| **void loop()** |

Die Loop Funktion wird dauernd (nach Setup) ausgeführt und dient der Programmausführung. In der loop()-Methode werden jene Methoden, welche zyklisch abgearbeitet werden müssen, aufgerufen.

### Functions

**Modul:**

Das Modul *Functions* stellt alle Funktionen welche für die Aufgabe benötigt werden zu Verfügung.

Verwendende Librarys:

|  |
| --- |
| **#include <math.h>**  **#include "Wire.h"**  **#include "I2Cdev.h"**  **#include <HMC5883L.h>**  **#include <LiquidCrystal\_I2C.h>**  **#include <Adafruit\_NeoPixel.h>** |

Verwendete bestehende Methoden des Kompasses: HMC5883L.h, I2Cdev.h

|  |
| --- |
| **void initialize()**  **void getHeading(int16\_t \*x, int16\_t \*y, int16\_t \*z)** |

Verwendete bestehende Methoden des LCD-Displays: LiquidCrystal\_I2C.h

|  |
| --- |
| **void begin()**  **void clear()**  **void home()**  **void backlight()**  **void lcd.print(char[] text)**  **void createChar(uint8\_t, uint8\_t[])**  **void setCursor(uint8\_t, uint8\_t)** |

Verwendete bestehende Methoden des RGB-LED’s: Adafruit\_NeoPixel.h

|  |
| --- |
| **void begin()**  **void show()**  **void clear()**  **void setPixelColor(uint16\_t n, uint8\_t r, uint8\_t g, uint8\_t b),**  **void setBrightness(uint8\_t),** |

**Methoden:**

|  |
| --- |
| **void finitUp(tstPrvMain \*pstPrivate)** |

In dieser Methode werden alle Hardware-Komponenten initialisiert. Als Argument wird die komplette Private-Struktur der Methode mitgegeben. Unter anderem werden Speicherzugriffe oder auch die Schnittstellen genannt, zwischen den betreffenden Methoden (siehe Abbildung 7) bereitgestellt.

Diese Methode wird einmalig beim Aufstarten des Roboters in der setup-Methode ausgeführt.



Abbildung : Strukturzugriff im initUp

|  |
| --- |
| **void fMoveProcedure(tstMotor \*pstMotor)** |

Diese Methode hat als Argument die komplette Motoren-Struktur. Alle Aufgaben welche dazu führen, dass die Motoren angesteuert werden müssen, werden in dieser Methode umgesetzt. Folgend werden alle Möglichkeiten der Ansteuerung aufgezeigt:

Kalibrierung:

* Statische Geschwindigkeit
* Rechst drehen
* Links drehen

Auto-Mode:

* Dynamische Geschwindigkeit (abhängig des Winkels)
* Drehrichtung anhand der kürzesten Distanz

Die dynamische Geschwindigkeitsberechnung wird mittels folgender Formel realisiert:

Formel : Geschwindigkeitsberechnung

|  |
| --- |
| **void fsetColor(tstPrvMain \*pstPrivate)** |

Diese Methode hat als Argument die komplette Private-Struktur. Der Grund dazu ist, dass das RGB-LED für verschiedene Aufgaben benötigt wird. In der Methode werden die verschiedenen Modis des LED’s umgesetzt. Folgend werden alle Möglichkeiten der Ansteuerung aufgezeigt:

Kalibrierung: (Zugriff auf die Motorenstruktur)

* Während der gesamten Kalibrierung blinkt die LED (weiss)

Manual-Mode und Auto-Mode: (Zugriff auf die User Interface-Struktur)

* Die LED zeigt den Status der absoluten Winkeldifferenz an welches dazu führt, dass die Farben in einem Bereich von 0° bis 180° den kompletten Umfang abdecken
* Wie die Farben angezeigt werden kann aus der Abbildung 8 entnommen werden

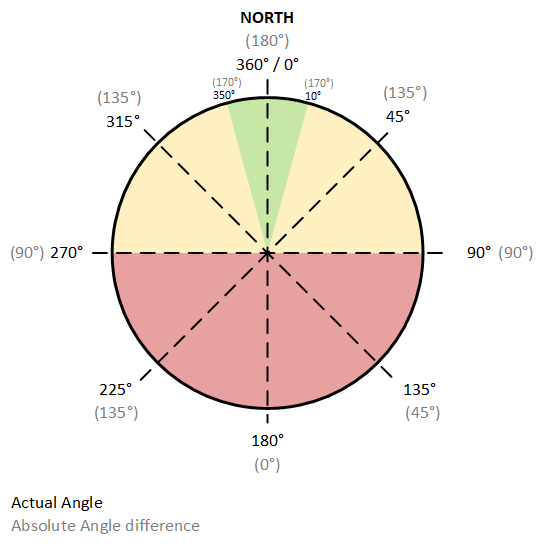


Abbildung : Statusanzeige der RGB-LED

|  |
| --- |
| **void fcompassCalibrate(tstPrvMain \*pstPrivate)** |

Diese Methode hat als Argument die komplette Private-Struktur. In dieser Methode wird die ganze Kalibrierung des Kompasses realisiert. Da sich der MiniQ-Roboter bei der Kalibrierung dreht, ist die Motoren-Struktur notwendig.

Beim einmaligen Aufrufen dieser Methode wird der MiniQ im Uhrzeigersinn und im Gegenuhrzeigersinn gedreht, gleichzeitig werden die die X und die Y-Komponenten aus dem Kompass gelesen und nach dem Minimalen und Maximalen Wert eingeteilt. Zum Schluss werden die Offsets in X und Y gebildet und in die Kompass-Struktur geschrieben.

Während der Kalibration zeigt das LCD-Display den aktuellen Status der Kalibration in Form eines Balkens dar. Der MiniQ kann während der Kalibration keine weiteren Aufgaben erledigen.

|  |
| --- |
| **void fgetAngle(tstCompass \*pstCompass)** |

Diese Methode hat als Argument die komplette Kompass-Struktur. Die Methode liest zyklisch die X und Y-Werte des Kompasses aus. Weiter wird der Offset, welcher aus der Kalibration berechnet wurde, mit den aktuellen Werten verrechnet. Anschliessend werden die Werte von Radiant in Grad gerechnet und in einer Variable gespeichert.

|  |
| --- |
| **void fgetLight(tstLightSensor \*pstLight)** |

Diese Methode liest, wenn der Manuelle Modus angewählt ist, zyklisch den analogen Pin aus, an welchen die zwei Fotodioden angeschlossen sind. Anschliessend wird der Wert nachfolgender Formel in die Spannung umgerechnet:

Formel : Spannung der Fotodioden

|  |
| --- |
| **unsigned short fgetKeyValue(tstUI \*pstUI)** |

Diese Methode gibt den gedrückten Hardwaretaster zurück:

* 0: Keine Taste
* 1: Key 1
* 2: Key 2
* 3: Key 3

|  |
| --- |
| **void fUIProcedure(tstPrvMain \*pstPrivate)** |

Diese Methode hat als Argument die gesamte Private-Struktur. Sie dient zur Ablaufsteuerung der einzelnen Strukturen wie den Motoren, den Kompass und des LCD-Displays. Bei dieser Methode wird zyklisch auf die Key-Events geachtet und dem entsprechend reagiert.

|  |
| --- |
| **void fsetUIMenu(tstPrvMain \*psttPrivate)** |

Diese Methode hat als Argument die gesamte Private-Struktur und ist zuständig für die gesamte Visualisierung der LCD-Texte. Der Grund für den kompletten Strukturzugriff ist, dass die Methode der Kompasskalibrierung in dieser Methode aufgerufen wird. Der Aufruf dieser Methode erfolgt zyklisch.

|  |
| --- |
| **void fDisplayProcedure(tstUI \*pstDisplay)** |

Diese Methode hat als Argument die Struktur des User Interfaces. Weiter ist diese Methode für das Echtzeitverhalten der Applikation verantwortlich. Durch den Zugriff auf die UI-Struktur ist es der Methode möglich auf den Daten Buffer der Applikation zuzugreifen. Diese Methode dient als Treiber, welcher für das Beschreiben des LCD-Displays verantwortlich ist.

In der folgenden Abbildung ist vereinfacht dargestellt, wie das Echtzeitverhalten der Applikation durch den Treiber implementiert ist:

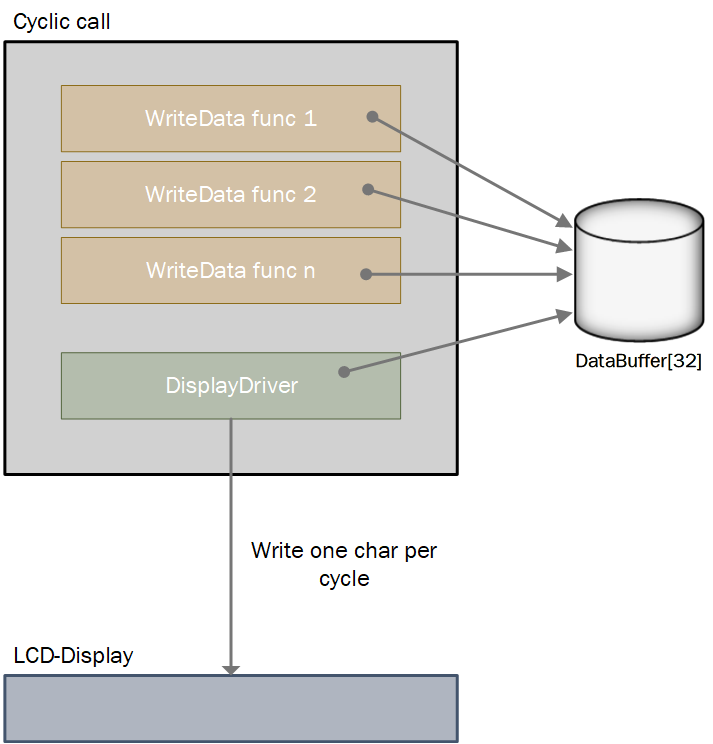


Abbildung : Display-Treiber

Durch das Zyklische beschreiben des Daten Buffers kann der Display-Treiber zyklisch die Charakter aus dem Buffer lesen und anschliessend mittels *lcd.print()-*Funktion an das LCD-Display senden. Durch diesen Treiber ist die Übermittlung eines einzelnen Charakters pro Zyklus möglich. Auch andere LCD-Funktionen wie zum Beispiel die *lcd.setCursor()-*Funktion, wird separat in einem Zyklus an das Display gesendet.

|  |
| --- |
| **void fWriteString(tstPrvMain \*pstPrivate, char szStringLine[], unsigned char uchLcdRow)** |

Diese Methode weist als Argumente die gesamte Private-Struktur, den zu beschreibende String wie auch der Index, auf welche Reihe der String auf das LCD-Display geschrieben werden soll, auf. Durch diese Methode werden nur Strings verarbeitet, welche statisch sind.

|  |  |  |  |
| --- | --- | --- | --- |
| **void fWriteSingleValue(tstPrvMain \*pstPrivate, char szStringLine[], unsigned int uiValue,** | | | |
|  | **unsigned char uchLcdRow)** |  |  |

Diese Methode weist als Argumente die gesamte Private-Struktur, den zu beschreibende String, ein dynamischen Wert, wie auch der Index, auf welche Reihe der String auf das LCD-Display geschrieben werden soll, auf. Zudem wird der Wert welcher der Methode mitgegeben wird in Charakter konvertiert, welche anschliessend in den String eingesetzt und danach in den Buffer geschrieben werden. Mittels des Index kann der Methode mitgeteilt werden, in welchen Bereich des Buffers der String geschrieben werden soll.

|  |  |  |
| --- | --- | --- |
| **void fWriteDoubleValue(tstPrvMain \*pstPrivate, char szStringLine[], unsigned int uiValue\_1,** | | |
|  | **float flValue\_2, unsigned char uchLcdRow)** |  |

Diese Methode weist als Argumente die gesamte Private-Struktur, den zu beschreibende String, zwei dynamische Werte, wie auch der Index, auf welche Reihe der String auf das LCD-Display geschrieben werden soll, auf. Beide Werte welche der Methode mitgegeben werden, werden in Charakter konvertiert, welche anschliessend in den String eingesetzt und danach in den Buffer geschrieben werden. Mittels des Index kann der Methode mitgeteilt werden, in welchen Bereich des Buffers der String geschrieben werden soll.

# Testspezifikation

## Testspezifikationen der Module in «functions.cpp»

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **Funktion** | **Methode** | **Ausgangslage** | **Ereignis** | **Vorgang** | **Ergebnis Soll** | **Ergebnis Ist** |
| Lichtstärke lesen | fgetLight | MiniQ auf ebenem Grund. | Aufruf Methode «fgetLight» durch «ebs2WDMiniQ» | Lichtstärke wird über analogen Eingang gemessen. | Wert liegt zwischen 0 und 5 V. | Vgl. Kapitel 4.1 |
| Winkel messen | fgetAngle | MiniQ auf ebenem Grund. | Aufruf Methode «fgetAngle» durch «ebs2WDMiniQ» | Winkel wird berechnet. | Wert liegt zwischen 0 und 360° | Vgl. Kapitel 4.1 |
| Kalibration Kompass | fcompassCalibrate | MiniQ auf ebenem Grund. | Aufruf Methode «fcompassCalibrate» durch «ebs2WDMiniQ» | Offsets x und y werden berechnet. | Offset x und Offset y werden im Speicher gesetzt. | Vgl. Kapitel 4.1 |
| LED Farbe ändern | fsetColor | MiniQ auf ebenem Grund. | Aufruf Methode «fsetColor» durch «ebs2WDMiniQ» | LED wird grün leuchten. | LED leuchtet grün | Vgl. Kapitel 4.1 |
| Taste einlesen | fgetKeyValue | MiniQ auf ebenem Grund. | Aufruf Methode «fgetKeyValue» durch «ebs2WDMiniQ» | Taste 1 wird gedrückt und eingelesen. | Taste 1 wird eingelesen | Vgl. Kapitel 4.1 |
| Rechts drehen (Uhrezeigersinn) | fMoveProcedure | MiniQ auf ebenem Grund. | Aufruf Methode «fMoveProcedure» durch «ebs2WDMiniQ» | Winkel von Hand zwischen 0 und 180° setzen. | MiniQ dreht rechts herum.(Uhrzeigersinn) | Vgl. Kapitel 4.1 |
| Links drehen (Gegenuhrzeigersinn( | fMoveProcedure | MiniQ auf ebenem Grund. | Aufruf Methode «fMoveProcedure» durch «ebs2WDMiniQ» | Winkel von Hand zwi-schen 180 und 360° setzen. | MiniQ dreht links herum.(Gegenuhrzeigersinn) | Vgl. Kapitel 4.1 |

Tabelle : Testspezifikationen der Methoden in «functions.cpp»

## Globale Testspezifikationen des Gesamtsystems

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **Funktion** | **Objekt** | **Ausgangslage** | **Ereignis** | **Vorgang** | **Ergebnis Soll** | **Ergebnis Ist** |
| Warten auf Kalibration | «ebs2WDMiniQ» | MiniQ auf ebenem Grund. MiniQ eingeschaltet. Programm gestartet. | LCD wird beschrieben. | Nach PowerOn wird die Kalibrations-Aufforderung auf dem LCD-Display angezeigt. | Auf dem LCD wird man aufgefordert die Kalibration zu starten. | Vgl. Kapitel 4.3 |
| Kalibration läuft | «ebs2WDMiniQ» | MiniQ auf ebenem Grund. Taste 1 wurde gedrückt. | Taste 1 wird gedrückt. Magnetisches Feld wird gemessen, Offsets werden berechnet. | Kalibration ermittelt den x und y Offset für den Kompass. | Die Kalibration läuft durch, das beenden der Kalibration wird bestätigt auf dem LCD. | Vgl. Kapitel 4.3 |
| Menu erscheint mit Auswahl der Modi | «ebs2WDMiniQ» | MiniQ hat Kalibration absolviert. | LCD wird beschrieben. | LCD wird mit Auswahlmenu der Modi beschrieben. | Auf dem LCD-Display wird die Auswahl der Modi angezeigt. | Vgl. Kapitel 4.3 |
| Auswahl manueller Modus | «ebs2WDMiniQ» | Menu mit Auswahl der Modi wird angezeigt. | Taste 1 wird gedrückt. | Taste wird erfasst Modus wird gestartet. | Betätigung Taste 1 wird erfasst. Automatischer Modus wird gestartet. | Vgl. Kapitel 4.3 |
| Betrieb manueller Modus | «ebs2WDMiniQ» | Taste 1 wurde im Menu der Modi gedrückt. | Messung Lichtstärke, Berechnung Winkel, LCD wird beschrieben. | Lichtstärke wird gemessen, Winkel wird berechnet, Informationen werden auf dem LCD-Display angezeigt. | Manueller Modus läuft und zeigt alle relevanten Werte auf dem LCD-Display an. | Vgl. Kapitel 4.3 |
| Manueller Modus beenden | «ebs2WDMiniQ» | Manueller Modus läuft. | Taste 3 wird gedrückt. | Betätigung Taste 3 wird erfasst. Modus wird abgebrochen. LCD wird beschrieben. | Betätigung Taste 3 wird erfasst, manueller Modus wird abgebrochen und bestätigt auf LCD. LCD zeigt anschliessend das Modi Auswahlmenu. | Vgl. Kapitel 4.3 |
| Auswahl automatischer Modus | «ebs2WDMiniQ» | Menu mit Auswahl der Modi wird angezeigt. | Taste 2 wird gedrückt. | Taste wird erfasst Modus wird gestartet. | Betätigung Taste 2 wird erfasst, automatischer Modus wird gestartet | Vgl. Kapitel 4.3 |

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **Funktion** | **Objekt** | **Ausgangslage** | **Ereignis** | **Vorgang** | **Ergebnis Soll** | **Ergebnis Ist** |
| Betrieb automatischer Modus | «ebs2WDMiniQ» | MiniQ auf ebenem Grund. Taste 2 wurde im Menu der Modi gedrückt. | Berechnung Winkel, Ansteuerung Motoren, LCD wird beschrieben. | Winkel wird berechnet, Informationen werden auf dem LCD-Display angezeigt. | Automatischer Modus läuft und zeigt alle relevanten Werte auf dem LCD-Display an. | Vgl. Kapitel 4.3 |
| Automatischer Modus beenden | «ebs2WDMiniQ» | Automatischer Modus läuft. | Taste 3 wird gedrückt. | Betätigung Taste 3 wird erfasst. Modus wird abgebrochen. LCD wird beschrieben. | Betätigung Taste 3 wird erfasst, automatischer Modus wird abgebrochen und bestätigt auf LCD. LCD zeigt anschliessend das Modi Auswahlmenu. | Vgl. Kapitel 4.3 |

Tabelle : Globale Testspezifikationen des Gesamtsystems

## Testspezifikationen Systemanforderungen

Gemäss Aufgabenstellung «PA\_Aufgabenstellung\_mit\_Notizen.pdf» gibt es einige formale Anforderungen, bezüglich Zeitverhalten welche das System einhalten muss diese sollen ebenfalls überprüft werden.

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **Anforderung** | **Objekt** | **Ausgangslage** | **Ereignis** | **Vorgang** | **Ergebnis Soll** | **Ergebnis Ist** |
| **9.** Diese Verarbeitung nach 4. und 5. darf auf keinen Fall durch irgendeinen anderen Prozess für länger als **2ms** unterbrochen oder gestört werden, mit Ausnahme von Anforderung 10. | «ebs2WDMiniQ» | MiniQ läuft im automatischen Modus. | Betrieb automatischer Modus(Aufgabe 2) | Messung der Zeit vom Beenden der Funktion «fMoveProcedure»(Zyklus n) bis «fgetAngle»(Zyklus n+1). | Zwischen dem ansteuern der Motoren in Zyklus n und dem Messen und berechnen des Winkels vom Kompass in Zyklus n+1, dürfen maximal 2ms vergehen. | Vgl. Kapitel 4.2 |
| **10.** Jede Aufgabe (nach Anf. 2. bis Anf. 5) können jederzeit vom Benutzer beendet oder abgebrochen werden. (Innerhalb **4ms** nach Tastendruck) | «ebs2WDMiniQ» | Messung 1: MiniQ läuft im manuellen Modus.  Messung 2: MiniQ läuft im automatischen Modus. | Zyklus des Gesamtsystems messen. | Messung 1: Messung der Gesamtzykluszeit im manuellen Modus (Aufgabe 1)  Messung 2: Messung der Gesamtzykluszeit im automatischen Modus (Aufgabe 2) | Da die Tasten jeden Zyklus ausgewertet werden, darf die maximale Zykluszeit des Gesamtsystems für diese Anforderung maximal 4ms betragen. | Vgl. Kapitel 4.2 |
| **12.** Refreshrate Display **<= 250ms** | «ebs2WDMiniQ» | MiniQ läuft, egal ob in einem Modus oder im Menu. | Zyklus des Gesamtsystems messen. | Messung der Zykluszeit und Berechnung der Display Refreshrate. | Das Display soll zu jedem Zeitpunkt eine Refreshrate von <= 250ms haben. | Vgl. Kapitel 4.2 |

Tabelle : Testspezifikationen Systemanforderungen

# Abnahmetests

## Abnahmetest der Module in «functions.cpp»

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **Funktion** | **Methode** | **Ausgangslage** | **Ereignis** | **Vorgang** | **Ergebnis Soll** | **Ergebnis Ist** |
| Lichtstärke messen | fgetLight | MiniQ auf ebenem Grund. | Aufruf Methode «fgetLight» durch «ebs2WDMiniQ» | Lichtstärke wird über analogen Eingang gemessen. | Wert liegt zwischen 0 und 5 V. | OK |

Aufruf für Test:

tstPrvMain \*pstPrivate = &stPrivate;

void loop()

{

// Testaufruf für Modultest "Lichtstärke messen"

fgetLight(&stPrivate.stLight);

Serial.println(pstPrivate->stLight.flLightInVoltage);

delay(1000);

}

Resultat: OK

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **Funktion** | **Methode** | **Ausgangslage** | **Ereignis** | **Vorgang** | **Ergebnis Soll** | **Ergebnis Ist** |
| Winkel messen | fgetAngle | MiniQ auf ebenem Grund. | Aufruf Methode «fgetAngle» durch «ebs2WDMiniQ» | Winkel wird berechnet. | Wert liegt zwischen 0 und 360° | 120° -> Bedingt OK |

Aufruf für Test:

tstPrvMain \*pstPrivate = &stPrivate;

void loop()

{

// Testaufruf für Modultest "Winkel messen"

fgetAngle(&stPrivate.stCompass);

Serial.println(pstPrivate->stCompass.uiAngle);

delay(1000);

}

Resultat: 120° -> Bedingt OK

Wegen der fehlenden Kalibration, bzw. errechneten Offsets, des Kompasses stimmt der Wert nicht mit der Richtung überein.

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **Funktion** | **Methode** | **Ausgangslage** | **Ereignis** | **Vorgang** | **Ergebnis Soll** | **Ergebnis Ist** |
| Kalibration Kompass | fcompassCalibrate | MiniQ auf ebenem Grund. | Aufruf Methode «fcompassCalibrate» durch «ebs2WDMiniQ» | Offsets x und y werden berechnet. | Offset x und Offset y werden im Speicher gesetzt. | OK |

Aufruf für Test:

tstPrvMain \*pstPrivate = &stPrivate;

void loop()

{

// Testaufruf für Modultest "Kompass Kalibration"

fcompassCalibrate(&stPrivate);

Serial.print(pstPrivate->stCompass.iMagOffset\_x);

Serial.print('\t');

Serial.println(pstPrivate->stCompass.iMagOffset\_y);

delay(10000);

}

Resultat: OK, Offset x= 249, Offset y= 270

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **Funktion** | **Methode** | **Ausgangslage** | **Ereignis** | **Vorgang** | **Ergebnis Soll** | **Ergebnis Ist** |
| LED Farbe ändern | fsetColor | MiniQ auf ebenem Grund. | Setzen der verschiedenen Winkel und Aufruf Methode «fsetColor» durch «ebs2WDMiniQ» | LED wird zuerst grün dann gelb und danach rot leuchten. | LED leuchtet grün, gelb und rot | OK |

Aufruf für Test:

tstPrvMain \*pstPrivate = &stPrivate;

void loop()

{

// Testaufruf für Modultest "LED Farbe ändern"

pstPrivate->stUI.bStartManual = true;

pstPrivate->stCompass.uiAngle = 0;

fsetColor(&stPrivate);

delay(2000);

pstPrivate->stCompass.uiAngle = 80;

fsetColor(&stPrivate);

delay(2000);

pstPrivate->stCompass.uiAngle = 180;

fsetColor(&stPrivate);

delay(2000);

}

Resultat: OK

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **Funktion** | **Methode** | **Ausgangslage** | **Ereignis** | **Vorgang** | **Ergebnis Soll** | **Ergebnis Ist** |
| Taste einlesen | fgetKeyValue | MiniQ auf ebenem Grund. | Aufruf Methode «fgetKeyValue» durch «ebs2WDMiniQ» | Tasten 1,2 und 3 wird gedrückt und eingelesen. | Alle Tasten sollen erkannt werden. | OK |

Aufruf für Test:

tstPrvMain \*pstPrivate = &stPrivate;

void loop()

{

// Testaufruf für Modultest "Tasten auslesen"

Serial.println(fgetKeyValue(&stPrivate.stUI));

delay(1000);

}

Resultat: OK

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **Funktion** | **Methode** | **Ausgangslage** | **Ereignis** | **Vorgang** | **Ergebnis Soll** | **Ergebnis Ist** |
| Rechts drehen (Uhrezeigersinn) | fMoveProcedure | MiniQ auf ebenem Grund. | Aufruf Methode «fMoveProcedure» durch «ebs2WDMiniQ» | Winkel von Hand zwischen 0 und 180° setzen. | MiniQ dreht rechts herum.(Uhrzeigersinn) | OK |

Aufruf für Test:

tstPrvMain \*pstPrivate = &stPrivate;

void loop()

{

// Testaufruf für Modultest "MiniQ rechts drehen"

pstPrivate->stMotor.bCompassCalibrated = true;

pstPrivate->stMotor.bRun = true;

pstPrivate->stCompass.uiAngle = 270;

fMoveProcedure(&stPrivate.stMotor);

}

Resultat: OK

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **Funktion** | **Methode** | **Ausgangslage** | **Ereignis** | **Vorgang** | **Ergebnis Soll** | **Ergebnis Ist** |
| Links drehen (Gegenuhrzeigersinn( | fMoveProcedure | MiniQ auf ebenem Grund. | Aufruf Methode «fMoveProcedure» durch «ebs2WDMiniQ» | Winkel von Hand zwi-schen 180 und 360° setzen. | MiniQ dreht links herum.(Gegenuhrzeigersinn) | OK |

Aufruf für Test:

tstPrvMain \*pstPrivate = &stPrivate;

void loop()

{

// Testaufruf für Modultest "MiniQ links drehen"

pstPrivate->stMotor.bCompassCalibrated = true;

pstPrivate->stMotor.bRun = true;

pstPrivate->stCompass.uiAngle = 90;

fMoveProcedure(&stPrivate.stMotor);

}

Resultat: OK

## Abnahmetest der Systemanforderungen

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **Anforderung** | **Objekt** | **Ausgangslage** | **Ereignis** | **Vorgang** | **Ergebnis Soll** | **Ergebnis Ist** |
| **9.** Diese Verarbeitung nach 4. und 5. darf auf keinen Fall durch irgendeinen anderen Prozess für länger als **2ms** unterbrochen oder gestört werden, mit Ausnahme von Anforderung 10. | «ebs2WDMiniQ» | MiniQ läuft im automatischen Modus. | Betrieb automatischer Modus(Aufgabe 2) | Messung der Zeit vom Beenden der Funktion «fMoveProcedure»(Zyklus n) bis «fgetAngle»(Zyklus n+1). | Zwischen dem ansteuern der Motoren in Zyklus n und dem Messen und berechnen des Winkels vom Kompass in Zyklus n+1, dürfen maximal 2ms vergehen. | Nicht erfüllt |

**Vorgehen**:

Es wurde mit dem Code unter Anhang 6.2, jeweils mehrmals 1000 Messungen der Zykluszeit durchgeführt. Mittels *Serial.println()* wurde die maximale und minimale Zykluszeit dieser 1000 Messungen ausgegeben. Die Messdaten können unter Anhang 6.1 eingesehen werden. In Abbildung 10 wird die Zeitmessung nochmals verdeutlicht:

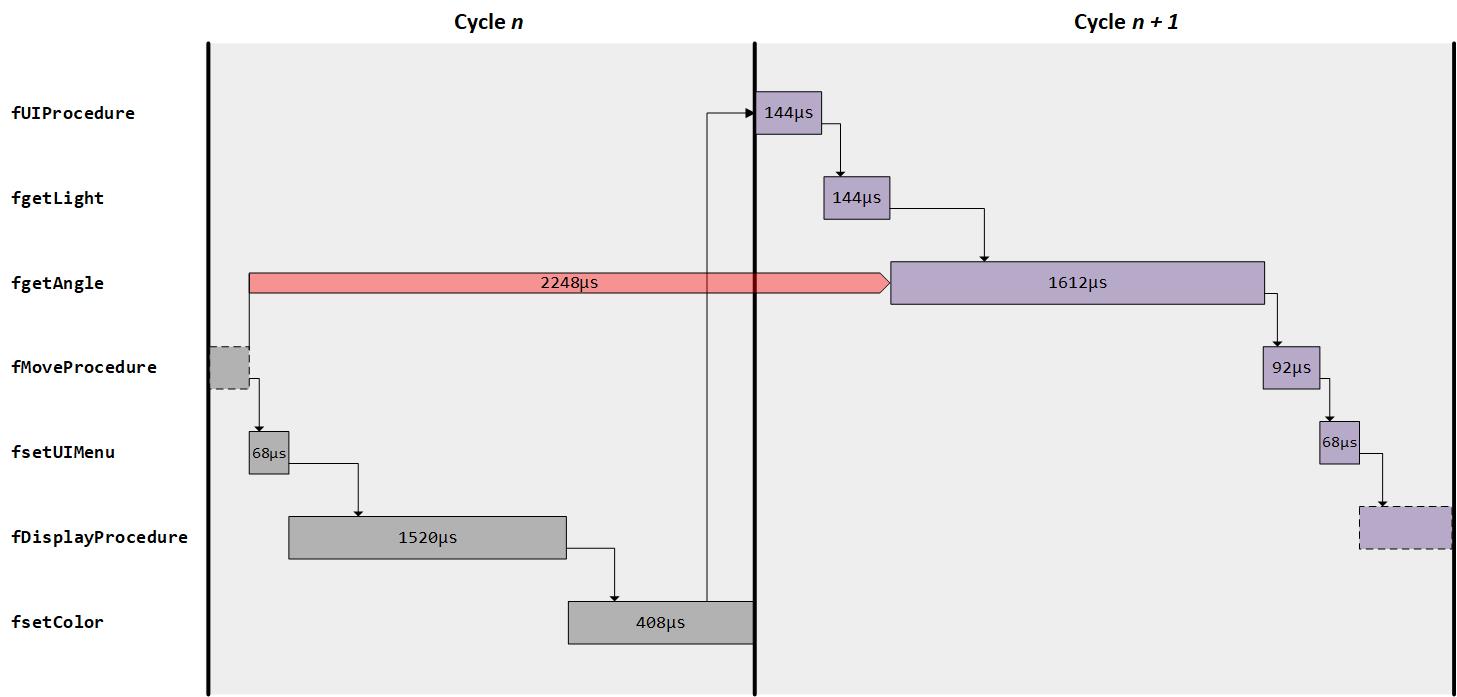


Abbildung : Zeitmessung Anforderung 9

**Resultat**: Nicht erfüllt.

Wie aus den Messdaten unter Anhang 6.1 zu entnehmen ist, beträgt die Zeit zwischen dem ansteuern der Motoren «fMoveProcedure» in Zyklus **n** und dem Messen und berechnen des Winkels vom Kompass «fgetAngle» in Zyklus **n+1**, maximal ca. **2.3ms**.

**Optimierung**:

Diese Anforderung kann durchaus erreicht werden, und zwar in dem wir in der Aufgabe 2 die LED nicht ansteuern, dies ist auch keine Bedingung. Die LED benötigt im Betrieb minimal **0.3ms** das würde bedeuten, dass die neue Zeit dann (2.3ms – 0.3ms) **2.0ms** betragen würde. Somit wäre die Anforderung erfüllt. Wir haben auf diese Anpassung verzichtet da die LED ein netter Zusatz ist und die Zeitbedingung nur um 0.3ms verletzt wird.

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **Anforderung** | **Objekt** | **Ausgangslage** | **Ereignis** | **Vorgang** | **Ergebnis Soll** | **Ergebnis Ist** |
| **10.** Jede Aufgabe (nach Anf. 2. bis Anf. 5) können jederzeit vom Benutzer beendet oder abgebrochen werden. (Innerhalb **4ms** nach Tastendruck) | «ebs2WDMiniQ» | Messung 1: MiniQ läuft im manuellen Modus.  Messung 2: MiniQ läuft im automatischen Modus. | Zyklus des Gesamtsystems messen. | Messung 1: Messung der Gesamtzykluszeit im manuellen Modus (Aufgabe 1)  Messung 2: Messung der Gesamtzykluszeit im automatischen Modus (Aufgabe 2) | Da die Tasten jeden Zyklus ausgewertet werden, darf die maximale Zykluszeit des Gesamtsystems für diese Anforderung maximal 4ms betragen. | erfüllt |

**Vorgehen**:

Es wurde mit dem Code unter Anhang 6.2, jeweils mehrmals 1000 Messungen der Zykluszeit durchgeführt. Mittels *Serial.println()* wurde die maximale und minimale Zykluszeit dieser 1000 Messungen ausgegeben. Die Messdaten können unter Anhang 6.1 eingesehen werden. In Abbildung 11 Abbildung 11: Messung Zykluszeitwird die Zeitmessung nochmals verdeutlicht:

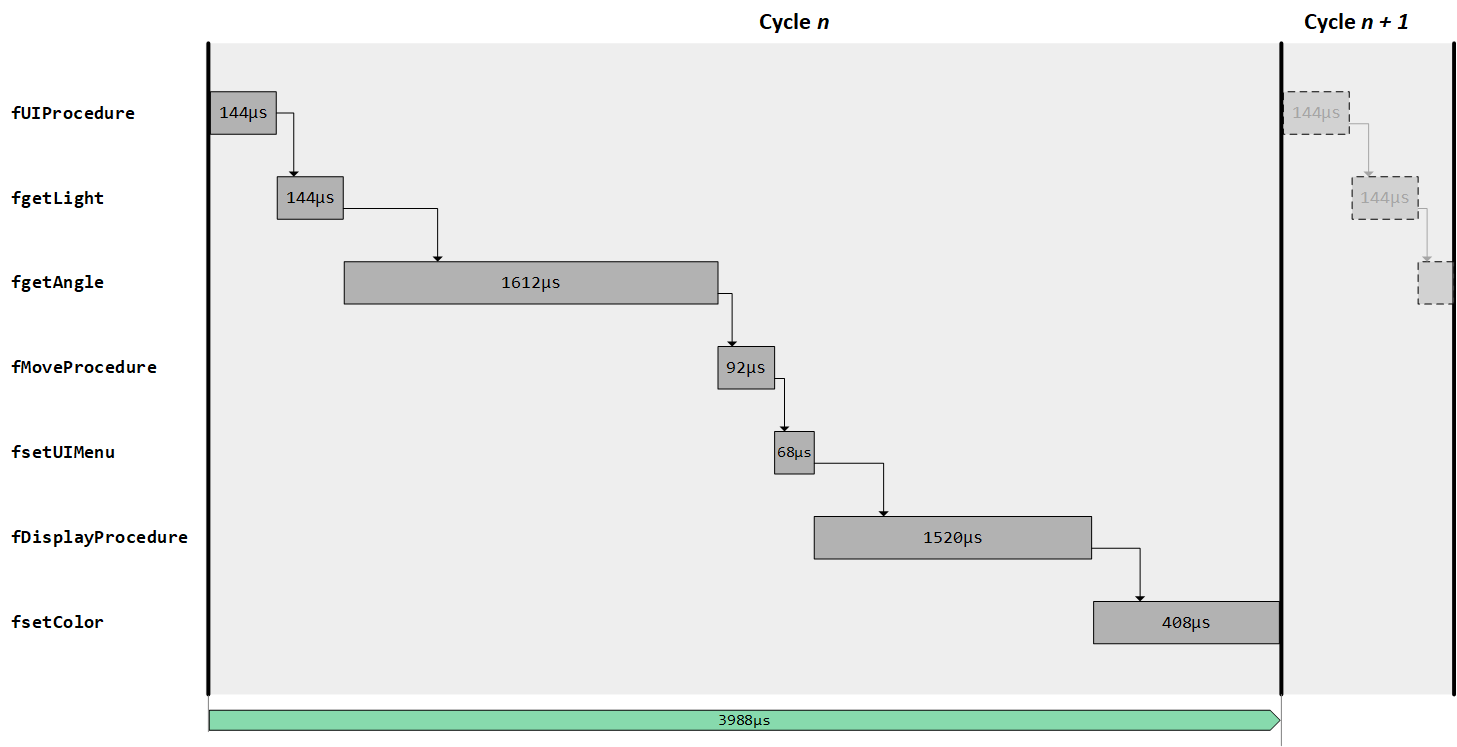


Abbildung : Messung Zykluszeit

**Resultat**: erfüllt

Wie aus den Messdaten unter Anhang 6.1 zu entnehmen ist, beträgt die maximale Zykluszeit in allen Modi <= **4ms**. Dies bedeutet, dass die Tasten ausgewertet und das Programm abgebrochen wird zyklisch innerhalb von 4ms.

**Optimierung**:

Da das Resultat positiv ist sind zurzeit keine Optimierungen geplant.

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **Anforderung** | **Objekt** | **Ausgangslage** | **Ereignis** | **Vorgang** | **Ergebnis Soll** | **Ergebnis Ist** |
| **12.** Refreshrate Display **<= 250ms** | «ebs2WDMiniQ» | MiniQ läuft, egal ob in einem Modus oder im Menu. | Zyklus des Gesamtsystems messen. | Messung der Zykluszeit und Berechnung der Display Refreshrate. | Das Display soll zu jedem Zeitpunkt eine Refreshrate von <= 250ms haben. | erfüllt |

**Vorgehen:**

Auswertung der vorhandenen Messdaten.

**Resultat**: erfüllt

Jeder Befehl an das LCD-Display benötigt minimum **ca.** **1.5ms**. Wie aus der Funktion fDisplayProcedure zu entnehmen ist wird deshalb pro Zyklus nur ein Befehl an das LCD-Display geschickt. Das heisst um das LCD-Display komplett neu zu laden benötigt man:

* 2 x *setCursor()* Befehle für obere und untere Zeile
* 2 x 16 *print()* Befehle, da jede Zeile 16 Zeichen besitzt

Somit benötigt man 34 Zyklen um das Display zu aktualisieren. Wenn wir unsere maximale Zykluszeit von 4ms einsetzten, liegt die Refresh-Rate des Displays bei:

Formel : Refresh-Rate Display

**Optimierung**:

Da das Resultat positiv ist sind zurzeit keine Optimierungen geplant.

## Abnahmetest des Gesamtsystems

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **Funktion** | **Objekt** | **Ausgangslage** | **Ereignis** | **Vorgang** | **Ergebnis Soll** | **Ergebnis Ist** |
| Warten auf Kalibration | «ebs2WDMiniQ» | MiniQ auf ebenem Grund. MiniQ eingeschaltet. Programm gestartet. | LCD wird beschrieben. | Nach PowerOn wird die Kalibrations-Aufforderung auf dem LCD-Display angezeigt. | Auf dem LCD wird man aufgefordert die Kalibration zu starten. | OK |

Resultat: OK

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **Funktion** | **Objekt** | **Ausgangslage** | **Ereignis** | **Vorgang** | **Ergebnis Soll** | **Ergebnis Ist** |
| Kalibration läuft | «ebs2WDMiniQ» | MiniQ auf ebenem Grund. Taste 1 wurde gedrückt. | Taste 1 wird gedrückt. Magnetisches Feld wird gemessen, Offsets werden berechnet. | Kalibration ermittelt den x und y Offset für den Kompass. | Die Kalibration läuft durch, das beenden der Kalibration wird bestätigt auf dem LCD. | OK |

Resultat: OK

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **Funktion** | **Objekt** | **Ausgangslage** | **Ereignis** | **Vorgang** | **Ergebnis Soll** | **Ergebnis Ist** |
| Menu erscheint mit Auswahl der Modi | «ebs2WDMiniQ» | MiniQ hat Kalibration absolviert. | LCD wird beschrieben. | LCD wird mit Auswahlmenu der Modi beschrieben. | Auf dem LCD-Display wird die Auswahl der Modi angezeigt. | OK |

Resultat: OK

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **Funktion** | **Objekt** | **Ausgangslage** | **Ereignis** | **Vorgang** | **Ergebnis Soll** | **Ergebnis Ist** |
| Auswahl manueller Modus | «ebs2WDMiniQ» | Menu mit Auswahl der Modi wird angezeigt. | Taste 1 wird gedrückt. | Taste wird erfasst Modus wird gestartet. | Betätigung Taste 1 wird erfasst. Automatischer Modus wird gestartet. | OK |

Resultat: OK

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **Funktion** | **Objekt** | **Ausgangslage** | **Ereignis** | **Vorgang** | **Ergebnis Soll** | **Ergebnis Ist** |
| Betrieb manueller Modus | «ebs2WDMiniQ» | Taste 1 wurde im Menu der Modi gedrückt. | Messung Lichtstärke, Berechnung Winkel, LCD wird beschrieben. | Lichtstärke wird gemessen, Winkel wird berechnet, Informationen werden auf dem LCD-Display angezeigt. | Manueller Modus läuft und zeigt alle relevanten Werte auf dem LCD-Display an. | OK |

Resultat: OK

Hinweis: Wichtig ist, dass der Kompass möglichst wenig Störungen, wie z.B. metallische Gegenstände, ausgesetzt ist. Am besten funktioniert der Kompass draussen im Freien.

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **Funktion** | **Objekt** | **Ausgangslage** | **Ereignis** | **Vorgang** | **Ergebnis Soll** | **Ergebnis Ist** |
| Manueller Modus beenden | «ebs2WDMiniQ» | Manueller Modus läuft. | Taste 3 wird gedrückt. | Betätigung Taste 3 wird erfasst. Modus wird abgebrochen. LCD wird beschrieben. | Betätigung Taste 3 wird erfasst, manueller Modus wird abgebrochen und bestätigt auf LCD. LCD zeigt anschliessend das Modi Auswahlmenu. | OK |

Resultat: OK

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **Funktion** | **Objekt** | **Ausgangslage** | **Ereignis** | **Vorgang** | **Ergebnis Soll** | **Ergebnis Ist** |
| Auswahl automatischer Modus | «ebs2WDMiniQ» | Menu mit Auswahl der Modi wird angezeigt. | Taste 2 wird gedrückt. | Taste wird erfasst Modus wird gestartet. | Betätigung Taste 2 wird erfasst, automatischer Modus wird gestartet | OK |

Resultat: OK

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **Funktion** | **Objekt** | **Ausgangslage** | **Ereignis** | **Vorgang** | **Ergebnis Soll** | **Ergebnis Ist** |
| Betrieb automatischer Modus | «ebs2WDMiniQ» | MiniQ auf ebenem Grund. Taste 2 wurde im Menu der Modi gedrückt. | Berechnung Winkel, Ansteuerung Motoren, LCD wird beschrieben. | Winkel wird berechnet, Informationen werden auf dem LCD-Display angezeigt. | Automatischer Modus läuft und zeigt alle relevanten Werte auf dem LCD-Display an. | OK |

Resultat: OK

Hinweis: Wichtig ist, dass der Kompass möglichst wenig Störungen, wie z.B. metallische Gegenstände, ausgesetzt ist. Am besten funktioniert der Kompass draussen im Freien.

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **Funktion** | **Objekt** | **Ausgangslage** | **Ereignis** | **Vorgang** | **Ergebnis Soll** | **Ergebnis Ist** |
| Automatischer Modus beenden | «ebs2WDMiniQ» | Automatischer Modus läuft. | Taste 3 wird gedrückt. | Betätigung Taste 3 wird erfasst. Modus wird abgebrochen. LCD wird beschrieben. | Betätigung Taste 3 wird erfasst, automatischer Modus wird abgebrochen und bestätigt auf LCD. LCD zeigt anschliessend das Modi Auswahlmenu. | OK |

Resultat: OK

## Geplante Optimierungen

Es sind Optimierungen vorgesehen, diese sind in Abschnitt 4.2 zu finden.

# Ehrlichkeitserklärung

«Hiermit erklären wir, die vorliegende Dokumentation selbständig, ohne Hilfe Dritter und nur unter Benutzung der angegebenen Quellen verfasst zu haben»

Herr Alex Gessler, 12.01.2018, Windisch

Herr Bruno Hürzeler, 12.01.2018, Windisch

# Anhang

## Messdaten

Siehe nachfolgende Seiten.

## Code für Messungen

Siehe nachfolgende Seiten.

## Code

Siehe nachfolgende Seiten.